

Adabas for UNIX, Windows and OpenVMS

Adabas Utilities

Version 6.3.1

April 2013

This document applies to Adabas for UNIX, Windows and OpenVMS Version 6.3.1.

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

Copyright © 1987-2013 Software AG, Darmstadt, Germany and/or Software AG USA, Inc., Reston, VA, United States of America, and/or their licensors.

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: ADAOS-UTILITIES-631-20130422

Table of Contents

Adabas Utilities	vii
1 Conventions	1
Use of character fonts	2
Syntax conventions	2
Upper-Case Conversions	4
Symbols used in control parameter summaries	6
Order of parameters	6
Numeric Values	6
Maximum Values	10
Syntax diagrams in the HTML documentation	7
Obsolete Parameters	7
2 Overview	9
3 ADABAS (Starting The Database Nucleus)	15
Functional Overview	16
Procedure Flow	17
Control Parameter	18
4 ADABCK (Dump And Restore Database Or Files)	19
Functional Overview	20
Procedure Flow	22
Checkpoints	24
Control Parameters	25
Restart Considerations	37
5 ADACLP (Command Log Report)	39
Functional Overview	40
Procedure Flow	41
Checkpoints	42
Control Parameters	42
Specifying Multiple Selection Criteria	48
6 ADACMP (Compression Of Data)	49
Functional Overview	50
Procedure Flow	51
Checkpoints	52
Control Parameters	52
Output	64
Report	65
Restart Considerations	65
7 ADADBA (DBA Workbench)	67
Functional Overview	68
Procedure Flow	69
8 ADADBM (Database Modification)	71
Functional Overview	72
Procedure Flow	74
Checkpoints	76

Control Parameters	77
Restart Considerations	100
9 ADADCU (Decompression Of Data)	101
Functional Overview	102
Procedure Flow	103
Checkpoints	104
Control Parameters	105
Input and Output Data	113
Restart Considerations	114
10 ADADEV (Disk Space Management)	115
Functional Overview	116
Procedure Flow	117
Checkpoints	118
Control Parameters	118
11 ADAERR (Error File Report)	127
Functional Overview	128
Procedure Flow	129
Checkpoints	129
Control Parameter	129
Example	130
Rejected Data Records	130
12 ADAFDU (File Definition)	131
Functional Overview	132
Procedure Flow	133
Checkpoints	135
Control Parameters	135
Examples	149
13 ADAFIN (File Information Report)	151
Functional Overview	152
Procedure Flow	153
Checkpoints	154
Control Parameters	154
14 ADAFRM (Format And Create A New Database)	167
Functional Overview	168
Procedure Flow	169
Checkpoints	170
Control Parameters	170
Restart Considerations	174
Control Statement Examples	174
15 ADAINV (Creating, Removing And Verifying Inverted Lists)	175
Functional Overview	176
Procedure Flow	177
Checkpoints	178
Control Parameters	179
Restart Considerations	188

Examples	189
16 ADAMON (Monitoring The Database Nucleus)	195
Functional Overview	196
Procedure Flow	197
Checkpoints	197
Control Parameters	198
17 ADAMUP (Mass Add And Delete)	203
Functional Overview	204
Procedure Flow	205
Checkpoints	208
Control Parameters	209
Restart Considerations	215
SORT Data Set Placement	215
TEMP Data Set Placement	215
Examples	216
18 ADANUC (Starting The Database, Defining Nucleus Parameters)	217
Functional Overview	218
Procedure Flow	220
Checkpoints	222
Control Parameters	222
Summary of ADANUC Parameters	241
19 ADAOPR (Operator Utility)	245
Functional Overview	246
Procedure Flow	247
Checkpoints	248
Control Parameters	248
20 ADAORD (Reorder Database Or Files, Export/Import Files)	285
Functional Overview	286
Procedure Flow	287
Checkpoints	289
Control Parameters	289
Restart Considerations	298
Examples	298
21 ADAPLP (Protection Log Printout)	301
Functional Overview	302
Procedure Flow	303
Checkpoints	304
Control Parameters	304
ADAPLP Output	313
22 ADAPRI (Print Adabas Blocks)	317
Functional Overview	318
Procedure Flow	319
Checkpoints	320
Control Parameters	320
23 ADAREC (Recovery Of Database Or Files)	323

Functional Overview	324
Procedure Flow	325
Checkpoints	327
ADAREC Input Data	327
Control Parameters	327
Examples	333
ADAREC Restart Considerations	340
24 ADAREP (Database Report)	341
Functional Overview	342
Procedure Flow	343
Checkpoints	343
Control Parameters	344
25 ADASCR (Security Functions)	357
Functional Overview	358
Procedure Flow	359
Checkpoints	360
Control Parameters	360
26 ADATST (Issuing Adabas Commands)	377
Functional Overview	378
Procedure Flow	379
Checkpoints	379
Control Parameters	380
27 ADAULD (File Unloading)	395
Functional Overview	396
Procedure Flow	398
Checkpoints	400
Control Parameters	401
Examples	407
TEMP Data Set Space Estimation	408
Restart Considerations	408
28 ADAVFY (Database Consistency Check)	409
Functional Overview	410
Procedure Flow	411
Checkpoints	412
Control Parameters	412
Examples	417
A Appendix A - Example Utility Input Files	419
B Appendix B - prilogc	423

Adabas Utilities

This manual describes the Adabas utilities. The database administrator (DBA) uses the Adabas utilities to create and maintain Adabas databases. For each utility, the following information is provided:

- a description of the purpose of the utility;
- a functional overview of the utility;
- a description of the utility's control parameters;
- examples to illustrate the use of the utility, where appropriate.

This manual is intended principally for the DBA. Certain Adabas utilities contain functionality for modifying or deleting existing database information, so caution is advised when these utilities are used. Some utilities, such as ADAREP, provide status information only, and can be used freely by the end user.



Note: The Adabas utilities also contain some undocumented features that can be invoked using syntax that is not documented (this also includes the FDT syntax as described in the *Administration* documentation). Software AG strongly recommends that you do not use such undocumented features; there is no guarantee that undocumented features will work correctly and that they will not have negative side effects on the general behaviour of Adabas.

The *Overview* provides a summary of the utilities available and their purpose.

The subsequent documents describe the individual utilities in detail, with one utility per document.

Appendix A contains a description of the demo utility input files provided with the Adabas kit.

Appendix B contains a description of the example program `prilogc`, which is used for printing a command log that is created with the nucleus parameter `CLOGLAYOUT` set to 6.

1 Conventions

▪ Use of character fonts	2
▪ Syntax conventions	2
▪ Upper-Case Conversions	4
▪ Symbols used in control parameter summaries	6
▪ Order of parameters	6
▪ Numeric Values	6
▪ Maximum Values	10
▪ Syntax diagrams in the HTML documentation	7
▪ Obsolete Parameters	7

The following conventions have been used in this manual:

Use of character fonts

References to other manuals are shown in *italics*.

References to documents or sections within documents are shown in *bold face*.

Examples of utility output and file contents are shown in a typewriter font, for example:

```
%ADADBМ-I-OPENED, ds DATA2, file DATA2.001 opened
%ADADBМ-F-DSSTALL, allocation error DSST
```

In examples which show both user input and utility output, the typewriter font is used for the whole example:

```
adadbm: add_container = data, size = 35
%ADADBМ-I-OPENED, ds DATA2, file DATA2.001 opened
%ADADBМ-F-DSSTALL, allocation error DSST
```

Syntax conventions

The syntax of the utility control parameters is as follows.

Items shown in UPPERCASE letters are keywords and must be entered exactly as shown. You can enter any keyword with uppercase or lowercase letters.

Items shown in lowercase letters indicate that you have to replace the item by a value of your choice. If the item is "number", you can specify any decimal number. Only positive numbers or 0 can be specified, no negative numbers are allowed. If the item is "string", you can specify a text string, i.e. any number of alphanumeric characters. For numbers or strings, it is also possible to specify hexadecimal values preceded by "0x", "0X". "^x" or "^X"; for numbers specified as hexadecimal values, leading zeroes may be omitted. Other items are possible, for example "descriptor", in which case your input must be a descriptor name.

Items enclosed in square brackets ("[" , "]") are optional.

Items enclosed in curly brackets ("{" , "}") are mandatory.

A vertical bar ("|") separates items which are alternatives, i.e. you can enter one item or the other but not both.

The ellipsis ("...") indicates that you can repeat the immediately preceding element of the syntax as often as you like.

If an ellipsis is preceded by a comma, i.e. ",...", this means that you can repeat the immediately preceding element of the syntax as often as you like, with a comma preceding each repetition.

If round brackets are used for a list of elements and only one keyword is supplied, the round brackets may be omitted if only one element is supplied.

Example 1

```
RB=0x33445566
```

Here the string value has been specified as a hexadecimal string; it consists of printable characters equivalent to the ASCII character string "3DUf".

Example 2

```
DBID = number
```

This means that you must type in the keyword "DBID" (using uppercase or lowercase letters or any combination thereof), followed by the "=" character, followed by a decimal number, for example:

```
DBID = 27
```

Example 3

```
RABN = number [ - number ]
```

This means that you must type in the keyword "RABN" (using uppercase or lowercase letters or any combination thereof), followed by the "=" character, followed by a decimal number. You can also provide a hyphen ("-") followed by another decimal number, but this is not required. Here are a few examples of input that corresponds to this syntax:

```
RABN = 25
RABN = 1000 - 1125
```

Example 4

```
RABN = 0x400
```

Here the value is specified as a hexadecimal value; it is the equivalent to specifying

```
RABN = 1024
```

Example 5

```
SORTSEQ = { descriptor | ISN }
```

This means that you must type in the keyword "SORTSEQ" (using uppercase or lowercase letters or any combination thereof), followed by the "=" character, followed by either a descriptor value or the keyword "ISN", for example:

```
SORTSEQ = ISN
```

Example 6

```
number[-number] [,number[-number] ] ...
```

This example shows the use of the ellipsis ("..."). Here, the ellipsis follows the syntax element "[number[-number]]". This means that you can repeat this syntax element as often as you like in your input line. Here are a few examples of input that corresponds to this syntax:

```
27
27-50
27-50,68
27,68-90
27-50,68-90
27-50,68-90,102,105,118-140,160
```

Example 7

As an alternative to example 4, the following syntax specification using the "... " construction is also possible:

```
{ number[-number] },...
```

This syntax allows all of the combinations shown in example 4.

Example 8

```
(number[,number]...)
```

Valid input examples for this syntax are.

```
(12,23,45)
(123)
123 - only one list element, so brackets can be omitted
```

Upper-Case Conversions

Parameter names that are specified are always converted to upper case.

For most utility control parameters, the specified parameter values are also converted to upper case, but this is not always desirable. Starting with Adabas Version 6.1.6, a new convention for upper-case conversion of control parameter values has been introduced:

- If you specify "*parameter=value*", the value is converted to upper case.
- If you specify "*parameter:value*", the value is not converted to upper case.

Although the option of specifying a colon or an equals sign after a parameter name has been introduced generally via the parser for all parameters, Software AG recommends that you specify a colon only for those parameters where it is explicitly described in the syntax, because the behaviour described above is only guaranteed for these parameters; due to compatibility reasons with previous Adabas versions, the upper-case conversion is handled differently for some other parameters.

Example

Assume the following syntax:

```
NAME{=|:}string
```

If you specify

```
NAME=Production
```

The parameter value for NAME is set to "PRODUCTION".

If you specify

```
NAME:Production
```

The parameter value for NAME is set to "Production".

However, some utility input is not provided as "parameter{=|:}value", for example field, descriptor or referential constraint definitions. The specifications are converted to upper case by default, unless the parameter LOWER_CASE_FIELD_NAMES has been specified before the definitions.



Note: After specifying LOWER_CASE_FIELD_NAMES, none of the definitions are converted to upper case; then you must specify keywords, for example field options, in upper case.

Example

Without LOWER_CASE_FIELD_NAMES, the field definition

```
1,aa,8,a,de
```

is correct and is equivalent to

```
1,AA,8,A,DE
```

With LOWER_CASE_FIELD_NAMES, however, this field definition is invalid; in order to define the field "aa", you must specify

```
1,aa,8,A,DE
```

Symbols used in control parameter summaries

The description of each utility contains a table which summarizes the syntax of the control parameters that are available for that utility. Some control parameters are preceded by the letter "M" or the letter "D".

The letter "M" indicates "mandatory", i.e. the you must specify this parameter in your input to the utility otherwise the utility cannot run. If the letter "M" is not present, the parameter is optional, i.e. you do not have to specify it.

The letter "D" indicates that the control parameter has a default value. This means that if you do not specify this parameter explicitly in your input, the utility will use a preset value for the parameter.

Order of parameters

The parameters of the utilities are listed in this documentation in alphabetical order, but in some cases, there are restrictions on the order in which they can or must be specified. Usually, the DBID parameter has to be specified first, and depending on the utility, there may be more restrictions.

Numeric Values

Numeric values may be specified in the following ways:

- number
- number[K], where the value is $1024 * \text{number}$
- number[M], where the value is $1024 * 1024 * \text{number}$

number[K] and number[M] are only allowed in cases in which large numeric values are expected.

Maximum Values

Maximum values for numeric parameters are only mentioned if there is a fixed limit that is given by restrictions within Adabas. They are not mentioned if they result from the fact that a 4 byte signed or unsigned integer is used to store the variable: in this case, the limit may be defined a little smaller than the maximum possible integer, for example 4000 M.

Syntax diagrams in the HTML documentation

There is a syntax diagram at the start of each utility description, and these diagrams contain links to the detailed descriptions of the keywords and parameters that are available. The hyperlinks in these syntax diagrams are underlined in order to make them visible, but please note that the underlining is not a part of the syntax.

Obsolete Parameters

Sometimes, utility or nucleus parameters will become obsolete when a new version of Adabas is released. Usually, the obsolete parameters are still accepted by the utility of the nucleus, but you will receive a PAROBS warning, for example:

```
%ADANUC-W-PAROBS, parameter NH has become obsolete
```


2 Overview

This chapter gives an overview of the Adabas utilities, which provide all of the functions necessary to manage an Adabas database.

ADABAS

Start the database nucleus (for Windows only)

This utility starts the database nucleus with the required environment.

ADABCK

Backup and restore database or files

The Adabas backup utility dumps/restores the contents of the database (or a specific file or files) to/from a sequential data file. The utility can also be used to copy an Adabas backup copy.

ADACLPL

Command log report

This utility prints the command log.

ADACMP

Compression of data

The compression utility compresses user data. The compressed data is used as input for the mass update utility ADAMUP. The input for this utility is the raw data together with the data definitions that describe the structure of the data provided.

ADADBA

DBA Workbench

This utility provides functionality for operating and maintaining Adabas databases and files via a graphical user interface.

ADADBM

Database modification

The ADADBM utility consists of the following functions which can be used to make modifications to the database:

- The ADD_CONTAINER function adds a new container file to the Associator or Data Storage data set;
- The ADD_FIELDS function appends one or more new fields to the end of a file's FDT;
- The ALLOCATE functions increase the Normal Index, Upper Index, Address Converter or Data Storage space assigned to a file. The DEALLOCATE functions are the inverse;
- The CHANGE function changes the standard length of a field in the FDT;
- The CHANGE_FIELDS function changes a field definition;
- The DEFINE_REFINT function defines a new referential constraint;
- The DELCP function deletes old checkpoint records from the checkpoint file in the specified range of dates;
- The DELETE function deletes an Adabas file or a range of files from the database;
- The DISPLAY function displays the UCB;
- The DROP_FIELDS function marks the specified fields as not existing, which means that they can no longer be accessed;
- The DROP_LOBFILE function is the inverse function of ADAFDU ADD_LOBFILE;
- The DROP_REFINT function drops an existing referential constraint;
- The EXTEND_CONTAINER function extends the last container file defined for the database;
- The NEW_DBID function changes the identifier of the database in use;
- The NEWWORK function allocates and formats a new Adabas WORK data set;
- The PGM_REFRESH function is used to disable or enable refreshing an Adabas file inside an application program with an E1 command;
- The RECOVER function returns lost space to the FST;
- The REDUCE_CONTAINER function reduces the size of the last container file defined for the database;
- The REFRESH function resets a file or a range of files to the state of zero records loaded;
- The REMOVE_CONTAINER function deletes a container file;
- The REMOVE_DROP function, used in conjunction with a subsequent REFRESH, removes dropped fields from the FDT;
- The REMOVE_REPLICATION function stops all replication processing and deletes the replication system files;
- The RENAME function changes the database name or the name of a loaded file;
- The RENUMBER function renumbers a loaded file or exchanges the numbers of loaded files;

- The REPLICATION_FILES function creates the systems files required for Adabas - Adabas replication;
- The RESET function removes entries from the UCB;
- The RESET_REPLICATION_TARGET function resets the replication target flag of Adabas files;
- The REUSE function controls the reuse of data storage space or ISNs by Adabas;
- The SYFMAX function specifies the maximum number of values generated for a system generated multiple-value field in the file specified.

ADADCUC

Decompression of data

The ADADCUC utility decompresses records to be used with a non-Adabas application program, or as input for the compression utility ADACMP. The file to be decompressed must be unloaded from the database (unload utility ADAULD) before it can be used as input for this utility. With ADADCUC, complete records can be decompressed, fields can be rearranged within a record, default lengths can be changed, some types of fields can be truncated, formats can be changed and space can be allocated for the addition of new fields.

ADADEV

Disk space management (UNIX only)

This utility consists of several functions for managing the disk space to be used by Adabas. It can be used to preallocate space for a database.

ADAERR

Error file report

The ADAERR utility displays the contents of the error files generated by various utilities.

ADAFDU

File definition

The file definition utility ADAFDU defines a file in the database. It only loads the FCB and the FDT into the database and allocates the requested space for ASSO and DATA for the specified file.

ADAFIN

File information report

The ADAFIN utility displays information about one or more files, e.g. FDT, descriptor statistics and the fill percentage of blocks in the Data Storage, Normal Index and Upper/Main Index.

ADAFRM

Format and create a new database

The formatting utility ADAFRM allocates and formats the files that are used by Adabas (Associator, Data Storage, WORK, TEMP and SORT). It can also format files which have been preallocated by ADADEV.

ADAINV

Creating, removing and verifying inverted lists

The invert utility ADAINV creates, reinverts or removes inverted lists for a loaded file in a database or validates specified descriptors.

ADAMON

This utility monitors the performance of an Adabas nucleus and displays statistics on a terminal.

ADAMUP

Mass add and delete

The ADAMUP utility adds or deletes large numbers of records to/from a file in the database.

ADANUC

Starting the database, defining nucleus parameters

The ADANUC utility starts the database for online operations and defines the runtime environment.

ADAOPR

Operator utility

The operator utility is used to operate the Adabas nucleus.

ADAORD

Reorder database or files, export/import files

The reorder utility ADAORD provides functions to reorganize a database or files within a database (REORDER function) and to migrate files between databases (EXPORT and IMPORT functions).

ADAPLP

Protection log printout

This utility prints the protection log.

ADAPRI

Print Adabas blocks

The ADAPRI utility prints the contents of a block or a range of blocks in the Associator, Data Storage, WORK, TEMP or SORT for maintenance or auditing purposes.

ADAREC

Recovery of database or files

This utility reapplies updates made to the database (REGENERATE function).

ADAREP

Database report

The ADAREP utility produces the database status report. This report contains information about the current physical layout and logical contents of the database.

The information in this report includes the following: the amount and location of the space currently allocated for the Associator and Data Storage; the amount and location of unused space available for Associator and Data Storage; database file summary; checkpoint information; information about each file in the database (space allocation, space available, number of records loaded, MAXISN setting, field definitions).

ADASCR

Security functions

The security utility ADASCR creates, modifies and deletes file protection levels and user passwords, and enables the record locking capabilities of individual passwords (by using value criteria for individual database files) to be set or modified. Additionally, the utility is used to display file and password security information.

ADATST

Issuing Adabas commands

This utility issues commands to an Adabas nucleus.

ADAULD

File unloading

The unload utility ADAULD unloads a file from a database or an Adabas backup copy and produces compressed records with the same format as those produced by the compression utility ADACMP. Unloaded records may be used as input for the decompression utility ADADCU or with the mass update utility ADAMUP. Records can be unloaded from a database in the sequence in which they are currently stored in Data Storage, in the sequence of a descriptor or in ISN sequence. However, records can only be unloaded from a backup copy in the order in which they were stored by the utility.

ADAVFY

Database consistency check

This utility checks the consistency of the database. The General Control Block (GCB) is validated together with each File Control Block (FCB) and each Field Definition Table (FDT) of the loaded files. The index structure and Data Storage are validated. If specified, ADAVFY also looks for lost RABNs.

3 ADABAS (Starting The Database Nucleus)

- Functional Overview 16
- Procedure Flow 17
- Control Parameter 18

This chapter describes the utility "ADABAS".



Note: This utility is only available on Windows platforms.

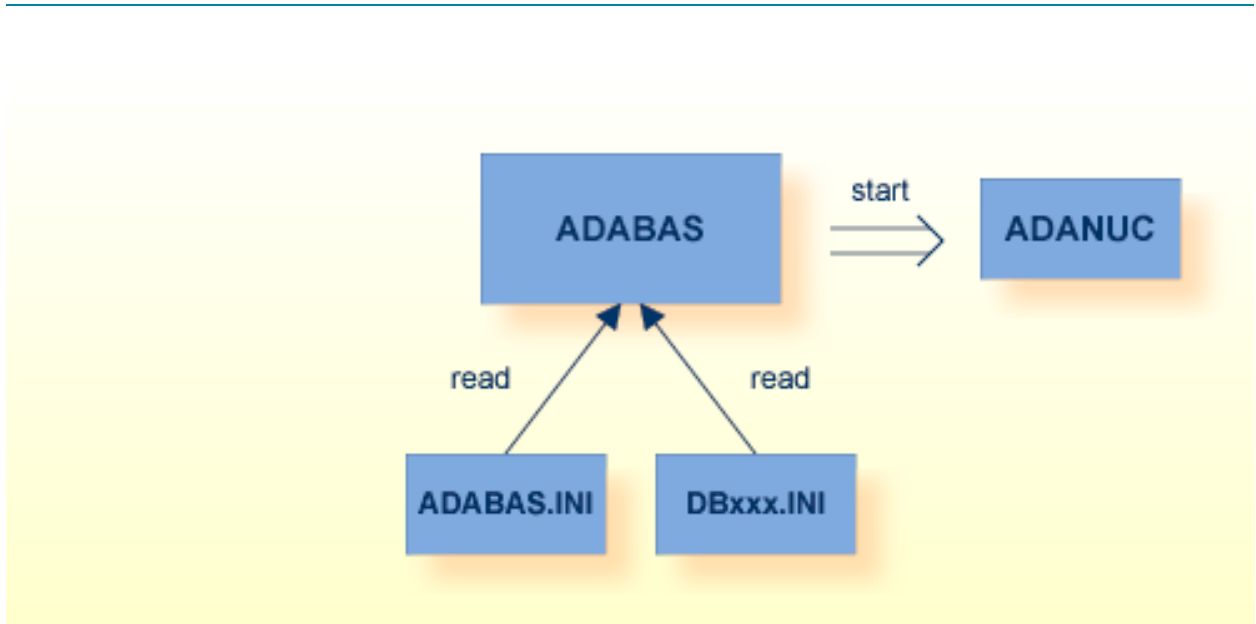
Functional Overview

The utility ADABAS is used to start the database nucleus with the nucleus parameters that are specified in the database initialization file (*DBxxx.INI*) - if you start the database nucleus ADANUC directly without parameters, the *DBxxx.INI* file is not evaluated, and the default values of the nucleus parameters are used.



Note: Control parameters and values cannot be entered interactively, and must be entered at the command prompt when the utility is started.

Procedure Flow



1. ADABAS reads the global initialization file `%ADADIR%\ETC\ADABAS.INI` to get the database initialization file of the database to be started.
2. ADABAS reads the database initialization file to get the nucleus parameters of the database to be started.
3. ADABAS starts ADANUC with the parameters read.



Note: Please refer to the *Extended Operations* section for further information about the database initialization files.

Control Parameter

The following control parameter is available:

```
M      [DBID =] number
```

DBID

```
[DBID =] number
```

This parameter selects the database to be used.

Example:

Database 20 can be started by entering either

```
adabas dbid=20
```

or

```
adabas 20
```

4 ADABCK (Dump And Restore Database Or Files)

- Functional Overview 20
- Procedure Flow 22
- Checkpoints 24
- Control Parameters 25
- Restart Considerations 37

This chapter describes the utility "ADABCK".

Functional Overview

The backup utility ADABCK provides protection against database corruption by creating Adabas backup copies. ADABCK should be used at regular intervals.

The utility dumps or restores a database or selected files from/to a database.

Making use of the internal structure of the database, this utility provides optimum performance. Unused blocks do not have to be read and can be omitted when dumping. Even though such blocks are not included in the Adabas backup copy, they can be re-created during a restore.

The backup copy can be directly assigned to tape: this option supports consecutive tapes (see *Administration, Using Utilities*).

Furthermore, a backup copy may be directed to stdout in order to support the piping of the backup data (this feature is only available on UNIX platforms). This feature is enabled by setting the environment variable (BCK001) to '-' (minus). In this case, the output messages are directed to stderr. The RESTORE and OVERLAY functions can also be used in this way, i.e. a backup copy can be read from stdin. In this case, the ADABCK control statements must be given in the command line (see *Administration, Using Utilities*).

The following functions are available:

- The COPY function copies an Adabas backup copy;
- The DUMP function dumps a database or selected files from a database to one or more sequential files, which is called an Adabas backup copy. The nucleus may be active and parallel updates are permitted on the files to be dumped while the dump is in progress;
- The EXU_DUMP function dumps a database or selected files from a database to one or more sequential files, which is called an Adabas backup copy. Only ACC users are permitted on the files to be dumped while the dump is in progress;
- The IOSTAT function prints information about the data transfer rate and the I/O waiting times.
- The OVERLAY function restores selected files or a database. The files to be restored may already be loaded in the database: ADABCK performs an implicit delete before restoring such files;
- The READ_CHECK function checks the readability (i.e. absence of parity errors) and completeness of the Adabas backup copy. These checks ensure that the dump file can be read by the RESTORE or OVERLAY function;
- The RESTORE function restores a database or selected files from an existing Adabas backup copy. If there are no security definitions for the files in the target database, the corresponding entries (as they were defined at the time the files were dumped) are set up in the security table when the file is restored;

- The list functions CONTENTS, FILES and SUMMARY display information about an Adabas backup copy. When the list functions are used, the DBID does not have to be entered first; the exception to this is when the backup file is in a raw section. In this case, the DBID is required, but the database itself does not have to be present (UNIX platforms only).

The functions DUMP, EXU_DUMP, OVERLAY and RESTORE are mutually exclusive and only one of them may be executed during a single run of this utility. The list functions can only be used together with the READ_CHECK, RESTORE or OVERLAY function.

If you perform the RESTORE or OVERLAY function and the database is too small or database containers are missing, ADABCK will automatically increase the size of the database or create the missing containers.



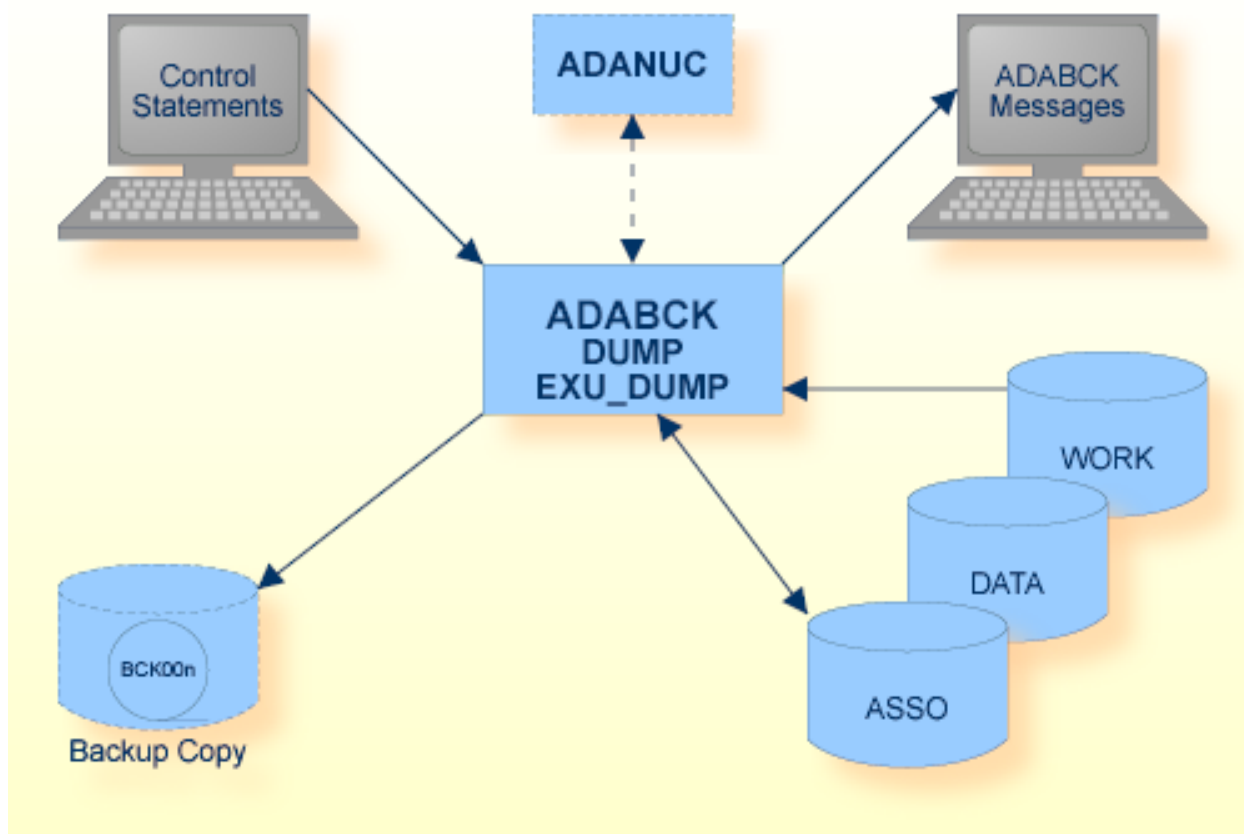
Note: The RESTORE and OVERLAY functions can process backup files created with earlier Adabas versions, but not backup files created with later Adabas versions.

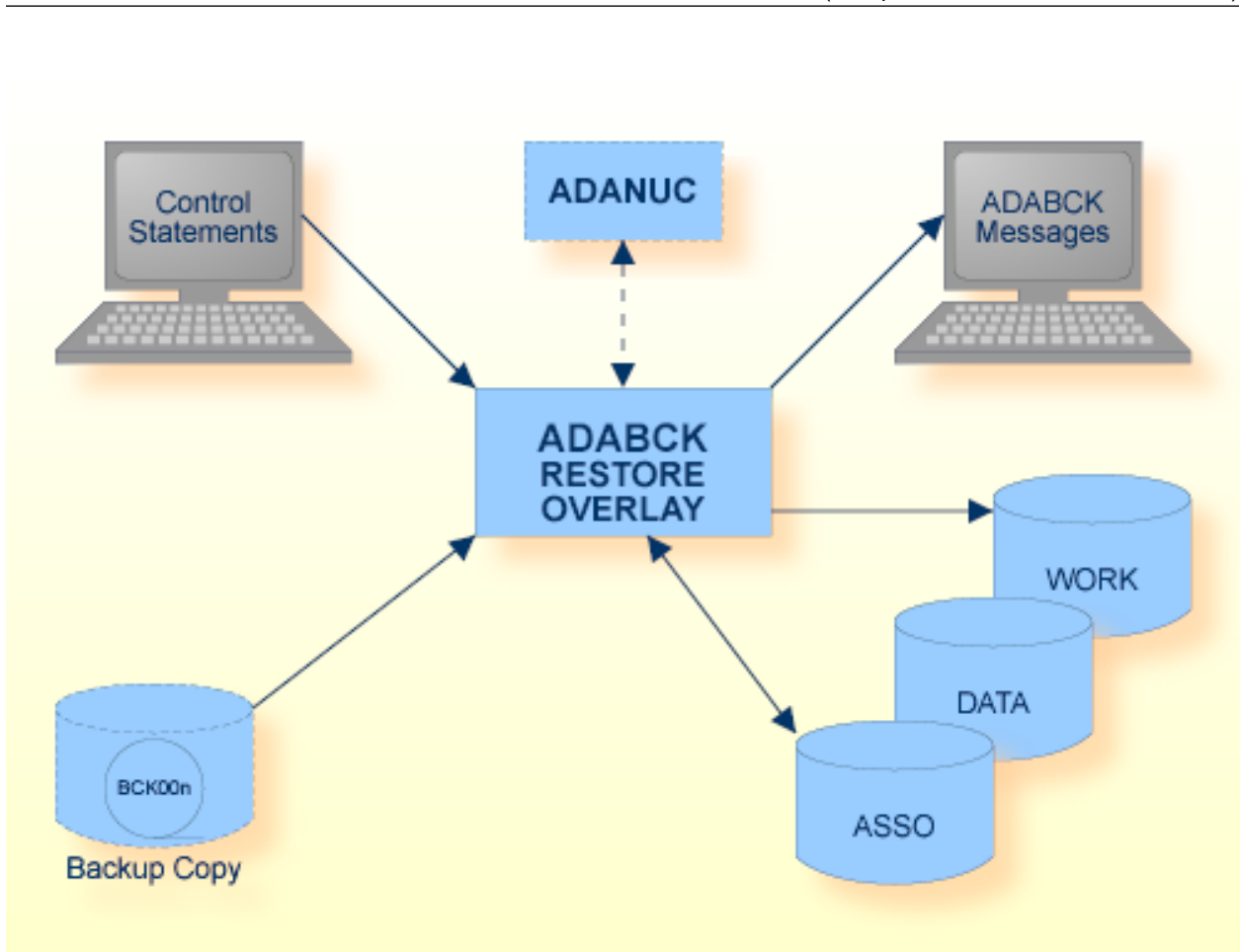


Caution: If you don't use the Adabas INI files, but instead use environment variables to specify the container file names, and if you forget to assign the environment variables/logical names before you start ADABCK, a copy of the database will be created in the database directory. If you perform a file overlay or restore when the Adabas nucleus is active, and the database has to be extended, the database is extended by the nucleus, and not by ADABCK. In this case, the nucleus extends the database even if OPTION=AUTOEXPAND was NOT specified. If you use environment variables to specify the database containers, you must consider the following when a new container has to be created for the restore/overlay: it is important that the nucleus was started with the correct environment variable settings for the new container - because the new containers are created by the nucleus, specifying the environment variable for the ADABCK process has no effect.

This utility is a single-function utility.

Procedure Flow





Data Set	Environment Variable/ Logical Name	Storage Medium	Additional Information
Associator	ASSOx	Disk	
Backup copy	BCK00n	Disk, Tape (see note 1) stdin/ SYS\$INPUT (see note 2), stdout/ SYS\$OUTPUT (see note 3)	Output of DUMP/EXU_DUMP function, input for other functions
	BCKOUT	Disk, Tape (see note 1)	Output of COPY function
Data storage	DATAx	Disk	
Control statements	stdin SYS\$INPUT		Utilities Manual
ADABCK messages	stdout/ SYS\$OUTPUT (see note 4), stderr/ SYS\$ERR (see note 5)		Messages and Codes
Work	WORK1	Disk	

**Notes:**

1. A named pipe can be used for this sequential file (UNIX platforms only, see *Administration, Using Utilities* for details).
2. For functions other than DUMP or EXU_DUMP (BCK001 only).
3. For DUMP or EXU_DUMP (BCK001 only).
4. If BCK001 is not stdout/SYS\$OUTPUT.
5. If BCK001 is stdout/SYS\$OUTPUT.

The sequential files BCK00n can have multiple extents. For detailed information about sequential files with multiple extents, see *Administration, Using Utilities*.

Checkpoints

The following table shows the nucleus requirements for each function and the checkpoint written:

Function	Nucleus must be active	Nucleus must NOT be active	Nucleus is NOT required	Checkpoint written
CONTENTS			X	-
COPY			X	-
DUMP	X(see note 1)		X	SYNX
EXU_DUMP	X(see note 1)		X	SYNX
FILES			X	-
NEW_PLOG				SYNC
OVERLAY		X(see note 2)	X(see note 3)	SYNP
READ_CHECK			X	-
RESTORE		X(see note 2)	X(see note 3)	SYNP
SUMMARY			X	-

**Notes:**

1. Nucleus only required when AUTORESTART is pending at the end of this function.
2. For restore of database or system files.
3. For restore of files.

Control Parameters

The following control parameters are available:

```

CONTENTS

COPY [= number]

M   DBID = number

    DUMP = { * | (number[-number][,number[-number]]...) }
          [,BLOCKSIZE = number [K|M]]
D     [{,DRIVES = number} |
D     {,[NO]DUAL} ]
D     [,ET_SYNC_WAIT = number]
D     [, [NO]NEW_PLOG]

    EXU_DUMP = { * | (number[-number][,number[-number]]...) }
              [,BLOCKSIZE = number [K|M]]
D       [{,DRIVES = number} |
D       {,[NO]DUAL} ]
D       [, [NO]NEW_PLOG]

    FILES = { * | (number[-number][,number[-number]]...) }

IOSTAT

    OVERLAY = { * | (number[-number][,number[-number]]...) }
              [,FMOVE [(number [,number [-number]]...)]]
              [,FORMAT = (keyword [,keyword])]
              [,KEEP_FILE_ALLOC]
              [,NEW_DBID = number]
              [,RENUMBER = (number[-number] [,number [-number]]...)]]

PARALLEL = keyword

READ_CHECK

    RESTORE = { * | (number[-number][,number[-number]]...) }
              [,FMOVE [(number [,number [-number]]...)]]
              [,FORMAT = (keyword [,keyword])]
              [,NEW_DBID = number]
              [,RENUMBER = (number[-number] [,number [-number]]...)]]

SUMMARY

```

CONTENTS

CONTENTS

This parameter displays a list of files in an Adabas backup copy created with the DUMP or EXU_DUMP function.

Example

```
adabck: contents

Database dumped on 14-JUL-2005 14:49:17

Database 76, DOKU-DATABASE

File      1, CHECKPOINT-FILE , loaded on 12-JUL-2005 14:57:05
File      2, SECURITY-FILE   , loaded on 12-JUL-2005 14:57:05
File      3, USER-DATA-FILE , loaded on 12-JUL-2005 14:57:05
File      9, EMPLOYEES-FILE , loaded on 12-JUL-2005 14:57:08
File     11, EMPLOYEES-NAT  , loaded on 12-JUL-2005 14:57:09
File     12, VEHICLES       , loaded on 12-JUL-2005 14:57:11
File     13, MISCELLANEOUS , loaded on 12-JUL-2005 14:57:10
```

COPY

COPY [= number]

This function creates a new file from an existing Adabas backup copy. The input file (BCK0xx) and the output file (BCKOUT) may be on either disk or tape, where xx is either the specified number, or 01 if no number is explicitly specified.

DBID

DBID = number

This parameter selects the database to be used.

DUMP

```
DUMP = { * | (number[-number][,number[-number]]...) }
      [,BLOCKSIZE = number [K|M]]
      [ {,DRIVES = number} |
        {, [NO]DUAL } ]
      [,ET_SYNC_WAIT = number ]
      [, [NO]NEW_PLOG ]
```

At the file level, this function dumps the files specified by the numbers in the list. LOB files specified are ignored, but the LOB files assigned to all base files are dumped too. An asterisk '*' specifies

that the complete database is to be dumped. Parallel updates are permitted on the files to be dumped while the dump is in progress.

If the nucleus is running in parallel (online backup), ADABCK must ensure that all transactions affecting the dumped files are completed by all users before ADABCK terminates. This is called ET synchronization - please refer to the section *ET Synchronization* in *Administration* for further information. If you perform a dump at the file level with the option `NONNEW_PLOG`, the ET synchronization is performed at the file level; otherwise the ET synchronization is performed for the complete database.

If you specify files with referential constraints, all files connected to these files via referential constraints must also be specified in order to maintain referential integrity.

BLOCKSIZE = number[K|M]

This parameter can be specified to change the I/O transfer blocksize. If `PARALLEL` is specified, the default blocksize is 512 KB. The following values can be specified: 64KB, 128KB, 256KB, 512KB, 1MB, 2MB, ... 12MB. The blocksize specified will be used in a subsequent `RESTORE` function.

DRIVES = number

This parameter limits the maximum number of output devices to be operated in parallel. It can be used to split a backup file into several extents. The output is sent to `BCK0xx`.

The default value is 1 and the maximum value is 10.

The parameters `DRIVES` and `DUAL` are mutually exclusive, and only one of them may be specified in a given call of the `DUMP` function.

[NO]DUAL

`DUAL` specifies that two physical copies of the dumped information are to be created. The output is sent to `BCK001` and `BCK002`.

The default is `NODUAL`.

The parameters `DUAL` and `DRIVES` are mutually exclusive, and only one of them may be specified in a given call of the `DUMP` function.

ET_SYNC_WAIT = number

This parameter defines the time (in seconds) that ADABCK waits for ET-logic users to come to ET status at the end of the DUMP function.

If this parameter is omitted, the value currently in effect for the database nucleus (ADANUC parameter TT) is taken.

The minimum value is 1 and the maximum value is 32767.

[NO]NEW_PLOG

This option specifies whether or not to close the protection log file and create a new log file at the end of the DUMP function.

The default for a database dump is NEW_PLOG, and for a file dump it is NONEW_PLOG.



Caution: Before V6.3 SP1 Fix 13, the default for a file dump was NEW_PLOG. In most cases, this change is of no consequence, but if you really need the PLOG switch, you must specify NEW_PLOG explicitly.

EXU_DUMP

```
EXU_DUMP = {*(number[-number][,number[-number]]...)}  
           [,BLOCKSIZE = number [K|M]]  
           [ {,DRIVES = number} |  
           {,[NO]DUAL} ]  
           [,[NO]NEW_PLOG]
```

At the file level, this function dumps the files specified by the numbers in the list. LOB files specified are ignored, but the LOB files assigned to all base files are dumped too. An asterisk '*' specifies that the complete database is to be dumped. Only ACC users are permitted on the files to be dumped while the dump is in progress. ET-synchronization is not required.

If you specify files with referential constraints, all files connected to these files via referential constraints must also be specified in order to maintain referential integrity.

BLOCKSIZE = number[K|M]

This parameter can be specified to change the I/O transfer blocksize. If PARALLEL is specified, the default blocksize is 512 KB. The following values can be specified: 64KB, 128KB, 256KB, 512KB, 1MB, 2MB, ... 12MB. The blocksize specified will be used in a subsequent RESTORE function.

DRIVES = number

This parameter limits the maximum number of output devices to be operated in parallel. It can be used to split a backup file into several extents. The output is sent to BCK0xx.

The default value is 1 and the maximum value is 10.

The parameters DRIVES and DUAL are mutually exclusive, and only one of them may be specified in a given call of the DUMP function.

[NO]DUAL

DUAL specifies that two physical copies of the dumped information are to be created. The output is sent to BCK001 and BCK002.

The default is NODUAL.

The parameters DUAL and DRIVES are mutually exclusive, and only one of them may be specified in a given call of the DUMP function.

[NO]NEW_PLOG

This option specifies whether or not to close the protection log file and create a new log file at the end of the EXU_DUMP function.

This option must not be used if dumping single files.

The default is NEW_PLOG for EXU_DUMP=*.

Examples 1-3

In the examples below, the files 1, 2, 4, 6, 7, 8, 10, 11, 12, and 13 are loaded in the selected database.

Example 1

```
adabck: dump = * , drives = 2
```

The database is dumped with two output devices operating in parallel.

Example 2

```
adabck: exu_dump = 8 , dual
```

File 8 is dumped and two physical copies are created. Only ACC users are allowed on file 8 while the dump is in progress.

Example 3

```
adabck: dump = (8-11,13,6,1-4), et_sync_wait = 5
```

Files 1, 2, 4, 6, 8, 10, 11 and 13 are dumped. ADABCK allows a maximum of 5 seconds for ET-logic users to come to ET status.

FILES

```
FILES = { * | (number[-number][,number[-number]]...)
```

This parameter displays status information of the specified files in a dump file.

IOSTAT

```
IOSTAT
```

If this parameter is specified, the data transfer rate and the I/O (waiting) times on the various devices are printed at the end of ADABCK processing.

Example:

```
adabck db=36 parallel=multi_process dump=\* drives=3 iostat
...
-----
Dump Method      : parallel
Blocksizes      : DB:          512 KB      BCK:          512 KB
DB I/O time     : total:        27.09 sec   average:      8084 us
BCK 1 I/O time  : total:         1.16 sec   average:      7606 us
BCK 2 I/O time  : total:         0.00 sec   average:       944 us
BCK 3 I/O time  : total:         1.24 sec   average:     1375 us
Wait rates      :      waits   nowaits    rate   mreq
DB              :      1439     1898      43%    8
```

```
Transfer rate   : 15215 KB/sec
-----
```

```
%ADABCK-I-IOCNT, 2 IOs on dataset WORK
%ADABCK-I-IOCNT, 3147 IOs on dataset DATA
%ADABCK-I-IOCNT, 229 IOs on dataset ASSO
%ADABCK-I-IOCNT, 153 IOs on dataset BCK001
%ADABCK-I-IOCNT, 2 IOs on dataset BCK002
%ADABCK-I-IOCNT, 906 IOs on dataset BCK003
```

The IOSTAT statistics display the following information:

Dump Method

Either parallel or non-parallel, depending on the setting of the PARALLEL parameter.

DB I/O time

The total I/O time in seconds and the average time per I/O operation in microseconds for the access to the ASSO and DATA containers.

BCK n I/O time

The total I/O time in seconds and the average time per I/O operation in microseconds for the access to the backup files.



Note: The I/O time measured is the time required for the I/O system functions. This may be different from the physical I/O times actually required to accessing the disks because of caches in the operating system or in the storage system and because of usage of asynchronous I/O.

Wait rates (only for dump method parallel)

For a parallel backup/restore, the I/Os for the database containers are performed asynchronously. The wait rate shows for how many ASSO or DATA I/Os a wait operation is required. mreq is the maximum number of parallel I/O requests for database containers.



Note: Only the I/Os for the real backup or restore are counted. During the startup phase of ADABCK, some additional I/Os are required; therefore the sum of wait and nowait I/Os is less than the sum of ASSO and DATA I/Os.

BF sync count (only for a backup in online mode)

In the case of a backup in online mode during a buffer flush, synchronization with the nucleus is required in order to guarantee that the modified database blocks written to disk by the buffer flush are also written to the backup file(s). The BF sync count is the number of these buffer flush synchronizations.

ET sync time (only for a backup in online mode)

At the end of a backup in online mode, an ET synchronization is required, i.e. ADABCK must wait until all ET logic users come to ET status. The ET sync time is the time required for this ET synchronization.

Transfer rate

This is the number of kilobytes read from or written to the backup file(s) per second.

**Notes:**

1. For the transfer rate, only the pure backup/restore time is taken into consideration, but not the time required for the preparation of the backup/restore. Therefore, the transfer rate may be higher than the transfer rate you would get if you compute the transfer rate based on the total elapsed time of ADABCK.
2. In the case of small backups, rounding errors may occur in the computation. Therefore, for very small backups the transfer rate is not displayed, because the value would be too inaccurate.
3. Because usually many database blocks are not filled completely, and because only the net data are copied to the backup file(s), the transfer rate is less than the rate you would get if you consider the processed database space.

OVERLAY

```
OVERLAY = {*(number[-number][,number[-number]]...)}  
          [,FMOVE [(number [,number [-number]]...)]]  
          [,FORMAT = (keyword [,keyword] ) ]  
          [,KEEP_FILE_ALLOC]  
          [,NEW_DBID = number]  
          [,RENUMBER = (number[-number] [,number [-number]]...)]]
```

This function restores the files specified by the numbers in the list at file level. LOB files specified are ignored, but the LOB files assigned to all base files are restored too. The files to be restored may already be loaded in the database. ADABCK performs an implicit delete before restoring such files. If only one file of a LOB group is overlaid, the other file of the LOB group is also deleted. An asterisk (*) specifies that a restore is to be made at the database level. Exclusive control over the database container files is required.

Only the specified files are overlaid, even if there are referential integrity constraints to other files; these referential integrity constraints are removed.

FMOVE [(number [,number [-number]]...)]

If this keyword is specified, ADABCK reallocates all files to be overlaid or the specified subset rather than attempting to restore them in the same block ranges as in the backup. Using this keyword reduces the number of file extents as much as possible.

FORMAT = (keyword [,keyword])

The keywords ASSO and/or DATA may be specified. This parameter is used to format Associator and/or Data Storage blocks. When restoring at the file level, only blocks contained in the unused areas of the files' extents are formatted.

KEEP_FILE_ALLOC

If this parameter is specified, ADABCK tries to keep the allocation of the file as it currently is in the database, as opposed to restoring it with the same block ranges as on the backup. This keyword can, for example, be used when a file has been reorganized since the backup was made or also if more space has since been preallocated to the file. If the file on the backup has more blocks allocated than are currently available in the database, the remaining blocks will be allocated in an arbitrary location. This keyword can only be used in conjunction with a file list.

NEW_DBID = number

This parameter can be used to change the identifier of the database to be restored. This parameter can only be specified when restoring a complete database.

A new identifier can be used to restore a backup copy of an active database into a different set of container files. The new identifier may not be identical to that of another active database.

If this parameter is omitted, the database identifier remains unchanged.

RENUMBER = (number[-number] [,number [-number]]...)

RENUMBER is used to renumber the files to be overlaid in the target database. The following restrictions and requirements apply:

- There must be a 1:1 relationship between the files specified in the OVERLAY file list and the RENUMBER file list.
- If you specify a range in the OVERLAY file list, the corresponding range in the RENUMBER file list must be the same size.
- Normally it is not necessary to specify LOB files in the OVERLAY file list. However, if the LOB file is also to be renumbered, the LOB file must also be specified.
- Files may occur more than once in the OVERLAY file list, for example: (11-55),(44-99). In this case, you are not allowed to specify different target file numbers for the same source file numbers. For the example file list, it is correct to specify RENUMBER=(1011-1055,1044-1099), whereas RENUMBER=(1011-1055,2044-2099) is incorrect.
- It is not allowed to renumber more than one file to the same target file number.

PARALLEL

```
PARALLEL = keyword
```

This parameter can be specified to increase processing speed when creating/restoring from backups on slow devices (e.g. tape drives) by using parallel devices. The keyword MULTI_PROCESS can be used. If PARALLEL=MULTI_PROCESS is specified, the default value of the BLOCKSIZE parameter changes to 512 KB.

The ADABCK operation is only performed in parallel if the number of backup files (ADABCK subparameter DRIVES for DUMP or EXU_DUMP) is greater than 1.



Notes:

1. If it is to be used, PARALLEL must be specified after the DBID parameter and before the DUMP, EXU_DUMP, OVERLAY or RESTORE parameter.
2. The PARALLEL parameter is not supported on Windows platforms.
3. It is possible to pass the output of ADABCK DUMP or ADABCK EXU_DUMP to named pipes, which can be directly used as input for an ADABCK RESTORE or ADABCK OVERLAY in order to copy a database or some files from one database to another database.
4. The PARALLEL parameter does not improve the performance of the READ_CHECK function.

READ_CHECK

```
READ_CHECK
```

This function checks the readability (i.e. absence of parity errors) and completeness of the Adabas backup copy. These checks are made to ensure that the dump file can be used to restore the database or files with the RESTORE or OVERLAY function of this utility.

RESTORE

```
RESTORE = {*(number[-number][,number[-number]]...)}  
          [,FMOVE [(number [,number [-number]]...)]]  
          [,FORMAT = (keyword [,keyword] ) ]  
          [,NEW_DBID = number]  
          [,RENUMBER = (number[-number] [,number [-number]]...)]]
```

This function restores the files specified by the numbers in the list at the file level. LOB files specified are ignored, but the LOB files assigned to all base files are restored too. If a file list is given, the files to be restored must not be loaded in the database. An '*' specifies that a restore is to be made at the database level. In this case, the files may already be loaded in the database and will implicitly be deleted or substituted by files in the dump with identical file numbers. Exclusive control over the database container files is required.

Only the specified files are restored, even if there are referential integrity constraints to other files; these referential integrity constraints are removed.



Note: You can only use RESTORE=* if the dump file was created with DUMP=* or EXU_DUMP=*

FMOVE [(number [,number [-number]]...)]

If this keyword is specified, ADABCK reallocates all files to be restored or the specified subset rather than attempting to restore them in the same block ranges as in the backup. Using this keyword reduces the number of file extents as much as possible.

FORMAT = (keyword [,keyword])

The keywords ASSO and/or DATA may be specified. This parameter is used to format Associator and/or Data Storage blocks. When restoring at the file level, only blocks contained in the unused areas of the files' extents are formatted.

NEW_DBID = number

This parameter can be used to change the identifier of the database to be restored. This parameter can only be specified when restoring a complete database.

A new identifier can be used to restore a backup copy of an active database into a different set of container files. The new identifier may not be identical to that of another active database.

If this parameter is omitted, the database identifier remains unchanged.

RENUMBER = (number[-number] [,number [-number]]...)

RENUMBER is used to renumber the files to be restored in the target database. The following restrictions and requirements apply:

- There must be a 1:1 relationship between the files specified in the RESTORE file list and the RENUMBER file list.
- If you specify a range in the RESTORE file list, the corresponding range in the RENUMBER file list must be the same size.
- Normally it is not necessary to specify LOB files in the RESTORE file list. However, if the LOB file is also to be renumbered, the LOB file must also be specified.
- Files may occur more than once in the RESTORE file list, for example: (11-55),(44-99). In this case, you are not allowed to specify different target file numbers for the same source file numbers. For the example file list, it is correct to specify RENUMBER=(1011-1055,1044-1099), whereas RENUMBER=(1011-1055,2044-2099) is incorrect.
- It is not allowed to renumber more than one file to the same target file number.

Examples:

It is assumed that none of the files specified in the DUMP examples above are loaded in the selected database.

```
adabck: restore = *
```

The complete database is restored. The output of DUMP=* or EXU_DUMP=* may be used as input for this example. The nucleus must be inactive.

```
adabck: restore = 8
```

File 8 is restored. The output of any of the DUMP/EXU_DUMP examples above can be used as input for this example. The nucleus may be active (assuming that file 8 is neither the checkpoint file nor the security file).

```
adabck: restore = (11-13, 1, 4-8)
```

Only the outputs of the first and last DUMP/EXU_DUMP example could be used. The output of the last example would restore files 1, 4, 6, 8, 11 and 13, whereas that of the first example would restore files 7 and 12 as well.

SUMMARY

SUMMARY

This parameter displays general information and physical layout of the database in the Adabas backup copy created by a previous run of the DUMP/EXU_DUMP function.

Example

```
adabck: summary

Database dumped on 14-JUL-2005 14:49:17

Database 76, DOKU-DATABASE

Summary of Database 76          14-JUL-2005 14:49:18

DATABASE NAME          DOKU-DATABASE
DATABASE ID            76
MAXIMUM FILE NUMBER LOADED 30
SYSTEM FILES          1 (CHK),    2 (SEC),    3 (USR)
```

```

ACTUAL FILES LOADED          6
CURRENT PLOG NUMBER         13
CURRENT CLOG NUMBER         1

```

Container File	Device Type	Extents from	in Blocks to	Number of Blocks	Block Size	Total Size (Megabytes)
ASS01	file	1	1,536	1,536	2KB	3.00 MB
DATA1	file	1	768	768	4KB	3.00 MB
WORK1	file	1	1,365	1,365	3KB	4.00 MB
						10.00 MB =====

Restart Considerations

ADABCK has no restart capability. An abnormally-terminated ADABCK execution must be rerun from the beginning.

An interrupted RESTORE/OVERLAY of one or more files will result in lost RABNs which can be recovered by executing the RECOVER function of the utility ADADBM. An interrupted RESTORE/OVERLAY of a database results in a database that cannot be accessed.

5 ADACLP (Command Log Report)

- Functional Overview 40
- Procedure Flow 41
- Checkpoints 42
- Control Parameters 42
- Specifying Multiple Selection Criteria 48

This chapter describes the utility "ADACLCP".

Functional Overview

The ADACLCP utility prints the command log with a line width of 132 characters.



Note: ADACLCP can only process command logs of nucleus sessions that were started with the ADANUC parameter CLOGLAYOUT=5 (5 is the default value). Please refer to *ADANUC*, **CLOGLAYOUT** for further information.

A record is written in the command log for each Adabas command issued. Command logging must be enabled during Adabas startup with the nucleus parameter LOGGING, or when the nucleus is already active with the ADAOPR parameter LOGGING.

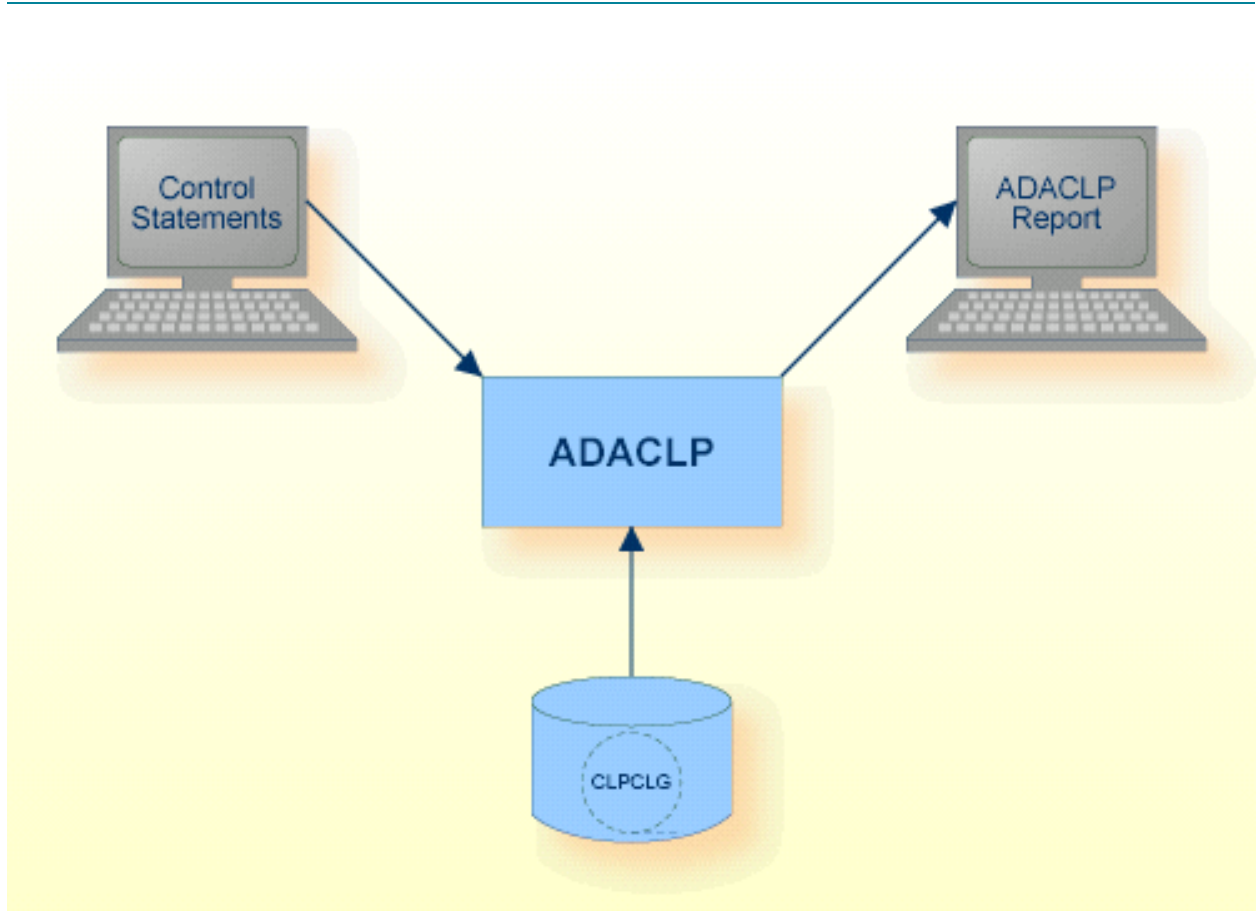


Note: For performance reasons, the Adabas nucleus determines the command start timestamp only if command logging has been enabled. For this reason, the command start date and the command duration are not displayed for Adabas commands that are already active but not yet finished when command logging is switched on.

Any of the ADACLCP parameters selects a subset of the command log information.

This utility is a single-function utility.

Procedure Flow



Data Set	Environment Variable/ Logical Name	Storage Medium	Additional Information
Command log	CLPCLG	Disk, Tape (* see note)	Utilities Manual, ADACLP
Control statements	stdin/ SYS\$INPUT		Utilities Manual
ADACLP report	stdout/ SYS\$OUTPUT		Messages and Codes



Note: (*) A named pipe can be used for this sequential file (not on OpenVMS, see *Administration, Using Utilities* for details).

The sequential files CLPCLG can have multiple extents. For detailed information about sequential files with multiple extents, see *Administration, Using Utilities*.

Checkpoints

The utility writes no checkpoints.

Control Parameters

The following control parameters are available:

```
D  [NO]ADDITIONS_2
    CLASS = (keyword[,keyword]...)
    CLOG = (number[,number])
    COMMAND = (keyword[,keyword]...)
    DATE = ([absolute-date][,[absolute-date]])
    DBID = number
D  DISPLAY = (keyword[,keyword]...)
D  ES_ID [=number]
D  FILE = (number[-number][,number[-number]]...)
D  [NO]HEXADECIMAL
D  LOGIN_ID = string
    NODE_ID = string
D  PAGE = number
D  RECORDS = number[-number]
    RESPONSE = (number[-number][,number[-number]]...)
D  USER_ID = string
D  WIDTH = number
```

[NO]ADDITIONS_2

```
[NO]ADDITIONS_2
```

This option can be used to display the Additions 2 field instead of the command ID.

The default is NOADDITIONS_2.

CLASS

```
CLASS = (keyword [,keyword]...)
```

This parameter selects the log records whose command codes belong to the specified command class. All records are selected if neither the CLASS parameter nor the COMMAND parameter is specified.

CLASS and COMMAND are mutually exclusive.

The following keywords can be used:

KEYWORD	USE
CONTROL	Selects control commands such as `open' and `close';
FIND	Selects find commands;
READ	Selects read commands;
UPDATE	Selects update commands.

Example:

```
adaclp: class = find
```

The log records of the commands S1, S2, S4, S8 and S9 are selected.

CLOG

```
CLOG = (number [,number])
```

This parameter is required if the command log is within a raw section. It is optional if the command log is within a file system. The CLOG number and the extension count can be specified. If no extension count is specified, Adabas will open subsequent extents as necessary. If an extent count is specified, then only the specified extent will be processed.



Note: This parameter applies to UNIX platforms only.

COMMAND

```
COMMAND = (keyword [,keyword]...)
```

This parameter selects the log records with an Adabas command code specified by the keywords. Up to ten keywords can be defined. If neither the COMMAND parameter nor the CLASS parameter is specified, all records are selected.

COMMAND and CLASS are mutually exclusive.

All valid Adabas commands (A1...S9) can be used as keywords (see *Command Reference* for further information).

DATE

```
DATE = ([absolute-date] [, [absolute-date]])
```

This parameter selects the log records in the range specified by the optional date strings. The date strings must correspond to the following absolute date and time format:

```
dd-mmm-yyyy[:hh:mm:ss]
```

Leading zeroes in the date and time specification may be omitted. Any numbers not specified are set to 0, for example 28-jul-2012 is equivalent to 28-jul-2012:00:00:00.

By default, all log records are selected.

Examples:

```
adaclp: date = 8-aug-2012
```

The log record written on 8-AUG-2012 00:00:00 is selected

```
adaclp: date = (8-aug-2012:12,)
```

All log records written from 8-AUG-2012 12:00:00 onwards are selected.

```
adaclp: date = (,8-aug-2012:12:34)
```

All log records written up to 8-AUG-2012 12:34:00 are selected.

DBID

DBID = number

This parameter selects the database to be used. This parameter must be specified when the CLOG to be used is on a raw device.



Note: This parameter applies to UNIX platforms only.

DISPLAY

DISPLAY = (keyword [,keyword]...)

This parameter is used to display various kinds of information from the command log. The keywords shown in the following table are available. Information for these keywords can only be displayed if corresponding data was logged during the nucleus session.

KEYWORD	MEANING
CB	<ul style="list-style-type: none"> ■ Command log record number ■ Starting date and time of the command ■ Duration of the command in microseconds ■ User ID specified in the corresponding OP command ■ Node ID ■ Login ID or, if ES_ID was specified, environment-specific ID ■ Selected fields of the control block ■ '*' in column 'X' indicates utility or exclusive file usage ■ The thread which processed the command ■ I/O statistics <p>Note: For command logs created with versions lower than Version 6.3 SP1, the duration of the command is displayed in milliseconds.</p>
FB	Format buffer.
FULL_CB	All fields of the control block. Other information shown for DISPLAY=CB is not shown here.
IB	ISN buffer.
IO	IO list.
RB	Record buffer.
SB	Search buffer.
STATISTICS	Command statistics of the selected records.
VB	Value buffer.

The default is DISPLAY = CB.

ES_ID

```
ES_ID [ = number]
```

This parameter causes the environment-specific ID to be displayed instead of the login ID.

If a number is specified, only records with information for the specified environment-specific ID (process ID) will be selected.

By default, all records are selected.

FILE

```
FILE = (number [- number] [,number [- number]]...)
```

This parameter selects the log records with commands that reference the file(s) specified by number or range of numbers. A maximum of 20 files may be specified.

By default, all records are selected.

[NO]HEXADECIMAL

```
[NO]HEXADECIMAL
```

If this parameter is set to HEXADECIMAL, the record buffer and value buffer are displayed in hexadecimal format (when DISPLAY=RB or DISPLAY=VB is specified).

The default is NOHEXADECIMAL.

LOGIN_ID

```
LOGIN_ID = string
```

This parameter selects all records with the specified login ID.

By default, all records are selected.

NODE_ID

```
NODE_ID = string
```

This parameter selects the log records from the specified node.

The node identification shown while processing ADAOPR with the parameter DISPLAY = UQ must be used.

This parameter is valid only if ENTIRE NET-WORK is installed.

PAGE

```
PAGE = number
```

This parameter defines the page size, in lines, used for the printout.

The default is 59 lines.

RECORDS

```
RECORDS = number [-number]
```

This parameter selects the log records in the specified range of log record numbers. Log record numbers start with 1 after the log is switched on.

By default, all records are selected.

RESPONSE

```
RESPONSE = (number [- number] [,number [- number]] ... )
```

This parameter selects the records with the specified response code or range of response codes.

USER_ID

```
USER_ID = string
```

This parameter selects the records with the user ID specified in 'string'.

By default, all users are selected.

Example

```
user_id = *adarep
```

All records that represent commands issued from the utility ADAREP are selected.

WIDTH

```
WIDTH = number
```

This parameter selects the output line width. Valid values are 80 and 132.

The default is 132.

Specifying Multiple Selection Criteria

If multiple selection criteria are specified, they are combined by a logical AND, e.g.

```
command = 13, file = 5
```

This selects all L3 commands on file 5.

6 ADACMP (Compression Of Data)

▪ Functional Overview	50
▪ Procedure Flow	51
▪ Checkpoints	52
▪ Control Parameters	52
▪ Output	64
▪ Report	65
▪ Restart Considerations	65

This chapter describes the utility "ADACMP".

Functional Overview

The compression utility ADACMP compresses user raw data into a form which can be used by the mass update utility ADAMUP.

The input data for this utility must be contained in a sequential file. LOB field values can also be provided in separate files.

The logical structure and characteristics of the input data are described by a field definition table (FDT). These statements specify the level number, field name, standard length and format together with any definition options that are to be assigned to the field (descriptor, unique descriptor, multiple-value field, null value suppression, fixed storage, periodic group). See *Administration, FDT Record Structure* for more detailed information about the layout of the file in the database and characteristics of the input data.

Each field in the input record without the option SY (system generated) is compressed. Compression consists of removing trailing blanks from alphanumeric fields and leading zeros from numeric fields. Unpacked and packed fields are checked for correct data. Fields defined with the fixed storage option are not compressed. A user exit is provided to allow additional editing of each input record with a user-written routine.

System generated fields are either regenerated or decompressed, depending on the keyword specified for the ADACMP parameter SYFINPUT.

This utility creates three types of output files:

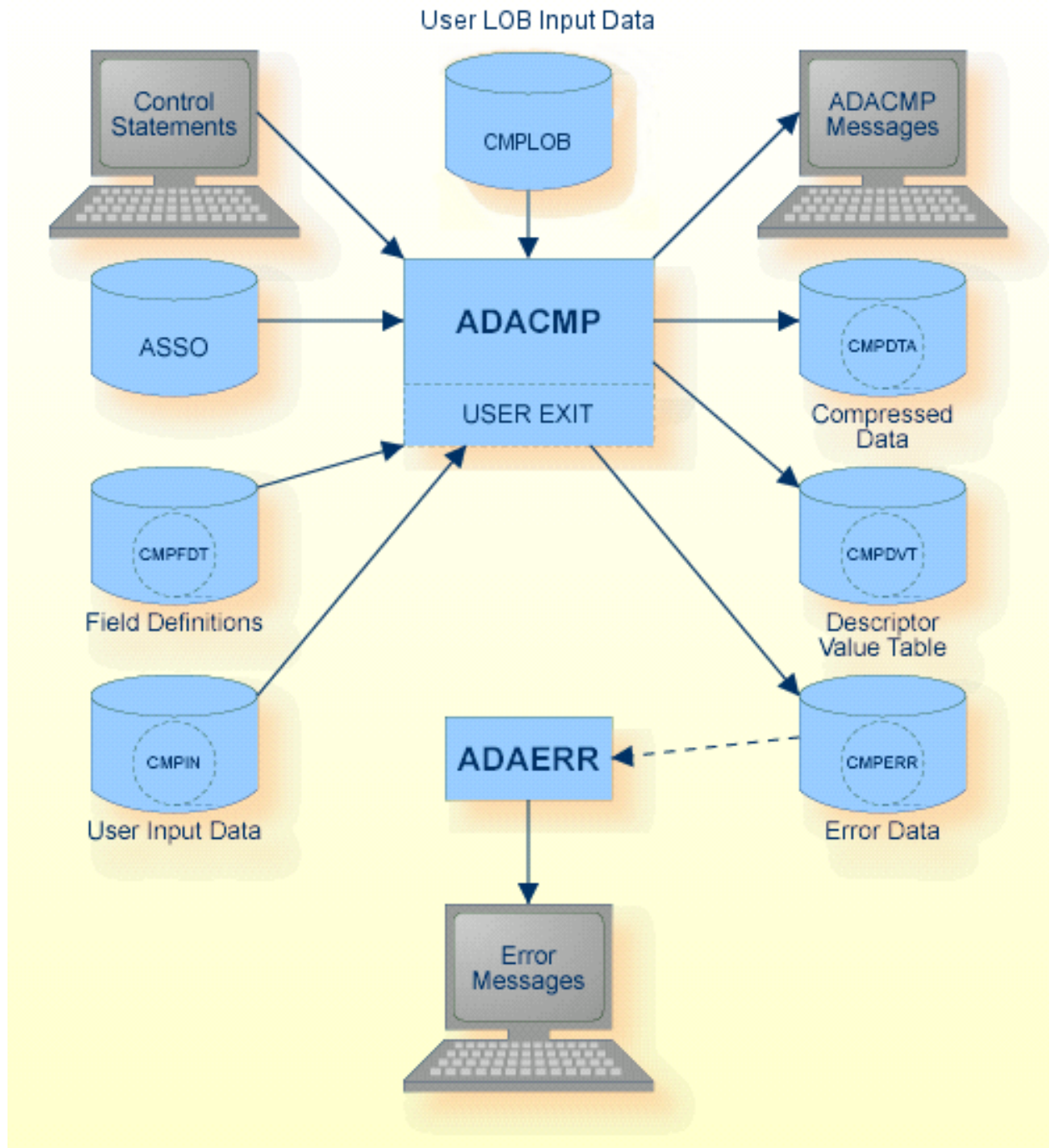
- Compressed data.
- Descriptor values.
- Records with errors.

The sizes of the descriptor values of all descriptors are listed at the end of execution.

If the utility writes records to the error file, it will exit with a non-zero status.

This utility is a single-function utility.

Procedure Flow



The sequential files CMPDTA, CMPDVT and CMPERR can have multiple extents. For detailed information about sequential files with multiple extents, see *Administration, Using Utilities, Adabas*

Sequential Files, Multiple Extents . CMPLOB is a directory that contains files which may be stored as LOB values in the database.

Data Set	Environment Variable/ Logical Name	Storage Medium	Additional Information
Associator	ASSOx	Disk	
Compressed data	CMPDTA	Disk, Tape (* see note)	output by ADACMP
Descriptor Value Table	CMPDVT	Disk, Tape (* see note)	output by ADACMP
Rejected data	CMPERR	Disk, Tape (* see note)	output by ADACMP
Input data FDT	CMPFDT	Disk, Tape (* see note)	Utilities Manual
User input data	CMPIN	Disk (* see note)	Utilities Manual
User LOB input data	CMPLOB	Disk	Utilities Manual
ADACMP control statements	stdin/ SYS\$INPUT		Utilities Manual
ADACMP messages	stdout/ SYS\$OUTPUT		Messages and Codes



Note: (*) A named pipe can be used for this sequential file (see *Administration, Using Utilities, Adabas Sequential Files, Using Named Pipes* for details).

If the SINGLE_FILE option is set, the Descriptor Value Table (DVT) and the compressed user data are written together to the logical name CMPDTA.

Checkpoints

The utility writes no checkpoints.

Control Parameters

The following control parameters are available:

```

DBID = number
D [NO]DST
FDT
FIELDS {uncompressed_field_definition | FDT}...{END_OF_FIELDS | . }

```

```
FILE = number
D [NO]LOBS
D [NO]LOWER_CASE_FIELD_NAME
D MAX_DECOMPRESSED_SIZE = number [K|M]
D MUPE_C_L = {1|2|4}
D [NO]NULL_VALUE
D NUMREC = number
D RECORD_STRUCTURE = keyword
SEPARATOR = character | \character
D [NO]SHORT_RECORDS
D [NO]SINGLE_FILE
SKIPREC = number
D SOURCE_ARCHITECTURE = (keyword[,keyword][,keyword])
D SYFINPUT = keyword
D TZ {=|:} [timezone]
D [NO]USEREXIT
D [NO]USERISN
D WCHARSET = char_set
```

DBID

```
DBID = number
```

This parameter selects the database that contains the file to be specified by the FILE parameter.

[NO]DST

[NO]DST

The parameter DST is required if a daylight saving time indicator is provided for date/time fields with the option TZ. The daylight saving time indicator must be appended behind the date/time value as a 2-byte integer value (format F) that contains the number of seconds to be added to the standard time in order to get the actual time (usually 0 or 3600).

Without the parameter DST, it is not possible to define time values in the hour before the time is switched back to standard time.

The default is NODST.



Notes:

1. The DST parameter is ignored if the FIELDS parameter is specified. In this case, you must specify a D element for fields with the daylight saving time indicator.
2. The DST parameter is not compatible with the RECORD_STRUCTURE = NEWLINE_SEPARATOR parameter because the daylight saving indicator in format F contains non-printable characters.

Example:

A DT field has the following definition: 1,DT,8,P,DT=E(DATE_TIME),TZ

The following values must then be specified for this field:

- The local date/time value corresponding to the edit mask DATE_TIME as an 8-byte packed value
- The daylight saving time indicator, usually 0 for standard time and 3600 for summer time as a 2-byte fixed point value

```
Case 1 (DT has a date/time value with daylight saving time): 0x0200910250230000E10  
Case 2 (DT has a date/time value with standard time): 0x0200910250230000000
```

FDT

FDT

If this parameter is the first parameter that is specified, ADACMP reads the FDT information contained in the sequential file CMPFDT. If it is specified in conjunction with the DBID and FILE parameters, the FDT of the specified file is displayed.

FIELDS

```
FIELDS {uncompressed_field_definition | FDT}...{END_OF_FIELDS | . }
```

This parameter is used to specify a subset of fields given in the FDT and their format and length. This means that the input records do not have to contain all of the fields given in the FDT, or that fields can be provided with a different format or length. The syntax and semantics are the same as for the format buffer, with the exception that you can also specify an R-element (for LOB references) if the decompressed record contains the name of a file containing the LOB value instead of the LOB value itself. See *Administration, Loading and Unloading Data, Uncompressed Data Format* for more detailed information.

While entering the specification list, the FDT function can be used to display the FDT of the file to be decompressed. The specification list can be terminated or interrupted by entering END_OF_FIELDS or `.`. The `.` option is an implicit END_OF_FIELDS and is compatible with the format buffer syntax. FIELDS or END_OF_FIELDS must always be entered on a line by itself, whereas the `.` may be entered on a line by itself or at the end of the format buffer elements.

If the field definitions are terminated with the END_OF_FIELDS parameter, this parameter must be specified in upper case when the LOWER_CASE_FIELD_NAMES parameter is used. In addition, the FDT parameter must also be specified in upper case when the LOWER_CASE_FIELD_NAMES parameter is used.

FILE

```
FILE = number
```

This parameter specifies the file from which the FDT information is to be read. This parameter can only be specified after the DBID parameter.

[NO]LOWER_CASE_FIELD_NAMES

```
[NO]LOWER_CASE_FIELD_NAMES
```

If LOWER_CASE_FIELD_NAMES is specified, Adabas field names are not converted to upper case. If NOLOWER_CASE_FIELD_NAMES is specified, Adabas field names are converted to upper case. The default is NOLOWER_CASE_FIELD_NAMES.

This parameter must be specified before the FIELDS parameter.

[NO]LOBS

[NO]LOBS

This parameter specifies whether LA and LB field values are to be stored in a LOB file after loading the compressed data into the database:

- If the parameters DBID and file number have been specified, this parameter is ignored, and the field is handled as described below;
- If the parameters DBID and file number have not been specified and LOBS is specified, field values for LA and LB fields are prepared for storage in a LOB file, except the field is defined as a descriptor.
- If the parameters DBID and file number have not been specified and NOLOBS is specified, field values for LA and LB fields are prepared for storage in the base file. In this case, the length of field values for LA and LB fields must not exceed 16381 bytes and the compressed record must fit into a 32 KB DATA block.

Please note that LA and LB fields which are descriptors or parent fields of a derived descriptor, e.g. a super descriptor, are always handled as described for the NOLOBS parameter.

Default behaviour is as follows:

- If the parameters DBID and file number have been specified and the file is a base file with corresponding LOB file, LOBS is default.
- If the parameters DBID and file number have been specified and the file is not a base file with corresponding LOB file, NOLOBS is default.
- If the parameters DBID and file number have not been specified, LOBS is default.

MAX_DECOMPRESSED_SIZE

```
MAX_DECOMPRESSED_SIZE = number [K|M]
```

This parameter specifies the maximum size of a decompressed record in bytes, kilobytes or megabytes, depending on the specification of "K" or "M" after the number. This parameter is intended to recognize invalid CMPIN files as early as possible.

The default is 65536. This is also the minimum value.



Notes:

1. This parameter does not include the size of LOB values stored in separate files.
2. The exact definition of this parameter is the size of the I/O buffer required for the largest decompressed record. Only multiples of 256 bytes are used for the I/O buffers, which means that you must specify a value greater than or equal to the largest decompressed record (including the preceding length field) rounded up to the next multiple of 256.

MUPE_C_L

```
MUPE_C_L = {1|2|4}
```

If the uncompressed data contain multiple-value fields or periodic groups, they are preceded by a binary count field with the length of MUPE_C_L bytes.

The default is 1.

[NO]NULL_VALUE

```
[NO]NULL_VALUE
```

The parameter NULL_VALUE is required if you are compressing data according to the standard FDT and the status values of the NC option fields are given in the input data. Normally, such input data is generated by ADADCU with the NULL_VALUE option set.

The default is NONNULL_VALUE.

Example

The definition in the FDT for the field AA is: 1, AA, 2, A, NC

Case 1 (AA has a non-NULL value): input record (hexadecimal) = 00004142

Case 2 (AA has a NULL value): input record (hexadecimal) = FFFF2020

NUMREC

```
NUMREC = number
```

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

Use of this parameter is recommended for the initial execution of ADACMP if the input data file contains a large number of records. This avoids unnecessary processing of all records in cases where a data 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.

RECORD_STRUCTURE

RECORD_STRUCTURE = keyword

This parameter specifies the type of record separation used in the input file with the environment variable CMPIN. The following keywords can be used:

Keyword	Meaning
ELENGTH_PREFIX	The records in the CMPIN file are separated by a two-byte exclusive length field.
E4LENGTH_PREFIX	The records in the decompressed data file are separated by a 4-byte exclusive length field.
ILENGTH_PREFIX	The records in the CMPIN file are separated by a two-byte inclusive length field.
I4LENGTH_PREFIX	The records in the decompressed data file are separated by a 4-byte inclusive length field.
NEWLINE_SEPARATOR	The records in the CMPIN file are separated by a new-line character. This keyword may only be specified if the field values do not contain characters interpreted as new-line (i.e. if there are only unpacked, alphanumeric and Unicode fields, and the alphanumeric and Unicode fields contain only printable characters). This keyword and the USERISN parameter are mutually exclusive.
RDW	<p>The records in the CMPIN file contain data that has been transferred from an IBM host using the FTP <i>site rdw</i> option. ADACMP is able to process such data without having to use <i>cvt_fmt</i> first (in previous versions, the unsupported tool <i>cvt_fmt</i> was used for such format conversions). For example:</p> <pre>% ftp IBM-host ftp> binary 200 Representation type is Image ftp> site rdw 200 Site command was accepted ftp> get decomp % setenv CMPIN decomp % adacmp fdt record_structure=rdw source=(ebcdic,high)</pre>
RDW_HEADER	Like RDW, for data decompressed on a mainframe with HEADER=YES.
VARIABLE_BLOCKED	The variable blocked format from BS2000 or IBM.

The default is ELENGTH_PREFIX.

SEPARATOR

```
SEPARATOR = character | \character
```

If you specify this option, ADACMP expects the fields in the raw data record to be separated by the character specified. You can omit the apostrophes round the character specification if the character has no special meaning for the Adabas utilities. The same fields in different records are then permitted to be of different lengths.

If a format buffer is specified using the FIELDS parameter, the order of the specified field names must correspond with the order in which the fields are specified in the FDT. A mismatch results if this is not the case.

If the FDT contains multiple value fields or periodic groups, a format buffer must be specified with the FIELDS parameter. Members of periodic groups must be ordered by 1) periodic group index and 2) field sequence in the FDT (see example 2 below).

Because no binary data is expected in the input file using the SEPARATOR option, the RECORD_STRUCTURE parameter will be set to NEWLINE_SEPARATOR.

Example 1

```
FDT:      1, AA, 2, U
          1, AB, 8, U
          1, AC, 2, A

CMPIN:    12;12345678;AA
          1234;5;BB

adacmp
fdt
separator=\;

or for UNIX

adacmp fdt separator=\\;

or

adacmp fdt separator='\;'
```

In this example, 2 records are compressed with the default FDT, the separator character is the semicolon, and the default record structure is NEWLINE_SEPARATOR. Note that the semicolon must be preceded by a backslash, otherwise it would be treated as the start of a comment. If you enter the parameters under UNIX directly from the command line, it is necessary to precede the backslash and the semicolon by additional backslashes or to put them in quotes or double quotes since they are special characters.

Example 2

```

FDT:      1, XX, PE
          2, AA, 8, A
          2, AB, 8, U
          1, YY, 2, A

Correct:  CMPIN:   aaaa,1,bbbb,2,yy

          Command: adacmp fdt separator=, fields AA1,AB1,AA2,AB2,YY.
          First, the field values for the periodic group index 1 are
          specified, and then the field values for periodic group index 2.

Invalid:  CMPIN:   aaaa,bbbb,1,2,yy
          Command: adacmp fdt separator=, fields AA1-2,AB1-2,YY.
          The fields specification is invalid because the 2nd value of
          AA is specified before the 1st value of AB; you will get
          the error SEPINV.

```

In this example, 1 record with fields given in the format buffer is compressed, the separator character is the comma.

Example 3

```

FDT:      1, AA, 8, A
          1, MA, 1, A, MU

CMPIN:    aaaa%2%A%B
          bbbb%3%C%D%E

adacmp dbid=9 file=15 separator=%, fields "AA,MAC,1,U,MA1-N"

```

In this example, 2 records with fields given in the format buffer are compressed, the occurrence count or the multiple value field MA is different in different records. The separator character is the percent character.

[NO]SHORT_RECORDS

```
[NO]SHORT_RECORDS
```

If `SHORT_RECORDS` is specified, it is possible to omit fields at the end of the decompressed record that contain null values.

The default is `NOSHORT_RECORDS`.

You can only omit complete fields; it is not possible to truncate the last value:

Example

Assuming you have specified the parameters for a file containing alphanumeric fields AA and AB:

```
FIELDS
AA,20,AB,20
END_OF_FIELDS
SHORT_RECORDS
```

Then the following record is allowed:

```
"Field AA          "
```

The following record is not allowed:

```
"Field AA"
```

[NO]SINGLE_FILE

```
[NO]SINGLE_FILE
```

If the SINGLE_FILE option is set, ADACMP writes the Descriptor Value Table (DVT) and the compressed user data to a single file (CMPDTA) instead of writing them to separate files.

The default is NOSINGLE_FILE.

SKIPREC

```
SKIPREC = number
```

This parameter specifies the number of records to be skipped before compression is started.

SOURCE_ARCHITECTURE

```
SOURCE_ARCHITECTURE = ( keyword [,keyword [,keyword] ] )
```

This parameter specifies the format (character set, floating-point format and byte order) of the input data records. The following keywords can be used:

Keyword Group	Valid Keywords
Character set	ASCII
	EBCDIC
Floating-point format	IBM_370_FLOATING
	IEEE_FLOATING

Keyword Group	Valid Keywords
	VAX_FLOATING
Byte order	HIGH_ORDER_BYTE_FIRST LOW_ORDER_BYTE_FIRST

If no keyword of a keyword group is specified, the default for this keyword group is the keyword that corresponds to the architecture of the machine on which ADACMP is running.



Note: The FDT is always input in ASCII format.

Example

If the input records that are to be compressed are in IBM format, the user must specify the following:

```
SOURCE_ARCHITECTURE = (EBCDIC, IBM_370_FLOATING, HIGH_ORDER_BYTE_FIRST)
```

SYFINPUT

```
SYFINPUT = keyword
```

This parameter specifies the input used for the compression of system generated fields. The following keywords can be used:

Keyword	Meaning
SYSTEM	The system generated field values are regenerated by the system in ADACMP.
USER	The system generated field values are taken from the decompressed file.

The default is SYFINPUT = USER.

TZ

```
TZ {=|:} [timezone]
```

The specified time zone must be a valid time zone name that is contained in the time zone database known as the Olson database (<http://www.twinsun.com/tz/tz-link.htm>). If a time zone has been specified, this time zone is used for time zone conversions of date/time fields with the option TZ.

The default is UTC, which is used internally to store date/time fields with option TZ; no conversion is required.

If you specify an empty value, no checks are made to ensure that date/time fields are correct.



Note: The time zone names are file names. Depending on the platform, these file names may or may not be case sensitive. Also, the time zone names, depending on the platform, may or may not be case sensitive.

Examples:

```
tz:Europe/Berlin
```

This is correct on all platforms.

```
TZ=Europe/Berlin
```

With this specification, TZ is converted to upper case EUROPE/BERLIN. This is correct on Windows, because file names are not case sensitive on Windows, but it is not correct on Unix, because Unix file names are case sensitive.

[NO]USEREXIT

```
[NO]USEREXIT
```

This option specifies whether a user exit is to be taken or not. If USEREXIT is specified, the environment variable ADAUEX_6/logical name ADABAS\$USEREXIT_6 must point to a loadable user-written routine.

See *Administration, User Exits and Hyperexits* for more details.

The default is NOUSEREXIT.

[NO]USERISN

```
[NO]USERISN
```

If this option is set to USERISN, the ISN for each record in the input file will be assigned by the user.

If USERISN is specified, the user must give the ISN to be assigned to each record as a four-byte binary number immediately preceding each data record.

ISNs may be assigned in any order and must be unique (for the file). The ISN must not exceed the maximum number of records (MAXISN) specified for the file (see the file definition utility *ADAFDU* for more detailed information).

ADACMP does not check for unique ISNs or for ISNs which exceed MAXISN. These checks are performed by the mass update utility ADAMUP (if an error is detected, the ADAMUP run terminates with an error message).

If this option is set to NOUSERISN, the ISN is assigned by Adabas.

The default is NOUSERISN.

WCHARSET

```
WCHARSET = char_set
```

This parameter specifies the default encoding used in the decompressed file based on the encoding names listed at <http://www.iana.org/assignments/character-sets> - most of the character sets listed there are supported by ICU, which is used by Adabas for internationalization support.

The default is UTF-8.

Output

The ADACMP utility outputs three files:

1. Compressed data
2. Descriptor values
3. Records with errors

Compressed Data Records

The data records which ADACMP has processed, modified and compressed are output together with the FDT information to a sequential file. This file is used as input for the mass update utility ADAMUP.

If the output file contains no records (no records on the input file or all records rejected), the output may still be used as input for the mass update utility ADAMUP.

Descriptor-Value Table File

This file contains the descriptor value tables (DVT).

Compressed data records and descriptor value tables are written to one file if the SINGLE_FILE option is specified.

Rejected Data Records

Any records rejected by ADACMP are written to the ADACMP error file. The contents of this error file should be displayed using the ADAERR utility. Do not print the error file using the standard operating system print utilities since the records contain unprintable characters.

See the *ADAERR* utility for further information.

Report

The ADACMP report begins with a display of the field definition entered if CMPFDT is used for input. Any statement which contains a syntax error will be printed with a message immediately following the statement.

Following the display of the data-definition statements, a descriptor summary, the number of input records processed, the number of input records rejected, and the number of input records compressed are printed.

Restart Considerations

ADACMP does not have a restart capability. An interrupted ADACMP run must be re-started from the beginning.

ADACMP does not change the database; therefore, no considerations need to be made concerning database status before restarting ADACMP.

7 ADADBA (DBA Workbench)

- Functional Overview 68
- Procedure Flow 69

This chapter describes the utility "ADADBA".

Functional Overview

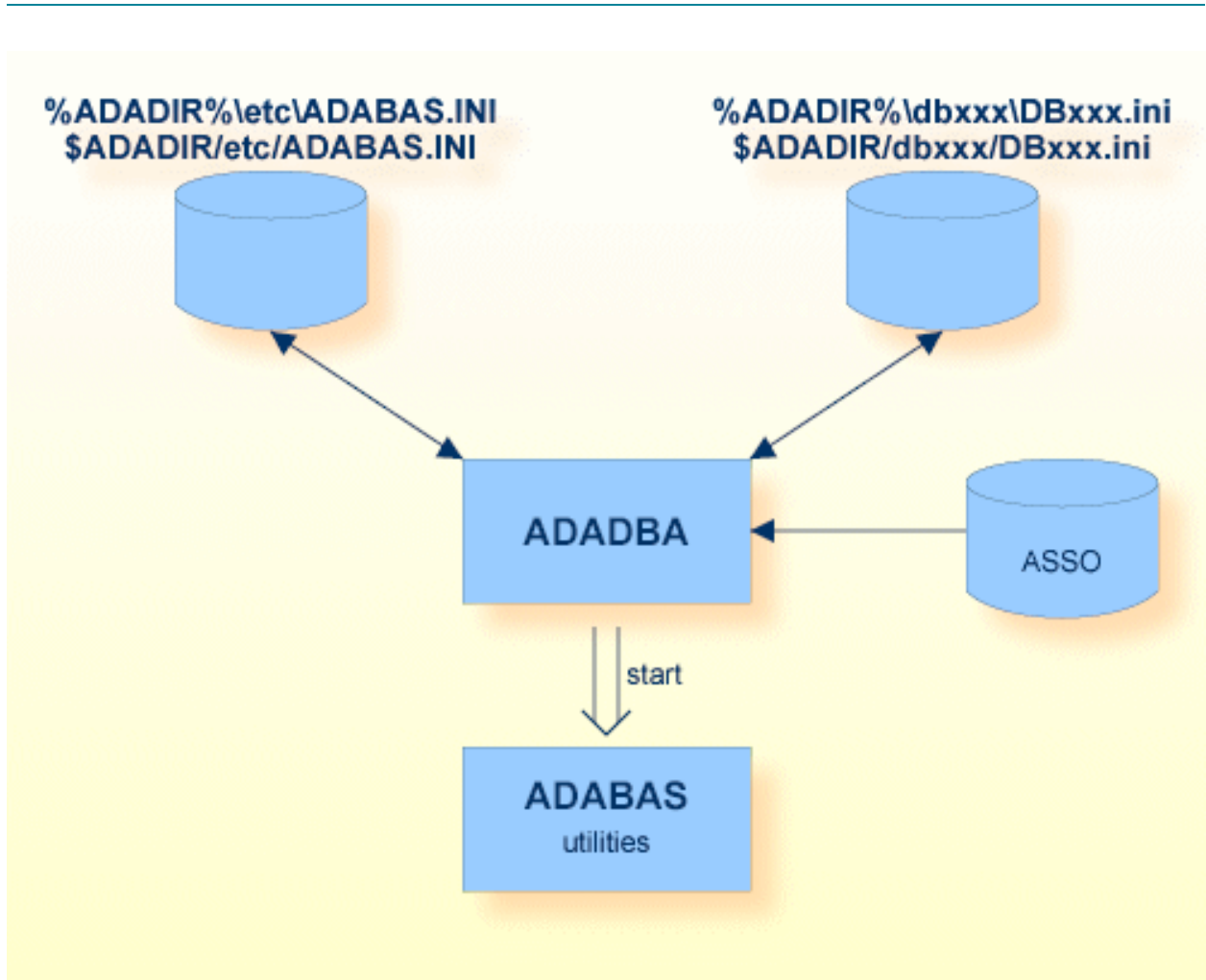
The DBA Workbench is a graphical user interface that is used to create, operate and maintain Adabas databases and files on the local node.

The Adabas Workbench is no longer supported starting with Version 6.2. On those platforms for which the DBA Workbench was delivered for Version 6.1, the Workbench will still be delivered, but the functionality is frozen to the functionality that was available with Version 6.1. New features that were introduced for Version 6.2, such as date/time fields, are not supported.

The DBA Workbench can be started either from the command line in the same manner as any other utility (UNIX and Windows), or by clicking the icon for the DBA Workbench in the Adabas folder on the desktop (Windows only).

Additional information is available as context-sensitive online help when the DBA Workbench is running.

Procedure Flow



8

ADADB (Database Modification)

- Functional Overview 72
- Procedure Flow 74
- Checkpoints 76
- Control Parameters 77
- Restart Considerations 100

This chapter describes the utility "ADADBM".

Functional Overview

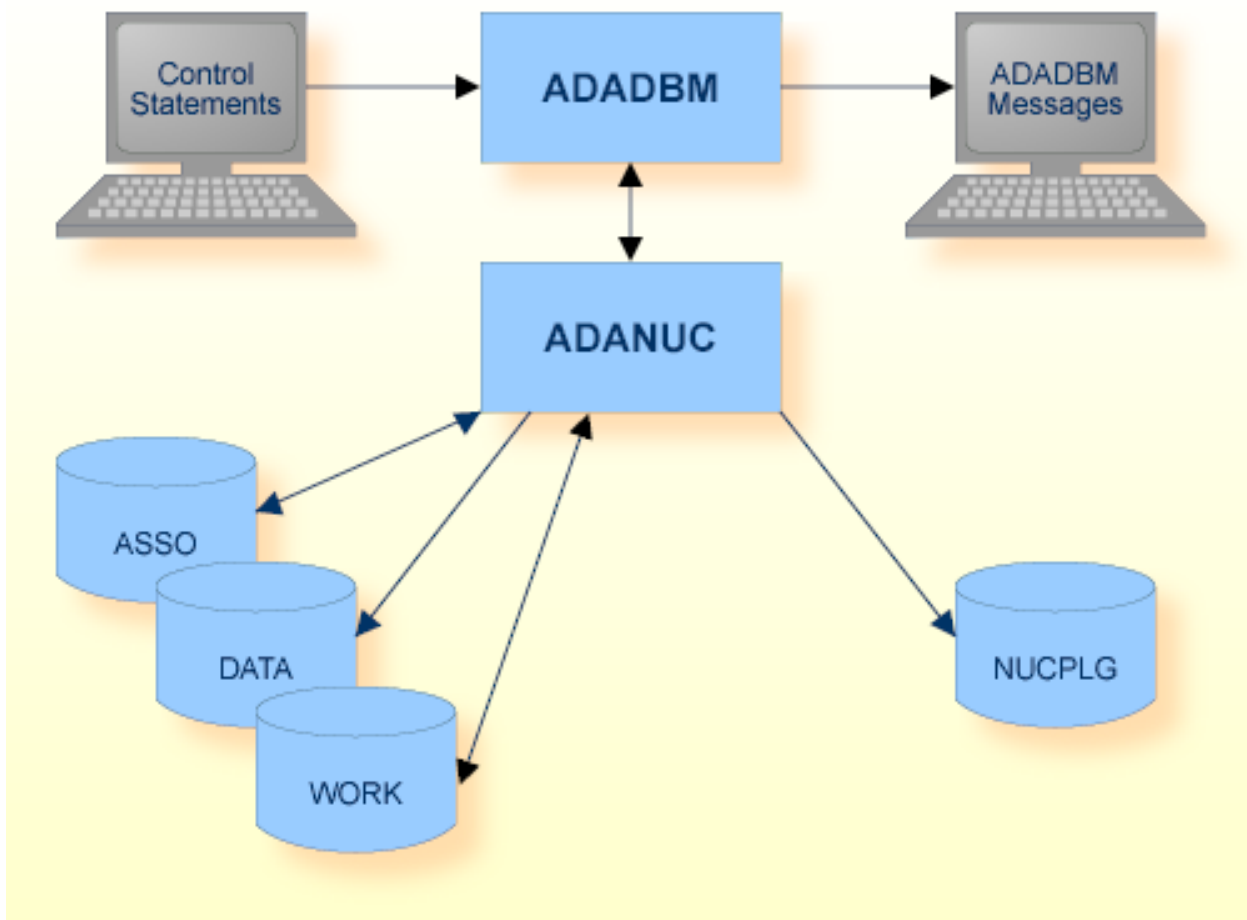
The ADADBM utility consists of the following functions which may be used to make modifications to the database:

- The ADD_CONTAINER function adds a new container file to the Associator or Data Storage data set;
- The ADD_FIELDS function adds new fields to the end of a file's FDT;
- The ALLOCATE NI, UI, AC or DS function increases the Normal Index, Upper Index, Address Converter or Data Storage space assigned to a file;
- The CHANGE function changes the standard length of a field in the Field Definition Table (FDT);
- The CHANGE_FIELDS function modifies one or more field specifications in a file;
- The DEALLOCATE functions are the inverse functions of ALLOCATE. The NI, UI, AC or DS function returns the Normal Index, Upper Index, Address Converter or Data Storage space which is no longer required by a file to the free space table (FST);
- The DELCP function deletes old checkpoint records from the checkpoint file in the specified range of dates;
- The DELETE function deletes a single Adabas file or a range of Adabas files from the database;
- The DISPLAY function displays the utility communication block (UCB);
- The DROP_FIELDS function marks the specified fields as not existing, which means that they can no longer be accessed ;
- The DROP_LOBFILE function is the inverse function of ADAFDU ADD_LOBFILE;
- The DROP_REFINT function drops an existing referential constraint;
- The EXTEND_CONTAINER function extends the last container file defined for the database;
- The NEW_DBID function changes the identifier of the database in use;
- The NEWWORK function allocates and formats a new Adabas WORK data set;
- The PGM_REFRESH function enables or disables refreshing an Adabas file inside an application program with an E1 command;
- The RECOVER function returns lost space to the free space table;
- The REDUCE_CONTAINER function reduces the size of the last container file defined for the database;
- The REFRESH function resets a single file or a range of files to the state of zero records loaded;

- The REMOVE_CONTAINER function removes a container file from the Associator, or Data Storage data set;
- The REMOVE_DROP function, used in conjunction with a subsequent REFRESH, removes dropped fields from the FDT;
- The REMOVE_REPLICATION function stops all replication processing and deletes the replication system files;
- The RENAME function changes the database name or names of loaded files;
- The RENUMBER function renumbers a loaded file or exchanges the numbers of loaded files;
- The REPLICATION_FILES function creates the system files required for Adabas - Adabas replication;
- The RESET function removes entries from the UCB;
- The RESET_REPLICATION_TARGET function resets the replication target flag of Adabas files;
- The REUSE function controls the reuse of Data Storage space or ISNs by Adabas;
- The SYFMAX function specifies the maximum number of values generated for a system generated multiple-value field in the file specified.

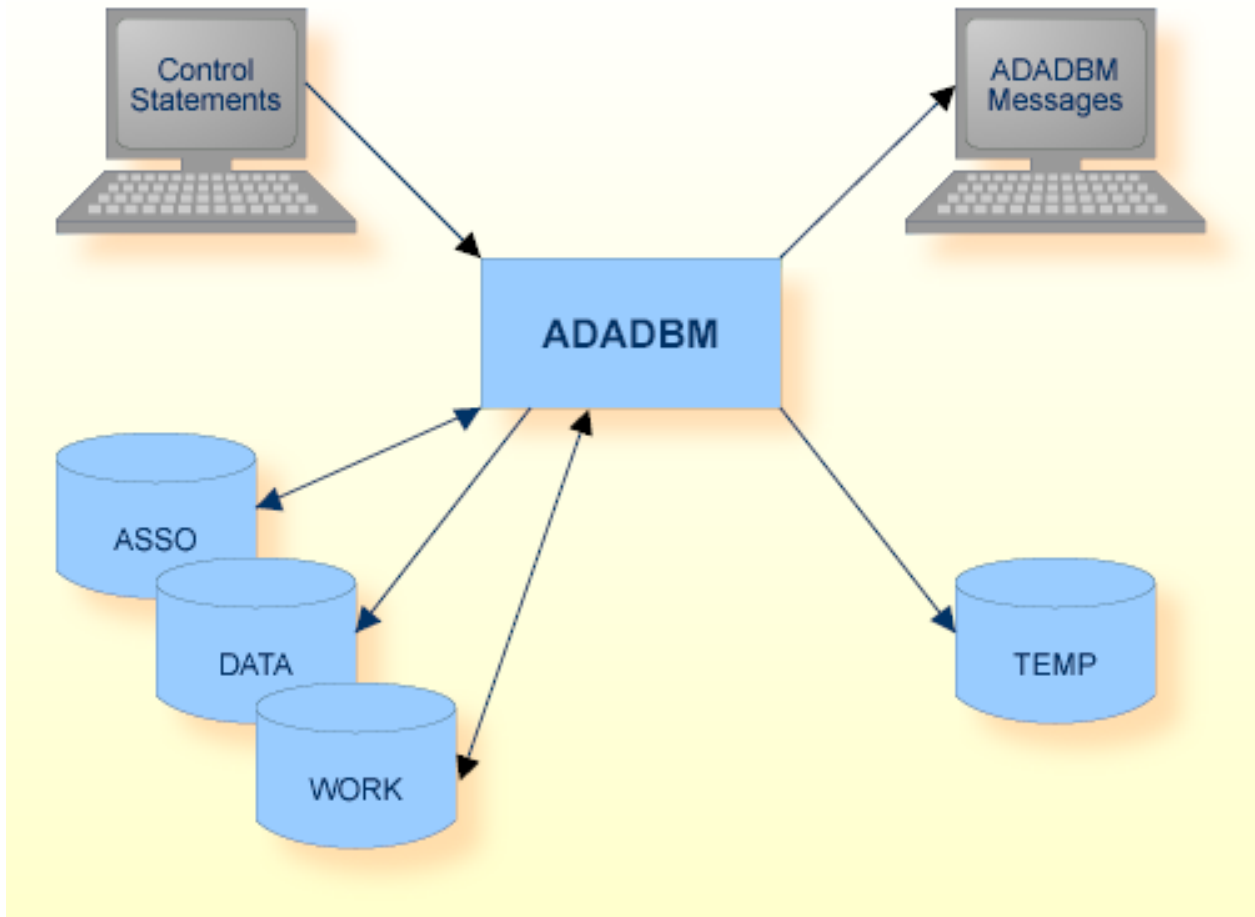
This utility is a multi-function utility.

Procedure Flow



Online Mode

If the Adabas nucleus is active, ADADBM calls the nucleus to modify the database containers. For some tasks, no checkpoints are written, but the activity is logged in the database log, and in the case of a recovery, the activity is re-executed automatically.



Offline Mode

If the Adabas nucleus is not active, ADADBМ itself modifies the database containers.

Data Set	Environment Variable/ Logical Name	Storage Medium	Additional Information
Associator	ASSOx	Disk	
Data storage	DATAx	Disk	
Control statements	stdin		Utilities Manual
ADADBМ messages	stdout		Messages and Codes
Protection Log (online mode only)	NUCPLG	Disk	Utilities Manual: ADANUC, ADAPLP
Temporary storage (offline mode only)	TEMP1	Disk	New WORK data set for the NEWWORK function. After this function is performed, the Work environment variable/logical name must be changed to point to the new Work data set.
Work	WORK1	Disk	

Checkpoints

The following table shows the nucleus requirements for each function and the checkpoints written:

Function	Nucleus must be active	Nucleus must NOT be active	Nucleus is NOT required	Checkpoint written
ADD_CONTAINER			X	SYNP
ADD_FIELDS			X	SYNP (offline) SYNX (online)
ALLOCATE			X	SYNP
CHANGE	X			-
CHANGE_FIELDS			X	SYNP (offline) SYNX (online)
DEALLOCATE			X	SYNP
DELCP	X			SYNP
DELETE			X	SYNP (offline) SYNX (online)
DISPLAY			X	-
DROP_FIELDS			X	SYNP (offline) SYNX (online)
DROP_LOBFILE			X	SYNP
EXTEND_CONTAINER		for WORK	for ASSO or DATA	SYNP
NEW_DBID		X (see note 1)		SYNP
NEWWORK		X (see note 1)		SYNP
PGM_REFRESH			X	SYNP
RECOVER			X	SYNP
REDUCE_CONTAINER			for ASSO or DATA	SYNP
REFRESH			X	SYNP
REMOVE_CONTAINER		X		SYNP
REMOVE_REPLICATION		X		SYNP (offline)
RENAME			X	SYNP
RENUMBER			X	SYNP
REPLICATION_FILES			X	SYNP (offline) SYNX (online) (see note 2)

Function	Nucleus must be active	Nucleus must NOT be active	Nucleus is NOT required	Checkpoint written
RESET			X	SYNX
RESET_REPLICATION_TARGET				
REUSE	X			SYNP
SYFMAX			X	SYNP (offline) SYNX (online)

**Notes:**

1. Function requires exclusive access to the database container files
2. In addition, ADADB or ADAFDU checkpoints are generated (also in offline mode) to indicate the system file numbers deleted or generated.

Control Parameters

The following control parameters are available:

```

ADD_CONTAINER = keyword
D      [,BLOCKSIZE=number[K] ]
      ,SIZE = number [B|M]

ADD_FIELDS = number {field_specification|FDT} ... END_OF_FIELDS

ALLOCATE = keyword, FILE = number [,RABN = number],
          SIZE = number [B|M]

CHANGE = number, FIELD = string, LENGTH = number

CHANGE_FIELDS = number {field_specification|FDT} ... END_OF_FIELDS
M

DBID = number

DEALLOCATE = keyword, FILE = number [,RABN = number],
           SIZE = numberB

DEFINE_REFINT = number constraint_specification

DELCP = { * | ([absolute-date] [, [absolute-date]]) }

DELETE = (number [-number][, number[-number]]...)

DISPLAY = UCB

```

```
DROP_FIELDS = number {field_name|FDT} ... END_OF_FIELDS
DROP_LOBFILE = number
DROP_REFINT = number, NAME {=:}constraint_name
EXTEND_CONTAINER = keyword, SIZE = number [B|M]
D [NO]LOWER_CASE_FIELD_NAMES
NEW_DBID = number
NEWWORK [,BLOCKSIZE=number[K] ], SIZE = number [B|M]
PGM_REFRESH = keyword, FILE = number
RECOVER
REDUCE_CONTAINER = keyword, SIZE = number B
REFRESH = (number [-number][,number[-number]]...)
REMOVE_CONTAINER = keyword
[NO]REMOVE_DROP
REMOVE_REPLICATION
RENAME = number, NAME {=:} string
RENUMBER = (number, number)
REPLICATION_FILES = (file1, file2, file3, file4)
RESET = UCB, IDENT = { (number [,number]...) | * }
RESET_REPLICATION_TARGET = number
REUSE = (keyword [,keyword]), FILE = number
SYFMAX = number, FILE = number
```

ADD_CONTAINER

```
ADD_CONTAINER = keyword
                [,BLOCKSIZE=number[K] ]
                ,SIZE = number [B|M]
```

The ADD_CONTAINER function adds a new container file to an existing Associator or Data Storage dataset in accordance with the keyword used. The keyword can take the values ASSO or DATA .

The new container file may be allocated on the same device as the current container files or it may be allocated on a different device type.

The placement of the new container file depends on the environment variable/logical name ASSOx or DATAx. This has to be set to a legal file name with its whole path name. If ASSOx or DATAx is not set, the container files are created in the current directory.



Notes:

1. If you add a new DATA container, free space is required on the existing Associator for the Data Storage Space Table (DSST): 1 byte is required for each new DATA block. Therefore, you should add a new ASSO container first if the existing Associator is full.
2. If you add a new container while the Adabas nucleus is active, the container is not added by ADADBM, but by ADANUC. In this case, you should consider the following, depending on the way the container files are specified: if the container files are specified as environment variables/logical names, you can only add the new container if you have already defined the corresponding environment variable/logical name for the current nucleus session. If you haven't, you have to terminate the current nucleus session before you can add the new container. If, on the other hand, the container files are specified in the *DBnnn.INI* files (which, for example, are generated if you use the Workbench to administrate Adabas), you can enter the new container in the file just before you add the container, since the nucleus re-reads the file before the container is added (please refer to the *Adabas Extended Operation* for further information about the *DBnnn.INI* files).

BLOCKSIZE = number[K]

This parameter specifies the block size in bytes (or in kilobytes, if "K" is specified after the number) of the new container file.

Adabas rounds up the value you specify to the next multiple of 1K. The minimum block size is 1K and maximum block size is 32K.

The default value for BLOCKSIZE is the block size of the last container file of the dataset in question that is currently present in the database.

SIZE = number [B|M]

This parameter specifies the number of blocks (B) or megabytes (M) to be allocated for the new container file. By default, the size is given in megabytes.

Example

```
adadbm: add_container=data, size=10
%ADADBM-I-CREATED, dataset DATA2 , file /FS/fs0395/Adabas/adadb/db076/DATA2 created
%ADADBM-I-FUNC, function ADD_CONTAINER executed
```

A new container file of 10 megabytes is added to the Data Storage. The block size is the same as the block size of DATA1.

ADD_FIELDS

```
ADD_FIELDS = number {field_specification|FDT}... END_OF_FIELDS
```

The ADD_FIELDS function adds one or more new fields to the end of the file defined by 'number'. Specifying a LOB file is not permitted. The function is completed by entering END_OF_FIELDS.

If the field definitions are terminated with the END_OF_FIELDS parameter, this parameter must be specified in upper case when the LOWER_CASE_FIELD_NAMES parameter is used. In addition, the FDT parameter must also be specified in upper case when the LOWER_CASE_FIELD_NAMES parameter is used.



Note: It is not possible to add derived descriptors using ADADBM - you should use the utility ADAINV to do this instead.

field_specification

The field specification list is entered in the same way as the FDT input in ADAFUDU:

```
level-number, name [,length] [,format] [(,option)...]
```

The first field to be added must be a level-one field.

The NN option is not permitted. DE is only permitted when the Adabas nucleus is active and together with the NU or NC option. Otherwise use the ADAINV utility to give the new fields descriptor status. UQ is only permitted together with the DE option.



Note: When you add system-generated fields (fields with the field option SY) to a file, these fields have null values in the records that are already in the database - this is the same behaviour as for fields without the SY option.

FDT

This parameter displays the FDT of the file to which the fields are to be added.

Example

```
adadbm: add_fields=12
adadbm: fdt
```

Field Definition Table:

Level	I Name	I Length	I Format	I Options	I Flags
1	I AA	I 15	I A	I DE,UQ,NU	I
1	I AB	I 4	I F	I FI	I
1	I AC	I 8	I A	I DE	I
1	I CD	I	I	I	I
2	I AD	I 20	I A	I DE,NU	I SP
2	I AE	I 20	I A	I NU	I
2	I AF	I 10	I A	I DE,NU	I
1	I AG	I 2	I U	I NU	I SP
1	I AH	I 1	I A	I DE,FI	I
1	I AI	I 1	I A	I FI	I
1	I AJ	I 6	I U	I NU	I SP
1	I AK	I	I	I	I
2	I AL	I 3	I A	I NU	I
2	I AM	I 4	I P	I NU,MU	I

Type	I Name	I Length	I Format	I Options	I Parent field(s)	Fmt
SUPER	I AN	I 4	I B	I NU	I AJ (5 - 6)	U
	I	I	I	I	I AJ (3 - 4)	U
SUPER	I A0	I 22	I A	I NU	I AG (1 - 2)	U
	I	I	I	I	I AD (1 - 20)	A

```
adadbm: 01,dd,1,a
adadbm: 01,gr
adadbm: 02,g1,20,a,fi
adadbm: fdt
```

Field Definition Table:

Level	I Name	I Length	I Format	I Options	I Flags
1	I AA	I 15	I A	I DE,UQ,NU	I
1	I AB	I 4	I F	I FI	I
1	I AC	I 8	I A	I DE	I

1	I	CD	I		I	I		I	
2	I	AD	I	20	I	A	I DE,NU	I	SP
2	I	AE	I	20	I	A	I NU	I	
2	I	AF	I	10	I	A	I DE,NU	I	
1	I	AG	I	2	I	U	I NU	I	SP
1	I	AH	I	1	I	A	I DE,FI	I	
1	I	AI	I	1	I	A	I FI	I	
1	I	AJ	I	6	I	U	I NU	I	SP
1	I	AK	I		I		I	I	
2	I	AL	I	3	I	A	I NU	I	
2	I	AM	I	4	I	P	I NU,MU	I	
1	I	DD	I	1	I	A	I	I	
1	I	GR	I		I		I	I	
2	I	G1	I	20	I	A	I FI	I	

Type	I	Name	I	Length	I	Format	I	Options	I	Parent field(s)	Fmt
SUPER	I	AN	I	4	I	B	I	NU	I	AJ (5 - 6)	U
	I		I		I		I		I	AJ (3 - 4)	U
SUPER	I	A0	I	22	I	A	I	NU	I	AG (1 - 2)	U
	I		I		I		I		I	AD (1 - 20)	A

adadbm: end_of_fields
 %ADADBM-I-FUNC, function ADD_FIELDS executed

ALLOCATE

ALLOCATE = keyword, FILE = number [,RABN = number], SIZE = number [B|M]

Depending on the keyword specified (AC, DS, NI or UI), the ALLOCATE function increases the Normal Index (NI), Upper Index (UI), Address Converter (AC) or Data Storage (DS) by a given size. Each extent for the required type is checked to see whether it can be extended or not. A new extent is created if none of the current extents can be extended.

This function lets the DBA override the automatic extension method and can be used to preallocate smaller or larger extents. This can be useful when adding a large number of records. Exclusive control of the file is NOT required for this function.

FILE = number

This parameter specifies the file to be extended.

RABN = number

This parameter specifies the allocation start RABN. For NI or UI allocation for a LOB file, the block size of the RABN specified must be less than 16 KB. For DS allocation for a LOB file, the block size of the RABN specified must be 32 KB.

SIZE = number [B|M]

This parameter specifies the size of the expansion area. If a 'B' is appended to size, the size is in blocks, otherwise it is in megabytes.

Example

```
adadbm: allocate=ni, file=11, size=100b
%ADADBM-I-ALLOC, 100 NI blocks allocated (611 - 710)

adadbm: allocate=ds, file=11, size=10
%ADADBM-I-DEALLOC, 2560 DS blocks allocated (245 - 2804)
```

CHANGE

```
CHANGE = number, FIELD = string, LENGTH = number
```

This function changes the standard length of a field in the file specified by number. Specifying a LOB file is not permitted. The length of fixed storage fields (option FI) and floating point fields (format G) cannot be changed.

Changing the length of a field does not lead to any modifications within the Data Storage, but may affect programs that use the standard length.

Fields defined with the option SY=OPUSER cannot be changed.

FIELD = string

This parameter specifies the field whose standard length is to be changed. The field must be defined in the Field Definition Table for this file.

LENGTH = number

This parameter defines the new standard length of the field.

Example

```
adadbm: change=12, field=ac, len=11
%ADADBM-I-FUNC, function CHANGE executed
```

CHANGE_FIELDS

```
CHANGE_FIELDS = number {field_specification|FDT}... END_OF_FIELDS
```

The CHANGE_FIELDS function modifies one or more field specifications of the file defined by 'number'. The function is completed by entering END_OF_FIELDS.

If the field definitions are terminated with the END_OF_FIELDS parameter, this parameter must be specified in upper case when the LOWER_CASE_FIELD_NAMES parameter is used. In addition, the FDT parameter must also be specified in upper case when the LOWER_CASE_FIELD_NAMES parameter is used.

The changes that are allowed depend on the existence of records in the file. The following restrictions apply to all files:

- The field level number must not change;
- A group must either remain a group or may be converted to a periodic group if it is defined on level 1;
- A periodic group must either remain a periodic group or may be converted to a non-periodic group;
- A field that is not a group or periodic group must not be converted to a group or a periodic group.

The following additional restrictions apply to non-empty files:

- Field length: the new length must be compatible with the new field format and field options. Such a change changes the behaviour of adabas commands in which the field length is not specified in the format buffer;
- Field format: A may be changed to W and vice versa. It is the responsibility of the user to ensure that the field contains UTF-8 values if the format is changed from A to W. After changing the

format from W to A, the field will contain UTF-8 values. Please note that the format specified in the format buffer of Adabas commands must be identical to the format in the field definition for A and W fields - therefore it may be necessary to adapt existing programs accordingly. Other changes of the field format except for the change between A and W are not allowed.

- Field options: it is not allowed to add or remove the options DE, FI, MU and UQ.

The following field option changes are allowed:

Old Field Options	New Field Options	Comments
DT not set	DT set, TZ not set or set	No check is made to see whether the values in the database are compliant with the date/time edit mask specified. TZ may not be set for edit mask names DATE, TIME and NATDATE. Caution: Usually, the semantics of the field values defined with the TZ option and the field values defined without the TZ option are different: the field without the TZ value usually contains local time values, whereas the field with the TZ value contains UTC values. The field values are not updated automatically; it is the user's responsibility to ensure that necessary updates are made.
DT set	DT not set	Specifying a date/time edit mask for the field in the format buffer is no longer allowed.
HF set	HF not set	The behaviour of cross-platform calls changes.
HF not set	HF set	
LA and LB not set	LA or LB set	The behaviour of calls accessing the field with variable length changes.
LA set	LA not set, LB set	
LB set	LB not set, LA set	Only allowed if there is no LOB file defined for the file or if the field is a descriptor of the parent of a derived descriptor. The behaviour of calls accessing the field with variable length changes.
NB not set	NB set	
NC and NN set	FI, NC, NU and NN not set	After this change, the field is no longer mandatory in the format buffer for N1/N2 commands; if not specified, the field gets the Adabas null value.
NC and NN set	NC set, NN not set	After this change, the field is no longer mandatory in the format buffer for N1/N2 commands; if not specified, the field gets the SQL null value.
NU set	NC set	Empty values are converted to NULL values. Note that NC set -> NU not set because NU and NC are mutually exclusive.
NV set	NV not set	The behaviour of cross-platform calls changes.
NV not set	NV set	

Old Field Options	New Field Options	Comments
SY not set	SY set	The behaviour of A1, N1 and N2 commands changes. The field format must be compatible with the SY option. Note that no check is made to ensure that the existing values are reasonable.
SY set	SY not set	The behaviour of A1, N1 and N2 commands changes.
TR not set	TR set	
TZ not set DT set	TZ set, DT unchanged	Values in the database will be converted from UTC to local time when you specify a date/time edit mask.
TZ set	TZ not set	Values in the database are no longer converted from UTC to local time when you specify a date/time edit mask.

field_specification

The field specification list is entered in the same way as the FDT input in ADAFDU:

```
level-number, name [,length] [,format] [(,option)...
```

The first field to be added must be a level-one field.

FDT

This parameter displays the FDT of the file to which the fields are to be added.

DBID

```
DBID = number
```

This parameter selects the database to be used.



Note: Utility functions which require or allow the nucleus to be shut down need logical assignments for the data sets.

Examples

```
adadbm: dbid=76
%ADADBAM-I-DBOFF, database 76 accessed offline
```

```
adadbm: dbid=76
%ADADBAM-I-DBON, database 76 accessed online
```

DEALLOCATE

```
DEALLOCATE = keyword, FILE = number [,RABN = number],
            SIZE = numberB
```

DEALLOCATE = AC, DS, NI or UI

Depending on the keyword specified (AC, DS, NI or UI), this function releases a given amount of space from the Address Converter (AC), Data Storage (DS), Normal Index (NI) or Upper Index (UI).

If too much space is allocated to an extent, either automatically or manually, the DBA can release this space and return it to the Free Space Table (FST).

Deallocation is done for only one extent at a time. To release space from multiple extents, DEALLOCATE has to be called several times.

FILE = number

This parameter specifies the file.

RABN = number

This parameter specifies the first RABN to be deallocated. If this parameter is omitted, deallocation starts at the end of the last extent.

SIZE = numberB

This parameter specifies the size of the area to be deallocated, in blocks.

Example

```
adadbm: deallocate=ni, file=11, size=110b
SIZE=110B
      ^
%ADADBM-E-VALUP, value has to be less-equal 100
%ADADBM-I-ABORTED,      14-NOV-2002 14:44:01, elapsed time: 00:00:00
```

```
adadbm: deallocate=ni, file=11, size=100b
%ADADBM-I-DEALLOC, 100 NI blocks deallocated (611 - 710)
```

```
adadbm: deallocate=ni, file=11, size=10b
%ADADBM-I-DEALLOC, 10 NI blocks deallocated (323 - 332)
```

DEFINE_REFINT

```
DEFINE_REFINT = number constraint_specification
```

This function adds a referential constraint to the file 'number', which contains a foreign key. The syntax for the constraint is the same as that used in the FDT file for ADAFDU and is described in *Administration, FDT Record Structure, Referential Constraints*. The constraint is also included in the FDT of the primary file, therefore, the constraint name must not already be defined in the primary file.

Adding a referential constraint is not allowed if the file specified as the primary file is defined with PGM_REFRESH=YES.

If there are violations of the referential integrity, adding of the constraint will fail - no updates are performed on the data of the file in order to establish referential integrity.

DELCP

```
DELCP = { * | ([absolute-date] [, [absolute-date]]) }
```

This function deletes checkpoint records from the checkpoint file.

If an asterisk '*' is entered, all checkpoint records are deleted.

Examples

```
adadbm: delcp=13-NOV-2006:15:09:48
%ADADBM-I-DELCP, 1 record deleted from CHECKPOINT file

adadbm: delcp=(13-NOV-2006:15:09:48,)
%ADADBM-I-DELCP, 81 records deleted from CHECKPOINT file

adadbm: delcp=(,14-NOV-2006:14:37:24)
%ADADBM-I-DELCP, 41 records deleted from CHECKPOINT file

adadbm: delcp=(14-NOV-2006:14:37:25,14-NOV-1996:14:38:15)
%ADADBM-I-DELCP, 42 records deleted from CHECKPOINT file

adadbm: delcp=*
%ADADBM-I-DELCP, 20 records deleted from CHECKPOINT file
```


DELETE

```
DELETE = (number [-number][,number[-number]]...)
```

The DELETE function deletes one or more files or ranges of files from the database and returns all space which was allocated for this file to the Free Space Table (FST). LOB files specified are ignored, but the LOB files assigned to all base files specified are deleted too. There must not be a referential constraint between a file that is to be deleted and another file, which is not specified.

ADADBМ does not request confirmation of the files to be deleted, i.e. care should be taken when entering the file numbers.

Example

```
adadbm: delete=(4-11,14)
%ADADBМ-I-DELETED, file 11 deleted
%ADADBМ-I-DELETED, file 14 deleted
```

DISPLAY

```
DISPLAY = UCB
```

The DISPLAY function displays the utility communication block. This function can also be executed during a pending AUTORESTART.

Example:

```
adadbm: display=ucb
```

Date/Time	Entry Id	Utility	Mode	Files
-----	-----	-----	-----	-----
14-NOV-2006 14:38:40	233	adaopr	UTO	11
14-NOV-2006 14:38:42	234	adabck	ACC	*

The display shows the following items:

- DATE/TIME shows the date and time on which the given files were locked.
- ENTRY ID shows the allocated identification of the entry.
- UTILITY shows the name of the utility.
- MODE shows the mode in which the files are being accessed.
- FILES shows the file numbers of the files that are locked.

DROP_FIELDS

```
DROP_FIELDS = number {field_name|FDT}... END_OF_FIELDS
```

The DROP_FIELDS function drops one or more fields from the file defined by 'number' - the specified fields are marked as no longer existing and they cannot be accessed. Specifying a LOB file is not permitted. The function is completed by entering END_OF_FIELDS.

If the field definitions are terminated with the END_OF_FIELDS parameter, this parameter must be specified in upper case when the LOWER_CASE_FIELD_NAMES parameter is used. In addition, the FDT parameter must also be specified in upper case when the LOWER_CASE_FIELD_NAMES parameter is used.

If you specify a group or a periodic group, all of the fields that belong to the group or periodic group are dropped. You must not specify a field that is a descriptor or from which a descriptor is derived - if you want to drop such a field, you must first release all corresponding descriptors with ADAINV.

Once the DROP_FIELDS function has been executed, you can redefine the names of the dropped fields, for example using ADADBAM's ADD_FIELDS function.



Notes:

1. The DROP_FIELDS function does not physically remove the fields. You should not drop and then add fields repeatedly, since this can cause the data records or the FDT of the file in question to overflow.
2. ADAMUP is not able to load data into a file that contains the same visible fields but which contains different dropped fields.

FDT

This parameter displays the FDT of the file from which the fields are to be dropped.

DROP_LOBFILE

```
DROP_LOBFILE = number ↵
```

The number must specify the file number of a base file with an empty assigned LOB file to be deleted.

DROP_LOBFILE is not allowed if the assigned LOB file is not empty.

DROP_REFINT

```
DROP_REFINT = number, NAME {=|:} constraint_name
```

The function removes a referential constraint from the file specified by 'number', which contains the foreign key. The constraint is also removed from the FDT of the primary file.

EXTEND_CONTAINER

```
EXTEND_CONTAINER = keyword, SIZE = number [B|M]
```

The EXTEND_CONTAINER function extends the last Associator, Data Storage or WORK container file defined for the database in accordance with the keyword used. The keyword can take the values ASSO, DATA or WORK.



Note: The WORK container can only be extended in the offline mode.

SIZE = number [B|M]

This parameter specifies the size of the expansion area in blocks (B) or megabytes (M). By default, the size is in megabytes.

[NO]LOWER_CASE_FIELD_NAMES

```
[NO]LOWER_CASE_FIELD_NAMES
```

If LOWER_CASE_FIELD_NAMES is specified, Adabas field names are not converted to upper case. If NOLOWER_CASE_FIELD_NAMES is specified, Adabas field names are converted to upper case. The default is NOLOWER_CASE_FIELD_NAMES.

This parameter must be specified before the ADD_FIELDS, CHANGE_FIELDS or DEFINE_REFINT parameters.

NEW_DBID

```
NEW_DBID = number
```

This function is used to change the identifier of the database in use. The new identifier may not already be in use by another active database.

Example

```
adadbm: new_dbid=77
%ADADBAM-I-FUNC, function NEW_DBID executed
```

NEWWORK

```
NEWWORK [,BLOCKSIZE = number[K] ], SIZE = number [B|M]
```

This function removes the existing WORK1 container file and replaces it with a new WORK1 container file. The new WORK1 container file is allocated and then formatted, if required.

Before a new WORK can be created, the nucleus and all utilities using the database must have been successfully terminated. Since this function requires the current WORK, it must not be deleted before NEWWORK has been executed. TEMP1 must point to the new work file when this function is used.



Note: The new WORK can be directed to a disk section or to a file system. If TEMP1 points to the same disk section as WORK1, then ADADBAM tries to extend/reduce the existing WORK file. In each case the name of the new WORK container file is WORK1. If the function completes successfully, the old WORK1 gets deleted.

BLOCKSIZE = number[K]

This parameter specifies the block size in bytes (or in kilobytes, if "K" follows the number) of the new container file.

Adabas rounds up the value you specify to the next multiple of 1024.

The minimum block size allowed is 3072 and the maximum block size allowed is 32768.

In addition to these minimum and maximum values, the following size restrictions apply in general to the block sizes for ASSO and WORK:

MAX (ASSOBLS) < WORKBLS

where MAX(ASSOBLS) represents the largest ASSO block size and WORKBLS represents the WORK block size.

The default value for BLOCKSIZE is the block size of the old WORK file.

SIZE = number [B|M]

This parameter specifies the number of blocks or megabytes to be allocated for the new WORK file. By default, the size is in megabytes. The minimum value is 200 blocks or the equivalent value in megabytes.

PGM_REFRESH

```
PGM_REFRESH = keyword, FILE = number
```

This function is used to disable or enable refreshing an Adabas file inside an application program with an E1 command (ISN=0, CID=BLANK). Specifying a LOB file is not permitted. The keyword can take the values YES or NO. It is not allowed to set PGM_REFRESH=YES for files that are primary files of referential constraints.

FILE = number

This parameter specifies the file for which refreshing is to be enabled/disabled.

RECOVER

```
RECOVER
```

This function returns lost space within the Associator and Data Storage to the Free Space Table (FST).

Space can be lost by a non-successful termination of an Adabas utility.

Example

```
adadbm: recover
%ADADBAM-I-FUNC, function RECOVER executed
```

REDUCE_CONTAINER

```
REDUCE_CONTAINER = keyword, SIZE = number B
```

The REDUCE_CONTAINER function deallocates free space at the end of the Associator or Data Storage container defined for the database in accordance with the keyword used. The keyword can take the values ASSO or DATA.

The requested number of blocks must not be in use at the end of the container specified. If the complete space of one or more container extents is to be released, the container extents are removed. Note that the message informing you that a container extent is removed is not displayed by ADADBAM if ADADBAM is executed online - instead, it is included in the nucleus log.

If less blocks than requested are free at the end of the container, all free space at the end of the container is deallocated, and the following warning is displayed:

```
%ADADBM-W-PREDCONT, not all requested blocks removed
```

SIZE = number B

This parameter specifies the size by which the container is to be reduced, in blocks.

REFRESH

```
REFRESH = (number [-number][,number[-number]]...)
```

This function resets the files specified by 'number' to the state of zero records loaded. Only the first extents for Normal Index, Address Converter and Data Storage are kept. The remaining extents are returned to the Free Space Table (FST). The Upper Index is rebuilt and the unused Upper Index extents are then returned to the Free Space Table. LOB files specified are ignored, but the LOB files assigned to all base files specified are refreshed too. The primary file of a referential integrity constraint may be refreshed only if the foreign file of the referential constraint is also refreshed.

ADADBM does not request confirmation of the files to be refreshed, i.e. care should be taken when entering the file numbers.

This function is useful for clearing a test file in a test environment. This method is faster than deleting and reloading the file.

Files using the ADAM feature cannot be refreshed.

If the REMOVE_DROP function has been specified, dropped fields are removed from the FDT.

Example

```
adadbm: refresh=13  
%ADADBM-I-REFRESH, file 13 refreshed
```

REMOVE_CONTAINER

```
REMOVE_CONTAINER = keyword
```

This function removes the last database container file from an existing Associator or Data Storage data set in accordance with the keyword used. The keyword can take the values ASSO or DATA.

The container file to be removed must not be in use when this function is executed, i.e. all of the blocks in the file must be free.

The container file will be deleted from the file system or from the raw disk section.

Before a container file can be removed, the nucleus and all of the utilities using the database must have terminated successfully.

Example

```
adadbm: remove_container=data
%ADADBM-I-DMCONREM, container DATA2 removed
```

REMOVE_DROP

```
[NO]REMOVE_DROP
```

If you specify REMOVE_DROP, subsequent REFRESH functions will remove dropped fields from the FDT.

If you specify NOREMOVE_DROP, subsequent REFRESH functions will not remove dropped fields from the FDT.

The default is NOREMOVE_DROP.

Example

```
adadbm: remove_drop
adadbm: refresh=2
%ADADBM-I-REFRESH, file 2 refreshed
adadbm: refresh=3
%ADADBM-I-REFRESH, file 3 refreshed
adadbm: noremove_drop
adadbm: refresh=4
%ADADBM-I-REFRESH, file 4 refreshed
```

File 2 has been refreshed and dropped fields have been removed from the FDT. File 3 has been refreshed and dropped fields have been removed from the FDT. File 4 has been refreshed and dropped fields have not been removed from the FDT.

REMOVE_REPLICATION

```
REMOVE_REPLICATION
```

This function stops all replication processing and deletes all replication system files.



Note: This function is only relevant for customers who are using the Adabas Event Replicator with Adabas - Adabas replication.

RENAME

```
RENAME = number, NAME {=:} string
```

This function changes the name of a file or a database. `number` is the number of the file whose name is to be changed. If `number` is 0, the name of the database is changed.

NAME {=:} string

`string` is the new name of the specified file or database. If you specify an equals sign, the value given for `string` will be converted to upper case; if you specify a colon, no upper-case conversion is performed.

Example

```
adadbm: rename=11, name=employee-file  
%ADADBAM-I-FUNC, function RENAME executed
```

RENUMBER

```
RENUMBER = (number, number)
```

This function changes the file number of a loaded Adabas file. If, however, the file's new number already belongs to a loaded file, the numbers of these files are exchanged.

The first `number` is the file number currently assigned to the file. The second `number` is the new file number to be assigned to the file.

Example:

```
adadbm: renumber=(12,14)  
%ADADBAM-I-RENUM, File 12 renumbered to 14  
%ADADBAM-I-RENUM, File 14 renumbered to 12
```

REPLICATION_FILES

```
REPLICATION_FILES = (file1, file2, file3, file4)
```

This function performs all of the necessary initialization steps for the Adabas - Adabas replication and creates the replication system files.

file1

The metadata file.

file2

The replication transaction file.

file3

The replication command file.

file4

LOB file for the replication command file.



Note: This function is only relevant for customers who are using the Adabas Event Replicator with Adabas - Adabas replication.

RESET

```
RESET = UCB, IDENT = { (number [,number]...) | * }
```

UCB

This function removes one or more entries from the utility communication block (UCB). This option can also be used during a pending AUTORESTART.

The UCB is used to control access to certain resources (the whole database, one or more files, etc.) within a database. It saves information about the Adabas utilities processing the database and the resources attached to them.

An entry is made in the UCB each time a utility is granted access to a resource. This entry contains information about the utility and the resources it locks. The utility automatically removes the entry when the resource is no longer required. Please refer to the DISPLAY=UCB function of this utility for information about how to display the contents of the UCB.

However, certain special conditions (e.g. an aborted ADAMUP) can cause entries to remain in the UCB and keep allocated resources locked. The RESET function releases these resources by removing one or more entries from the UCB.

Resetting a UCB entry also removes the associated entry from the user queue and returns lost blocks to the free space table if the nucleus is active. Otherwise, the resource can be returned to the free space table by using the RECOVER function.

IDENT = { (number [,number]...) | * }

This parameter specifies the unique ID of the entry to be removed. '*' removes all entries.

If the RESET UCB function is used offline, only '*' may be specified.

Example

```
adadbm: reset=ucb, ident=233
%ADADBAM-I-RESUCB1, 1 entry deleted from UCB
```

```
adadbm: reset=ucb, ident=(235,234)
%ADADBAM-I-RESUCB, 2 entries deleted from UCB
```

```
adadbm: reset=ucb, ident=*
%ADADBAM-I-RESUCB1, 1 entry deleted from UCB
```

RESET_REPLICATION_TARGET

```
RESET_REPLICATION_TARGET = number
```

This function resets the replication target flag of Adabas files, after which they are handled as normal files again. If you specify 0, the replication target flag of all replication target files is reset; if you specify a file number, the replication target flag of the file with this file number is reset.



Notes:

1. This function is only relevant for customers who are using the Adabas Event Replicator with Adabas - Adabas replication.
2. After performing this function, a replication to this replication target is no longer possible - if the replication to this replication target is still active, a new update transaction on the replication source will set the replication to status Error. If you want to replicate data to this replication target again, a new initial state processing is required.

REUSE

```
REUSE = (keyword [,keyword]), FILE = number
```

The REUSE function controls the reuse of Data Storage space or ISNs by Adabas.

The File Control Block (FCB) for the specified file is modified to indicate the type of allocation technique to be used when adding new records or moving updated records.

The valid keywords are [NO]DS and [NO]ISN.

If the DS keyword is specified, Adabas scans the Data Storage Space Table (DSST) in order to locate a block with sufficient space. In this case, the first block found with sufficient space is used.

If the NODS keyword is specified, then all newly-added records, together with records that have to be moved to another block (as a result of record expansion caused by updating), are placed in

the last used block in the Data Storage extent allocated to the file. If there is not sufficient space in this block, the next block is used.

DS and NODS are mutually exclusive. The default is REUSE = DS.

If the ISN keyword is specified, Adabas may reuse the ISN of a deleted record.

If the NOISN keyword is specified, Adabas does not reuse the ISN of a deleted record for a new record. Each new record will be assigned the next-highest unused ISN.

ISN and NOISN are mutually exclusive. The default is REUSE = NOISN.

FILE = number

This parameter specifies the file.

Example

```
adadbm: reuse=nods, file=11
%ADADBM-I-FUNC, function REUSE executed
```

```
adadbm: reuse=(ds,isn), file=12
%ADADBM-I-FUNC, function REUSE executed
```

SYFMAX

```
SYFMAX = number, FILE = number
```

This parameter specifies the maximum number of values generated for a system generated multiple-value field in the file specified. There is no explicit maximum value, but you should bear in mind, that you can get a record overflow if the value is defined too high; the compressed data record should also fit into one DATA block if SYFMAX values are defined for system generated multiple-value fields. If the SYFMAX value is decreased and a record contains more values for system generated fields than the new value of SYFMAX, the excess values are removed during the next update operation for this record.

FILE = number

This parameter specifies the file.

Restart Considerations

ADADBМ has no restart capability. At the end of each function, however, the system reports whether execution was successfully completed or not. If it is not successfully completed, the function has to be re-started.

9 ADADCU (Decompression Of Data)

▪ Functional Overview	102
▪ Procedure Flow	103
▪ Checkpoints	104
▪ Control Parameters	105
▪ Input and Output Data	113
▪ Restart Considerations	114

This chapter describes the utility "ADADCU".

Functional Overview

The decompression utility ADADCU decompresses records produced by the ADACMP, ADAMUP and ADAULD utilities.

The output of the decompression utility ADADCU can be used as input for a program using standard operating system file access methods.

It can also be used as input for the compression utility ADACMP once any required changes have been made to the data structure or once the data definitions of the file have been changed. A warning message is issued if the decompressed output file (DCUOUT file) created by the utility is empty.

ADADCU also decompresses files produced with the SINGLE option of the utilities ADAULD and ADACMP, but no parameter is required since this can be determined by the utility.

With ADADCU, the following functions are available:

- Complete records can be decompressed to the formats and lengths described in the FDT. A one-byte count field precedes each multiple-value field or periodic group in the output record.
- LOB field values can also be stored in separate files; the generated file names are put into the decompressed records.
- Several fields can be decompressed.

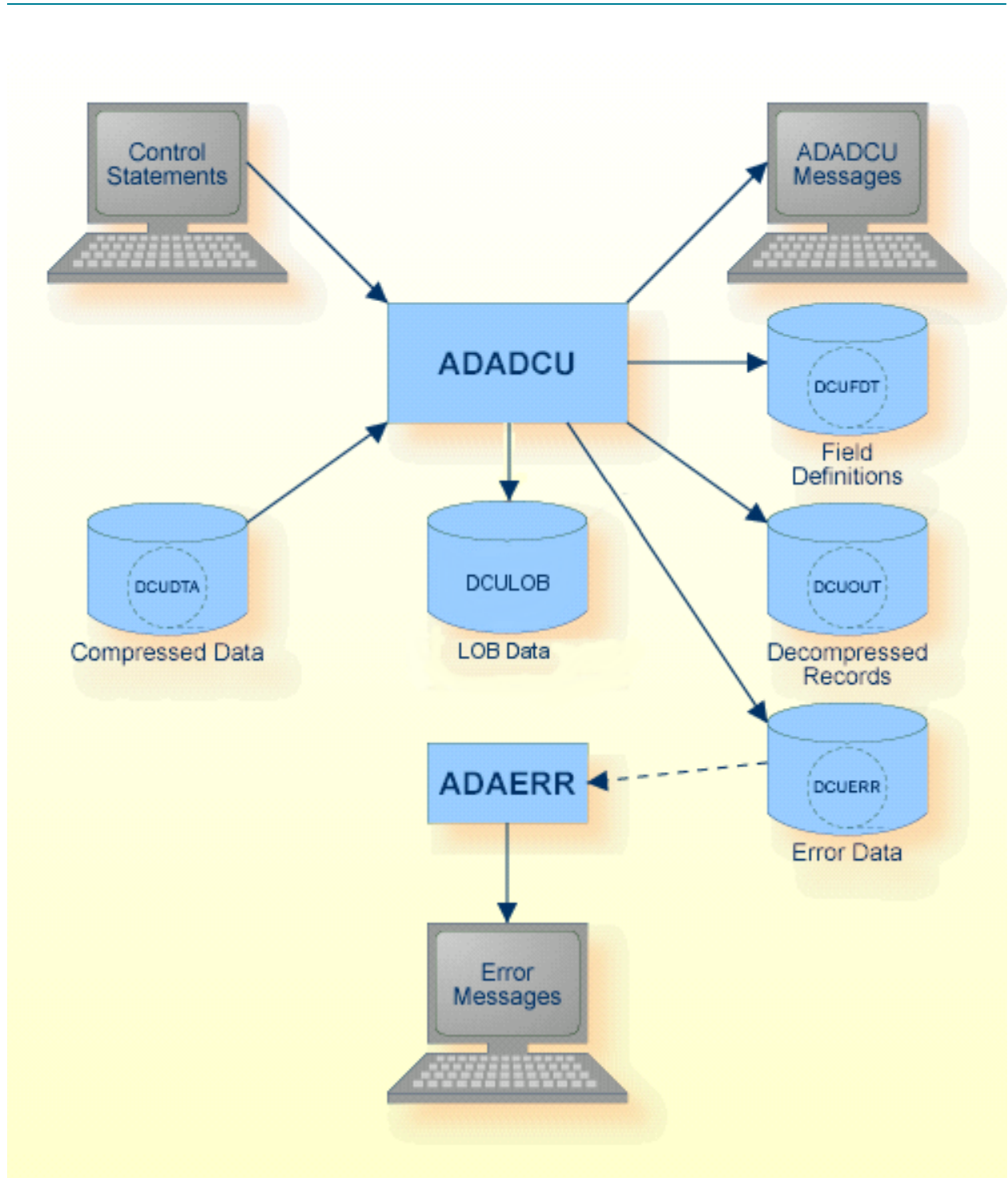
If several fields are decompressed, the fields can be re-arranged within a record, i.e. the record structure may be changed as follows:

- Field lengths can be changed;
- Field formats can be changed;
- Space can be allocated for subsequent addition of new fields using the literal element or blank element.

If the utility writes records to the error file, it will exit with a non-zero status.

This utility is a single-function utility.

Procedure Flow



DCULOB is a directory where LOB values are stored in separate files. The sequential files DCUDTA and DCUERR can have multiple extents. For detailed information about sequential files with multiple extents, see Administration, Using Utilities.

Data Set	Environment Variable/ Logical Name	Storage Medium	Additional Information
Compressed data records	DCUDTA	Disk, Tape (* see note)	Output of ADACMP or ADAULD
Rejected data	DCUERR	Disk, Tape (* see note)	Output of ADADCU
Output data FDT	DCUFDT	Disk, Tape (* see note)	Output of ADADCU Utilities Manual
Decompressed records	DCUOUT	Disk, Tape (* see note)	Utilities Manual
LOB data	DCULOB	Disk	Utilities Manual
Control statements	stdin/ SYS\$INPUT		Utilities Manual
ADADCU messages	stdout/ SYS\$OUTPUT		Messages and Codes



Note: (*) A named pipe can be used for this sequential file (not on OpenVMS, see *Administration, Using Utilities* for details).

Checkpoints

The utility writes no checkpoints.

Control Parameters

The following control parameters are available:

```

D   [NO]DCUFDT
D   [NO]DST
    FDT
    FIELDS {field_specification | FDT},...{END_OF_FIELDS | .}
D   [NO]LOWER_CASE_FIELD_NAMES
D   MAX_DECOMPRESSED_SIZE = number [K|M]
D   MUPE_C_L = {1|2|4}
    MUPE_OCCURRENCES
D   [NO]NULL_VALUE
D   NUMREC = number
D   RECORD_STRUCTURE = keyword
    SKIPREC = number
D   TARGET_ARCHITECTURE = (keyword[,keyword[,keyword]])
D   [NO]TRUNCATION
    TZ {=|:} [timezone]
D   [NO]USERISN
    WCHARSET = char_set

```

[NO]DCUFDT**[NO]DCUFDT**

If this option is set to DCUFDT, the FDT information of the decompressed records is written to the sequential file DCUFDT. The default is NODCUFDT.

If you have used the FIELDS parameter (see below), the fields are written to the sequential file DCUFDT in the order specified in FIELDS. Thus, the fields in DCUFDT might be in a different order to those in the original FDT.

[NO]DST**[NO]DST**

The parameter DST is required if a daylight saving time indicator is to be provided for date/time fields with the option TZ. The daylight saving time indicator will be appended behind the date/time value as a 2-byte integer value (format F) containing the number of seconds to be added to the standard time to get the actual time (usually 0 or 3600).

This parameter is required if there are records containing date/time values with the option TZ in the hour before the time is switched back to standard time, otherwise these values are written to the error file.

The default is NODST.

**Notes:**

1. The DST parameter is ignored if the FIELDS parameter is specified. In this case, you must specify a D element for fields with the daylight saving time indicator.
2. The DST parameter is not compatible with the RECORD_STRUCTURE = NEWLINE_SEPARATOR parameter because the daylight saving indicator in format F contains non-printable characters.

FDT**FDT**

This parameter displays the FDT of the file containing the compressed records.

FIELDS

```
FIELDS {field_specification | FDT},...{ END_OF_FIELDS | . }
```

This parameter is used to specify a subset of fields given in the FDT and their format and length. This means that the decompressed records created do not have to contain all of the fields given in the FDT, or that fields can be decompressed with a different format or length. The syntax and semantics are the same as for the format buffer, with the exception that you can also specify an R-element (for LOB references) if the decompressed record contains the name of a file containing the LOB value instead of the LOB value itself. See *Administration, Loading And Unloading Data, Uncompressed Data Format* for further information.

While entering the specification list, the FDT function can be used to display the FDT of the file to be decompressed. The specification list can be terminated or interrupted by entering END_OF_FIELDS or `.`. The `.` option is an implicit END_OF_FIELDS and is compatible with the format buffer syntax. FIELDS or END_OF_FIELDS must always be entered on a line by itself, whereas the `.` may be entered on a line by itself or at the end of the format buffer elements. Processing may be continued after setting any option or parameter by entering FIELDS.

If the field definitions are terminated with the END_OF_FIELDS parameter, this parameter must be specified in upper case when the LOWER_CASE_FIELD_NAMES parameter is used. In addition, the FDT parameter must also be specified in upper case when the LOWER_CASE_FIELD_NAMES parameter is used.

Example

```
adadcu: fields
adadcu: ; This is a comment line
adadcu: AA,AB,6,A,AC,P ; - inline comment -
adadcu: AD,AF,CBC,CB1-N . ; implicit END_OF_FIELDS
```

Field AA is output with default length and format, field AB with 6 byte alphanumeric and field AC with default length packed. Fields AD and AF are output in default length and format, followed by the one-byte binary multiple field count of field CB and all its occurrences.

[NO]LOWER_CASE_FIELD_NAMES

```
[NO]LOWER_CASE_FIELD_NAMES
```

If LOWER_CASE_FIELD_NAMES is specified, Adabas field names are not converted to upper case. If NOLOWER_CASE_FIELD_NAMES is specified, Adabas field names are converted to upper case. The default is NOLOWER_CASE_FIELD_NAMES.

This parameter must be specified before the FIELDS parameter.

MAX_DECOMPRESSED_SIZE

```
MAX_DECOMPRESSED_SIZE = number [K|M]
```

This parameter specifies the maximum size of a decompressed record in bytes, kilobytes or megabytes, depending on the specification of "K" or "M" after the number. This parameter is intended to prevent very large decompressed record files from being created unintentionally (if you didn't consider that a file contained LOB data).

The default is 65536. This is also the minimum value.



Note: The exact definition of this parameter is the size of the I/O buffer required for the largest decompressed record. Only multiples of 256 bytes are used for the I/O buffers, which means that you must specify a value greater than or equal to the largest decompressed record (including the preceding length field) rounded up to the next multiple of 256.

MUPE_C_L

```
MUPE_C_L = {1|2|4}
```

If the data contain multiple-value fields or periodic groups, they are preceded by a binary count field with the length of MUPE_C_L bytes in the decompressed data.

The default is 1.

MUPE_OCCURRENCES

```
MUPE_OCCURRENCES
```

This parameter is used to print a list of all multiple fields and periodic groups together with their maximum occurrence. Such information is important because the decompressed data can become very large; if the range specified is too large, it is even possible to exceed the limit for the size of a decompressed record.

Example

The FDT of the file containing the compressed records is as follows:

```
1,AA,4,A,NU  
1,PE,PE  
2,PA,2,A,NU  
2,PB,2,A,NU,MU  
1,MM,2,U,NU,MU  
1,X1,4,B
```

MUPE_OCCURRENCES might produce something of the form:

Name	Max occurrence
PE	4
PB	8
MM	12

```
%ADADCU-I-DCUREC, Number of decompressed records: 5023
%ADADCU-I-DCUIR, Number of incorrect records: 0
```

The file can then be decompressed as follows:

```
adadcu fields "AA,PA1-4,PB1-4(1-8),MM1-12,P,X1" ↵
```



Note: A record is considered to be incorrect if it has too many occurrences of a periodic group containing an MU field, and thus causes an internal overflow. It is not possible to decompress this record including the periodic group.

[NO]NULL_VALUE

```
[NO]NULL_VALUE
```

This parameter can be used to decompress records according to the standard FDT if the record contains NC option fields and their status values (S-elements). It is required if one or more fields have the null value, otherwise these records are put in the error file.

Example

If the FDT entry for field AA is: 1, AA, 2, A, NC, the effect of NULL_VALUE is as follows:

- NULL_VALUE: 1st output record (in hex) 00004141 (AA has a value), 2nd output record (in hex) FFFF2020 (AA has the null value).
- NONULL_VALUE: 1st output record (in hex) 4141 (AA has a value), 2nd output record (in hex) AA is null, therefore the record will be put into the error file.

The default is NONULL_VALUE.

NUMREC

```
NUMREC = number
```

This parameter specifies the number of records to be read from the input file and decompressed. If NUMREC is not specified and SKIPREC is also not specified, all records are processed.

Example

```
adadcu: numrec = 100
```

100 records are read and decompressed.

RECORD_STRUCTURE

```
RECORD_STRUCTURE = keyword
```

This parameter specifies the type of record separation used in the output file with the logical name DCUOUT. The following keywords can be used:

Keyword	Meaning
ELENGTH_PREFIX	The records in the DCUOUT file are separated by a two-byte exclusive length field. There is no separator character and the use of this format is not subject to any restrictions.
E4LENGTH_PREFIX	The records in the decompressed data file are separated by a 4-byte exclusive length field.
ILENGTH_PREFIX	The records in the DCUOUT file are separated by a two-byte inclusive length field. There is no separator character and the use of this format is not subject to any restrictions.
I4LENGTH_PREFIX	The records in the decompressed data file are separated by a 4-byte inclusive length field.
NEWLINE_SEPARATOR	The records in the DCUOUT file are separated by a new-line character. If the DCUOUT file is to be used as input for ADACMP, this keyword can only be specified if the field values of the output do not contain the new-line character (i.e. if there are only unpacked, alphanumeric and Unicode fields, and if the alphanumeric and Unicode fields only contain printable characters). This keyword and the USERISN parameter are mutually exclusive.
RDW	The records in the DCUOUT file are formatted such that they can be transferred to an IBM host using the FTP <i>site rdw</i> option.
RDW_HEADER	Like RDW, for decompressed records that can be compressed on a mainframe with HEADER=YES.
VARIABLE_BLOCKED	The records are stored as blocks. Each record begins with an inclusive four-byte length field.

The default is ELENGTH_PREFIX.

SKIPREC

```
SKIPREC = number
```

This parameter specifies the number of records to be skipped before decompression is started.

TARGET_ARCHITECTURE

```
TARGET_ARCHITECTURE = (keyword[,keyword[,keyword]])
```

This parameter specifies the format (character set, floating-point format and byte order) of the output data records. The following keywords can be used:

Keyword Group	Valid Keywords
Character set	ASCII EBCDIC
Floating-point format	IBM_370_FLOATING IEEE_FLOATING VAX_FLOATING
Byte order	HIGH_ORDER_BYTE_FIRST LOW_ORDER_BYTE_FIRST

If no keyword of a keyword group is specified, the default for this keyword group is the keyword that corresponds to the architecture of the machine on which ADADCU is running.



Note: The FDT is always output in ASCII format.

Example

If the output records are to be decompressed into IBM format, the user must specify the following:

```
TARGET_ARCHITECTURE = (EBCDIC, IBM_370_FLOATING, HIGH_ORDER_BYTE_FIRST)
```

[NO]TRUNCATION

[NO]TRUNCATION

This option enables or disables the truncation of alphanumeric field values.

NOTRUNCATION is the default. In this case, all the records with truncated alphanumeric field values are written to the error file.

Numeric values may not be truncated, and the value must fit into the standard or specified length. If truncated numeric values occur, the records concerned are written to the error file.

TZ

TZ {=|:} [timezone]

The specified time zone must be a valid time zone name that is contained in the time zone database known as the Olson database (<http://www.twinsun.com/tz/tz-link.htm>). If a time zone has been specified, this time zone is used for time zone conversions of date/time fields with the option TZ.

The default is UTC, which is used internally to store date/time fields with option TZ; no conversion is required.

If you specify an empty value, no checks are made to ensure that date/time fields are correct.



Note: The time zone names are file names. Depending on the platform, these file names may or may not be case sensitive. Also, the time zone names, depending on the platform, may or may not be case sensitive.

Examples:

```
tz:Europe/Berlin
```

This is correct on all platforms.

```
TZ=Europe/Berlin
```

With this specification, TZ is converted to upper case EUROPE/BERLIN. This is correct on Windows, because file names are not case sensitive on Windows, but it is not correct on Unix, because Unix file names are case sensitive.

[NO]USERISN

```
[NO]USERISN
```

This parameter indicates whether the ISN is to be output together with each decompressed record or not. The user can specify whether the ISN currently assigned to the record is to be output with the decompressed data or whether it is to be omitted. If the user intends to reload the file with the same ISNs, the USERISN option must be set.

This parameter cannot be specified if RECORD_STRUCTURE=NEWLINE_SEPARATOR is specified.

If this parameter is omitted, the ISN is not output with each record.

NOUSERISN is the default.

Example

```
adadcu: userisn
```

The ISN is output with each record.

WCHARSET

```
WCHARSET = char_set
```

This parameter specifies the default encoding used in the decompressed file based on the encoding names listed at <http://www.iana.org/assignments/character-sets> - most of the character sets listed there are supported by ICU, which is used by Adabas for internationalization support.

Input and Output Data

The input for ADADCU must be a file containing compressed records such as those output by the unload utility ADAULD or by the compression utility ADACMP.

ADADCU decompresses each input record in accordance with the FIELDS specifications and writes the resulting record to the file with the logical name DCUOUT. The records are written in variable-length format. By default, the records are separated by a two-byte exclusive length field (see the parameter RECORD_STRUCTURE in this section for more detailed information).

If USERISN is specified, the data record is preceded by its ISN in the form of a four-byte binary number.

ADADCU Output

The sequential file DCUFDT (field definition information of the decompressed records) can be used as input for the file definition utility ADAFDU or for the compression utility ADACMP.

Rejected Data Records

Any records rejected by ADADCU are written to the ADADCU error file. The contents of this error file should be displayed using the ADAERR utility. Do not print the error file using the standard operating system print utilities since the records contain unprintable characters.

See the ADAERR utility for further information.

Restart Considerations

ADADCU does not have a restart capability. An interrupted ADADCU run must be re-executed from the beginning.

ADADCU does not update the database, therefore, no considerations regarding the status of the database need to be made before re-executing an interrupted ADADCU execution.

10 ADADEV (Disk Space Management)

▪ Functional Overview	116
▪ Procedure Flow	117
▪ Checkpoints	118
▪ Control Parameters	118

This chapter describes the utility "ADADEV".



Note: This utility only applies to UNIX platforms.

Functional Overview

The ADADEV utility provides several functions for managing disk space to be used by Adabas via the raw disk I/O interface.

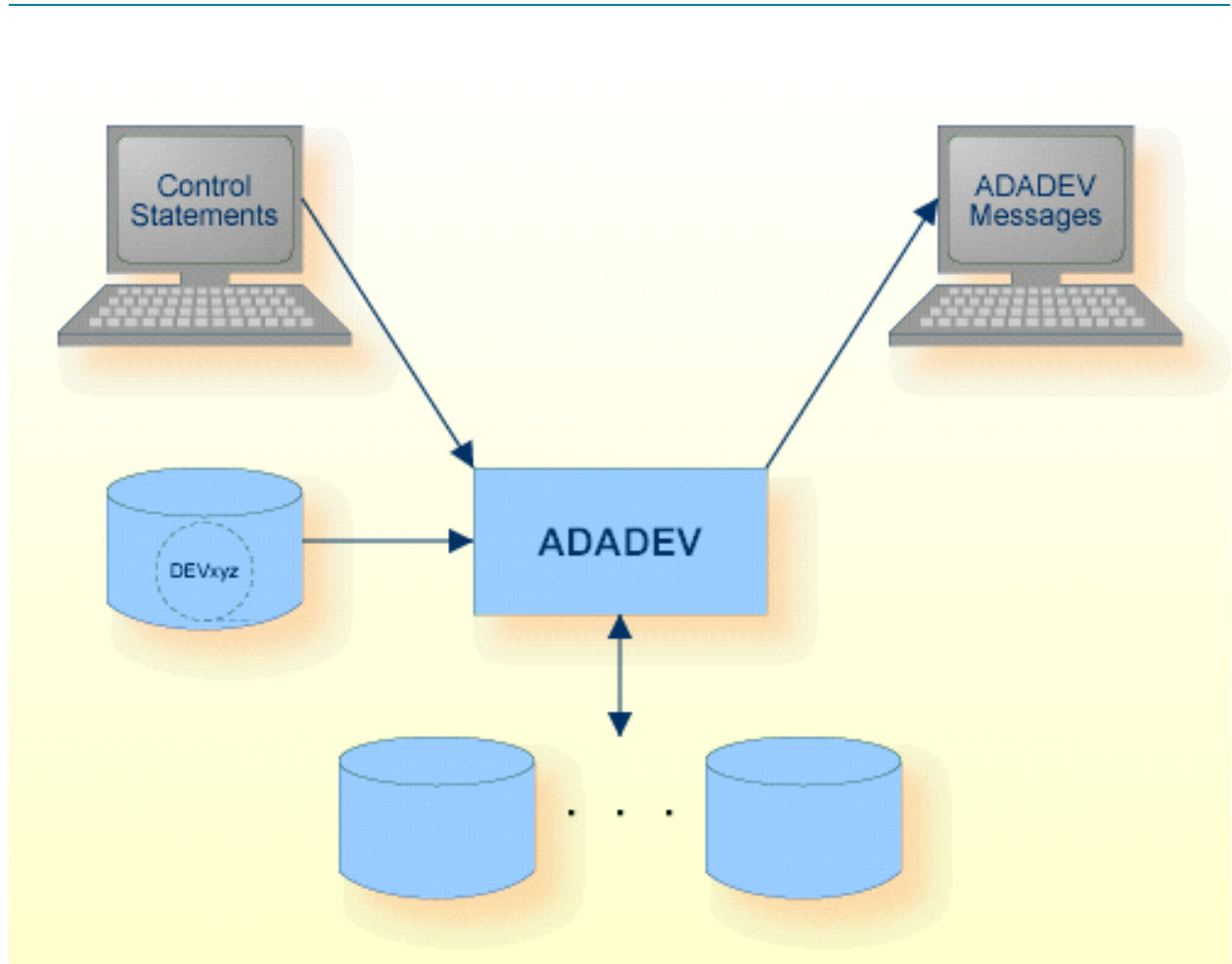
ADADEV requires READ/WRITE access to the specified disk-section device file. See *Installation, Installing Adabas* for information about raw disk-section usage in your system.

Each disk section used by Adabas must be initialized once with ADADEV. Preallocation for Adabas container files or for Adabas sequential files is not necessary, but can be useful sometimes. Disk space is allocated automatically when creating container extents with ADAFRM or when creating Adabas sequential files with an Adabas utility. If space has not been preallocated, a best-fit algorithm is used for container extents. For Adabas sequential files, one half of the largest available free space is allocated if it is larger than 1 MB. If the allocated space is exceeded, an automatic extension is performed if the immediate right neighbour is a free space area.

The number of container extents and sequential files per raw section is limited to 338.

This utility is a multi-function utility.

Procedure Flow



Data Set	Environment Variable	Storage Medium	Additional Information
Control statements	stdin		Utilities Manual
ADADEV messages	stdout		Messages and Codes
Adabas sequential file	DEVxyz (see note 1)	Disk, Tape (see note 2)	



Notes:

1. xyz = PLG, CLG, 00n, OUT, ERR, LOG, EXP, DTA, DVT.
2. A named pipe can be used for this sequential file (see Administration, Using Utilities for details).

The sequential files DEVPLG, DEVCLG, DEV00n, DEVOUT, DEVERR, DEVEXP, DEVDTA, DEV DVT and DEVLOG can have multiple extents. For detailed information about sequential files with multiple extents, see *Administration, Using Utilities*.

Checkpoints

The utility writes no checkpoints.

Control Parameters

The following control parameters are available:

```
ALLOCATE = keyword[,START_SECTOR = number]
           [,BLOCKSIZE = numberKB] ,SIZE = number [B|M]

CHANGE = (keyword,keyword)

COMBINE = keyword, DESTINATION = string

COPY = keyword, DESTINATION = string

DBID = number

DEALLOCATE = {*|keyword}

FREE_SPACE

INITIALIZE

LAYOUT

D [NO]MOUNTCHECK

MOVE = keyword, DESTINATION = string

NEW_DBID = (container-name, new-dbid)

REALLOCATE = {*|keyword}

RESET

RESIZE

M SECTION = string

UNLOCK = keyword
```

ALLOCATE

```
ALLOCATE = keyword[,START_SECTOR = number]
           [,BLOCKSIZE = numberKB] ,SIZE = number [B|M]
```

In accordance with the keyword specified, this function allocates space for an Adabas container file or Adabas sequential file. The space is allocated using a best-fit algorithm if no start sector is specified. The keyword can take the values ASSO_x, DATA_x, WORK_x, TEMP_x and SORT_x (where *x* is a number between 1 and the maximum number of extents allowed, as described in *Administration, Using Utilities*), PLG, CLG, BCK, BCKOUT, RECOUT, ERR, MUPLOG, MUPTMP, ORDEXP, DTA or DVT. The DBID parameter must be set before space can be allocated to an ASSO, DATA or WORK file, or to PLG, CLG, BCK, BCKOUT or RECOUT.

The DBA can use this function to preallocate container files or Adabas sequential files. Reasons for using it include performance aspects, avoiding fragmentation (by specifying the start sectors) and reserving space for a database that is to be created in the future.

If ADADBM or ADAFRM are used to create a container file in a disk section, the preallocated space is taken if it is available. If there was no preallocation made, the allocation is made using a best-fit algorithm. The same sizes must be used when preallocating and creating container files.

START_SECTOR = number

This parameter specifies the sector at which allocation is to start.

BLOCKSIZE = numberKB

Values up to 32KB can be used to specify the container block-size when the allocation is being made in blocks. The defaults are 2KB for ASSO container files, 4KB for all other container files and 1 KB for Adabas sequential files.

SIZE = number [B|M]

This parameter specifies the size of the area to be allocated in blocks or megabytes. If 'B' is appended to the number, the size is in blocks. By default, the size is given in megabytes.

CHANGE

CHANGE = (keyword, keyword)

This function changes the type of the container file or Adabas sequential file specified by the first keyword into the type specified by the second keyword.

The following keyword combinations are allowed:

From	To
WORK1	TEMP1, SORT1, SORT2
TEMP1	WORK1, SORT1, SORT2
SORT1	WORK1, TEMP1, SORT2
SORT2	WORK1, TEMP1, SORT1
DTA	MUPLOG
MUPLOG	DTA
RECOU	PLG (* see note)
BCKOUT	BCK (* see note)



Note: (*) If RECOU [BCKOUT] is a copy of PLG.n [BCK00n], the new Adabas sequential file name will be the corresponding name.

The DBID parameter must be set before the keywords WORK1, RECOU or BCKOUT are used.

The WORK1 container of a given database can only be changed to a SORT or TEMP container if there is no autorestart pending.

COMBINE

COMBINE = keyword, DESTINATION = string

This parameter combines multiple Adabas sequential file extents into a single extent. The keyword can take the values PLG.n, CLG.n, BCK00n, BCKOUT, RECOU.n, ERR, MUPLOG, ORDEXP, DTA or DVT. The keywords can also be followed by an extent label (m). In some cases, the DBID parameter must be set (see ALLOCATE for further information).

A COMBINE can start at an arbitrary extent, but must end with the final (end-of-file) extent. All subsequent extents can be specified interactively, or by predefined environment variables. If DEVxyz is set, the first extent is taken from the default path name. If DEVxyz is not set, Adabas looks in the current disk section to find the first extent.

'string' is either the path name of the device file that represents the raw interface of a disk section, the path name of a tape device, the (path) name of a non-existent file of a file system or a period (".").

See *Administration, Using Utilities* for further information.

Example 1:

In this example, PLOG 2 of database 100 consists of 3 extents that are all located at the same disk section /dev/rdsk/c4d0s2.

Environment Variable	Setting
DEVPLG	/dev/rdsk/c4d0s2 /dev/rdsk/c4d0s2

The ADADEV commands are as follows:

```
adadev: section=/dev/rdsk/...
adadev: dbid=100
adadev: combine=plg.2(1)
adadev: destination=PLOG_2
```

All extents of PLOG 2 are combined into one file which will be written into the file system under the name PLOG_2. The section must be specified twice in DEVPLG so that it can flip-flop.

Example 2:

In this example, PLOG 3 of database 100 consists of 9 extents, and the extents 5 to 9 are distributed across four disk sections:

Environment Variable	Setting
DEVPLG	/dev/rdsk/c3d0s2 # contains PLG.3(5) and PLG.3(9) EOF
DEVPLG2	/dev/rdsk/c4d0s2 # contains PLG.3(6)
DEVPLG3	/dev/rdsk/c5d0s2 # contains PLG.3(7)
DEVPLG4	/dev/rdsk/c6d0s2 # contains PLG.3(8)

The ADADEV commands are as follows:

```
adadev: section=/dev/rdsk/...
adadev: dbid=100
adadev: combine=plg.3(5)
adadev: destination=PLOG.3(5)
```

Extents 5 to 9 of PLOG 3 are combined into one extent in PLOG 3 with the extent number 5 and the EOF label. The combined PLOG extent is created in the current directory under the name PLOG.3(5).

COPY

```
COPY = keyword, DESTINATION = string
```

This function copies a container file (ASSO1, DATA1, etc.) or an Adabas sequential file (RECOU, DVT, etc.) from its present location to a specified destination (DESTINATION=string).

Valid keywords for the container files are ASSO x , DATA x and WORK x , where x is a number between 1 and the maximum number of extents allowed, as described in *Administration, Using Utilities*.

Valid keywords for the Adabas sequential files are PLG.n, CLG.n, BCK00n, BCKOUT, RECOU.n, ERR, MUPLOG, ORDEXP, DTA, and DVT. The 'n' extension on the Adabas sequential file keywords designates an extent number.

If a container file is to be copied, you must first set the appropriate environment variable (e.g. ASSO1) to the source location before the copy is attempted.

If an Adabas sequential file is to be copied into the current disk section, as specified by the SECTION parameter (e.g. SECTION=/dev/c5d0s2), the environment variable DEVxyz (where xyz may take the values PLG, CLG, 00n, OUT, ERR, LOG, EXP, DTA, and DVT) must be set to the Adabas sequential file or device file.

The DESTINATION keyword may be specified as either a path name for a raw disk section, a path name of a tape device, a path name of a non-existent file in the file system, a named pipe or a period '.', which indicates that the file will be copied into the current disk section.

In some cases, the DBID must be specified in order to uniquely identify the file before the COPY function can be executed: please refer to the ALLOCATE function for a list of those files that require the DBID.

Examples:

```
adadev: section=/dev/r1v02
adadev: dbid=23
adadev: copy=WORK1, destination=/FS/fs0395/SAG/ada/db023/WORK1.023
```

WORK1 is copied from a raw device to a file system.

The environment variable WORK1 has the value /FS/fs0395/SAG/ada/db023/WORK1.023.

```
adadev: section=/dev/rlv02
adadev: dbid=023
adadev: copy=WORK1, destination=.
```

WORK1 is copied from a file system to the currently-selected raw device.

DBID

```
DBID = number
```

This parameter specifies the database of an ASSO, DATA or WORK container file, or of a PLG, CLG, BCK, BCKOUT or RECOU Adabas sequential file.

DEALLOCATE

```
DEALLOCATE = { * | keyword }
```

In accordance with the keyword specified, this function deallocates space from an Adabas container file or Adabas sequential file, or from all of the extents of a given database (DEALLOCATE=*). The keyword can take the values ASSO_x, DATA_x, WORK_x, TEMP_x, SORT_x, NUCTMP_x and NUCSRT_x (where x is a number between 1 and the maximum number of extents allowed, as described in *Administration, Using Utilities*), PLG, PLG.n, PLG*, CLG, CLG.n, CLG*, BCK, BCK00n, BCK*, BCKOUT, RECOU, RECOU.n, ERR, MUPLOG, MUPTMP, ORDEXP, DTA or DVT. The Adabas sequential file keywords can also be followed by an extent label (m) or (*). In some cases, the DBID parameter must be set (see ALLOCATE for further information).

The DBA can use this function to deallocate space from container files, e.g. when a database is no longer required or e.g. when a PLOG extent is saved to tape. The deallocated space is managed as free space and can be allocated to other containers.

FREE_SPACE

```
FREE_SPACE
```

This function displays the areas of free space on the current disk section. It is a subset of the LAYOUT function. In order to make it easier to allocate space in units of the default block sizes, this function also lists the size of the free areas in units of 2KB and 4KB.

INITIALIZE

INITIALIZE

Some sectors at the beginning of each disk section are used to manage the allocated and free space areas. This function initializes this management part. Each disk section to be used by Adabas must be initialized beforehand. When a disk section is accessed for Adabas purposes, the first step is for the management part to be verified. A disk section can only be initialized if this verification fails.

LAYOUT

LAYOUT

This function provides a summary of the disk section usage. It lists the container and Adabas sequential areas as well as the free space areas. It also displays the status of a container area (allocated or created). Adabas sequential files can also have the status "during creation". This means that the Adabas sequential file is growing: some of the allocated space has been used up, but some of it is still free.

Following a power failure, TEMP and SORT containers may be locked for read or write (status: rlocked or wlocked). Please refer to the UNLOCK function in this section for further information about unlocking container files.

[NO]MOUNTCHECK

[NO]MOUNTCHECK

If MOUNTCHECK is used, ADADEV checks if a file system is mounted on the disk specified in the ADADEV SECTION parameter. If a file system is mounted, ADADEV terminates. This check can be skipped if NOMOUNTCHECK is specified before the SECTION parameter is used.

The default is MOUNTCHECK.

MOVE

`MOVE = keyword, DESTINATION = string`

MOVE is essentially the same as COPY, however, with the difference that the source file is removed (see [COPY](#) for further information).

NEW_DBID

```
NEW_DBID = (container-name, new-dbid)
```

This function changes the DBID of Adabas files within raw sections (for example, when the number of a database has been changed and the PLOGs are to be applied to the database with the new number).

The following can be specified for container-name:

- all containers/sequential files with an associated database ID (ASSO, DATA, PLG ...)
- wildcard character '*' with PLOG and CLOG (PLG*, CLG*) to change all files within the section
- when specifying a given PLOG (e.g. PLG.175) and there are sequential file extents (PLG.175(1) etc.), all these occurrences are changed

The following cannot be specified for container-name:

- containers without a DBID (SORT, DTA, ...)
- specific PLOG file extents (e.g. "PLG.175(1)")

Example

The following example shows how to change the DBID from 1 to 2:

```
adadev section=xxx dbid=1 new_dbid=(asso1,2)
adadev section=xxx dbid=1 new_dbid=(plg.175,2)
adadev section=xxx dbid=1 new_dbid=(plg\*,2)
```

REALLOCATE

```
REALLOCATE = { * | keyword }
```

This function deallocates space and directly allocates it in a single step. The keyword can take the same values as for the DEALLOCATE function. Adabas sequential file areas that are <=50 KB are always deallocated.

This function can be used as a short cut when deallocating a container file and then allocating it again at the same location. It is particularly useful if ADAFRM aborts while creating a database, with some container files already created and preallocated at given start sectors.

There is a flag for each container in a disk section, which indicates whether the container or Adabas sequential file has actually been created or whether the space has only been allocated for it. For security reasons, an existing container or Adabas sequential file cannot be overwritten just by creating it again: it must be deallocated or reallocated first.

RESET

RESET

The management part of the current disk section is set to binary zero if the section is initialized. This function is used if a disk section which has the same start sector as an overlapping, initialized disk section is to be initialized.

RESIZE

RESIZE

If a new, overlapping disk section is used with the same start sector as the current disk section, this function updates the size of the section to the size of the new section. The new section can be larger or smaller than the current section. If it is larger, the free space at the end of the section is increased. If it is smaller, existing free space at the end of the section is decreased.

SECTION

SECTION = string

This parameter selects the disk section to be used. The (path) name of the device file that represents the raw I/O interface of the disk section must be specified.

UNLOCK

UNLOCK = keyword

This parameter is used to unlock a container of an abnormally terminated utility (for example kill -9). The keyword can take the values TEMP1, SORT1 or SORT2.

Following a power failure, locked container files must be unlocked using the UNLOCK function in order to use them further. If an Adabas sequential file consists of more than one physical extent, the last extent is marked with EOF in the status field.

11 ADAERR (Error File Report)

▪ Functional Overview	128
▪ Procedure Flow	129
▪ Checkpoints	129
▪ Control Parameter	129
▪ Example	130
▪ Rejected Data Records	130

This chapter describes the utility "ADAERR".

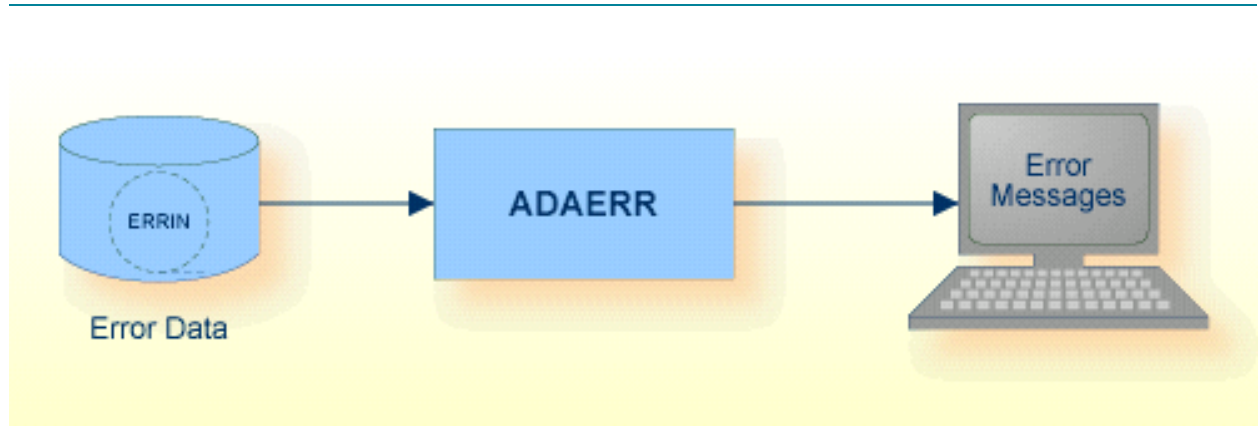
Functional Overview

The ADAERR utility displays the contents of error files generated by the utilities


- ADACMP
- ADADCU
- ADAINV
- ADAMUP
- ADAREC

This utility is a single-function utility.

Procedure Flow



Data Set	Environment Variable/ Logical Name	Storage Medium	Additional Information
Error data	ERRIN	Disk, Tape (* see note)	
Error messages	stdout/ SYS\$OUTPUT		

 **Note:** (*) A named pipe can be used for this sequential file (not on Open VMS, see *Administration, Using Utilities* for details).

The sequential file ERRIN can have multiple extents. For detailed information about sequential files with multiple extents, see *Administration, Using Utilities*.

Checkpoints

The utility writes no checkpoints.

Control Parameter

The following control parameter is available:

D [NO]DUMP

[NO]DUMP

```
[NO]DUMP
```

If NODUMP is specified, only a description (length of record, ISN of the record etc.) of each error record will be output, but not the actual record content. See the section Rejected Data Records in this section for information on the contents of the error records.

If DUMP is specified, the record content will be dumped in addition to the record description. For ADACMP, the decompressed record will be dumped, whereas for ADADCU the compressed record will be dumped.

The default is NODUMP.

Example

```
$ adaerr
```

```
%ADAERR-I-STARTED,      11-OCT-2006 18:59:20, Version 6.1.1
%ADAERR-I-RECN0TF, Record NOT found for ISN 317 in file 49
%ADAERR-I-PLOGRB,  from record 1 in block 6 on PLOG 1
%ADAERR-I-IOCNT,       1 IO  on dataset ERRIN
%ADAERR-I-TERMINATED,  11-OCT-2006 18:59:20, elapsed time: 00:00:01
```

Rejected Data Records

Any records rejected by the following utilities are written to the error file in variable-length format.

- ADACMP
- ADADCU
- ADAINV
- ADAMUP
- ADAREC

The structure of the error records is contained as a header file in the Adabas distribution kit - \$ADADIR/\$ADAVERS/inc/iodesam.h on Unix, %ADADIR%\%ADAVERS%\..\inc\iodesam.h on Windows.

12 ADAFDU (File Definition)

▪ Functional Overview	132
▪ Procedure Flow	133
▪ Checkpoints	135
▪ Control Parameters	135
▪ Examples	149

This chapter describes the utility "ADAFDU".

Functional Overview

The file definition utility ADAFDU defines a new base file and/or a LOB file in a database. It does not require the Adabas nucleus to be active.

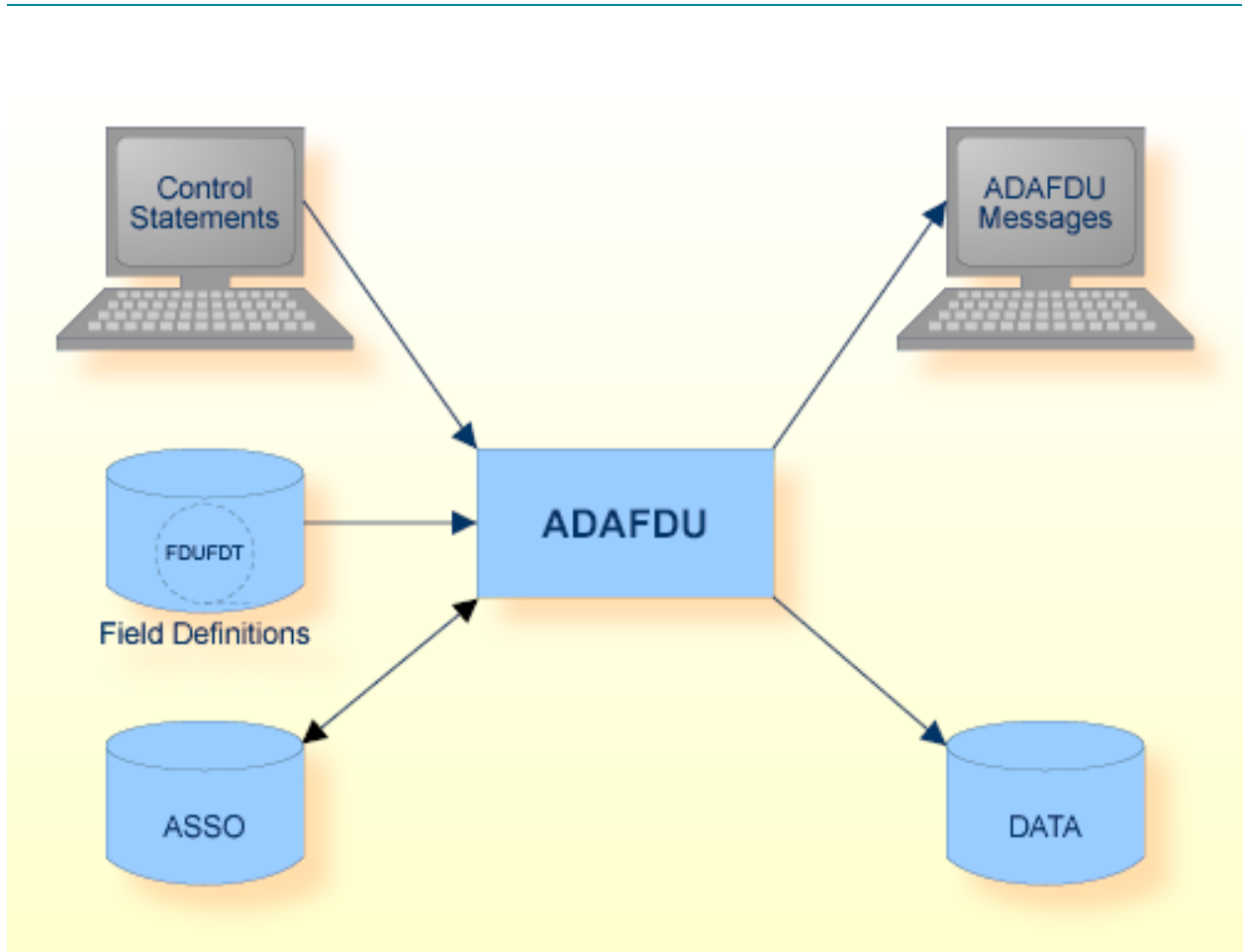
The field definitions for a base file, including special descriptor definitions and referential integrity definitions for foreign keys, are read from the sequential file FDUFDT; the field definition of a LOB file is predefined. Additional input for ADAFDU is provided by parameters.

See *Administration, FDT Record Structure* for information about the syntax and use of the data definitions to define the logical structure of the file in the database.

See *Administration, Loading And Unloading Data, File Space Estimation* for information about formulae for calculating the Associator and Data Storage space requirements for a file.

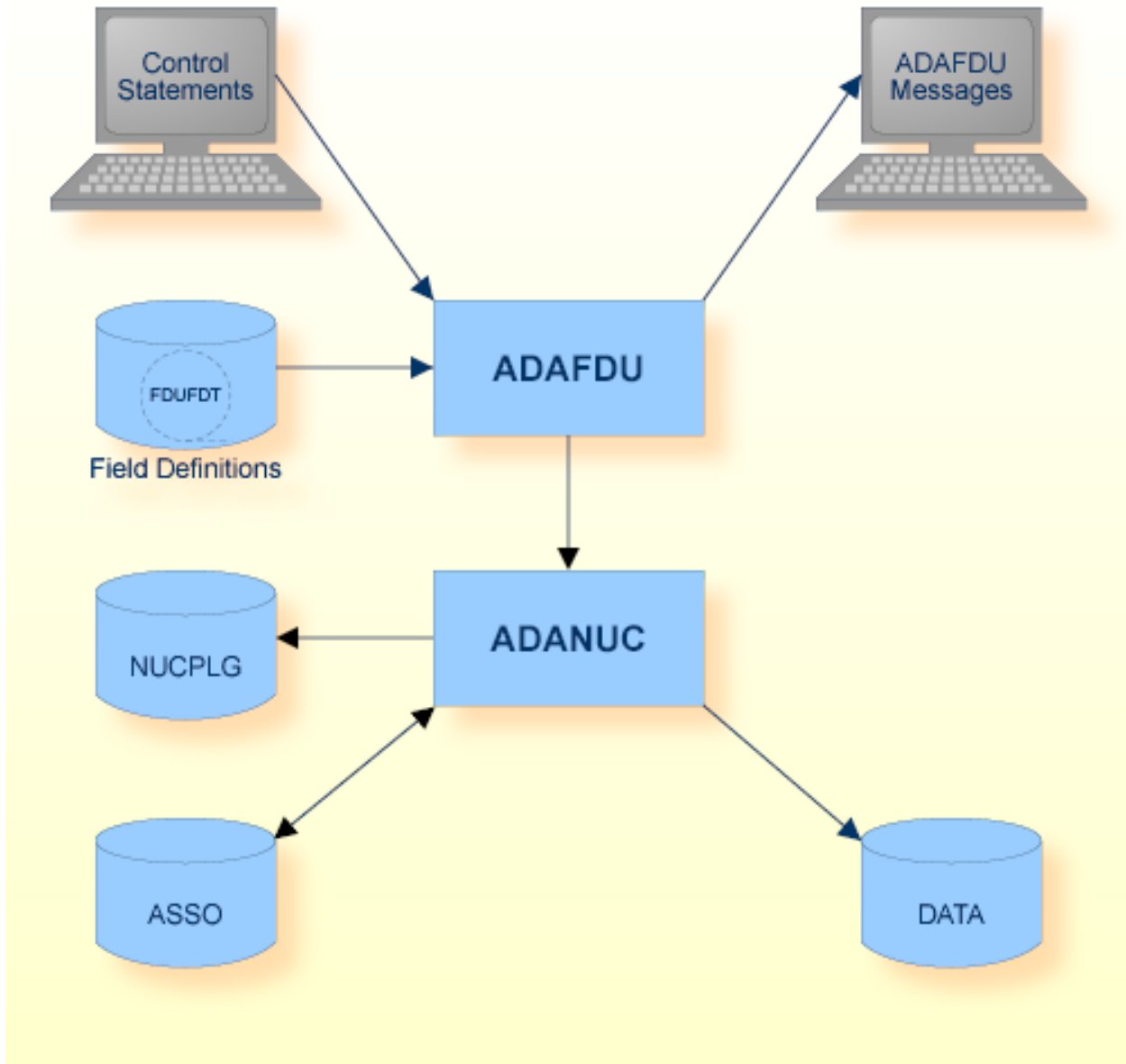
This utility is a single-function utility.

Procedure Flow



Offline Mode

If the nucleus is not active, ADAFDU itself creates the new file in ASSO and DATA



Online Mode

If the nucleus is active, ADAFDU calls the nucleus to create the new file in ASSO and DATA. In this case, no checkpoint is written, but the file creation is logged in the database log, and in case of a recovery, the file is created automatically.

Data Set	Environment Variable/ Logical Name	Storage Medium	Additional Information
Associator	ASSOx	Disk	
Data storage	DATAx	Disk	
Control statements	stdin/ SYS\$INPUT		Utilities Manual
ADAFDU messages	stdout/ SYS\$OUTPUT		Messages and Codes
FDT information	FDUFDT	Disk,Tape (* see note)	
Protection Log	NUCPLG	Disk	Utilities Manual ADAPLP



Note: (*) A named pipe can be used for this sequential file (not on OpenVMS, see *Administration, Using Utilities* for details).

Checkpoints

The utility writes a SYNp checkpoint if it is performed offline. If the utility is performed online, the file definition is written to the PLOG, a SYNx checkpoint is written.

Control Parameters

The following control parameters are available:

```

ACBLOCKSIZE = numberK
ACRABN = number
ADAM_KEY = key
D ADAM_OVERFLOW = number
D ADAM_PARAMETER = number
ADD_LOBFILE = (number,number)
D ASSOPFAC = number
D [NO]BT
D [NO]CIPHER

```

```
D   CONTIGUOUS = ([AC], [,DS] [,NI] [,UI])
D   DATAPFAC = number
M   DBID = number
    DSBLOCKSIZE = numberK
    DSRABN = number
D   DSSIZE = number[B|M]
    FDT
    FILE = number
D   [NO]FORMAT
    LOBFILE = number [,LOBSIZE = number[B|M]]
D   [NO]LOWER_CASE_FIELD_NAMES
D   MAXISN = number
D   NAME(=|:) string
    NIBLOCKSIZE = numberK|(numberK,numberK)
    NIRABN = number|(number,number)
D   NISIZE = number[B|M]|(number[B|M],number[B|M])
    [NO]PGM_REFRESH
D   REUSE = (keyword [,keyword])
    SYFMAX = number
    UIBLOCKSIZE = numberK|(numberK,numberK)
    UIRABN = number|(number,number)
D   UISIZE = number[B|M]|(number[B|M],number[B|M])
```


ACBLOCKSIZE

```
ACBLOCKSIZE = numberK
```

This parameter allows you to specify a block size for the allocation of the address converter extent.

Example:

```
acblocksize = 6k
```

The address converter will be allocated with a block size of 6 kilobytes.

If the database does not contain enough space with this block size, ADAFDU aborts.

ACRABN

```
ACRABN = number
```

This parameter specifies the RABN at which the space allocation for the Address Converter is to start.

This parameter can be used to allocate the Address Converter to a given container file extent.

If this parameter is omitted, ADAFDU assigns the starting RABN.

ADAM_KEY

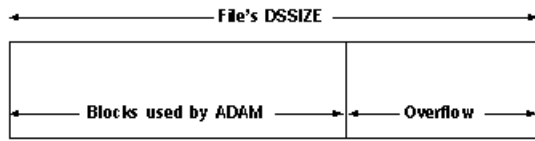
```
ADAM_KEY = key
```

If this parameter is specified, the file is defined as an ADAM file. The key can be either a descriptor name or the keyword 'ISN'. If an ADAM key is used, it must be defined with the UQ option in the FDT. It must not be a sub-, super-, phonetic or hyperdescriptor. It must not be a multiple-value field or a field within a periodic group. It must not have the NU/NC option.

ADAM_OVERFLOW

```
ADAM_OVERFLOW = number
```

This parameter specifies the number of DATA overflow blocks for the file. Overflow blocks are required in case ADAM-calculated blocks get full. The overflow blocks are taken from the end of the file's DATA blocks.



At least one overflow block must be allocated.

The maximum is DSSIZE - 1.



Note: When checking the maximum value, and DSSIZE is specified in megabytes, it is assumed that the Data Storage block size is 32 - independent of the actual value. If you want to specify a larger value for ADAM_OVERFLOW, which is possible with a smaller Data Storage block size, DSSIZE must be specified in blocks.

The default is 1.

ADAM_PARAMETER

```
ADAM_PARAMETER = number
```

This parameter specifies the number of consecutive ISNs to be stored in one block if the keyword 'ISN' is specified for the ADAM_KEY parameter.

If the ADAM key is a descriptor with fixed-point format, the parameter specifies the number of consecutive values for one block. For other key formats, it specifies an offset into the values. See Administration for more information.

A value may be specified in the range 1 to 10000.

The default value is 8.

ADD_LOBFILE

```
ADD_LOBFILE = (number, number)
```

The parameter ADD_LOBFILE is used to create a LOB file and assign it to an existing base file that is specified by the first number, the base file must not yet have an assigned LOB file. A LOB file, with the file number specified by the second number, is generated and assigned to the base file, and the base file is enabled for LOB processing. A file with the specified file number must not yet exist. The maximum number that can be specified is 32000. You can specify the parameters describing the data storage, the address converter, the normal and upper index of the LOB file, but the following should be taken into consideration:

- The block size for LOB file data blocks must be 32 KB.
- The block size for LOB NI and UI blocks must be < 16 KB.

It is not possible to specify FILE if you specify ADD_LOBFILE, and vice versa.

Because there are some predefined requirements for a LOB file, not all the other ADAFDU parameters make sense in connection with ADD_LOBFILE, for example the ADAM_* parameters. These parameters are ignored by ADAFDU when the LOB file is added.

ASSOPFAC

```
ASSOPFAC = number
```

This parameter specifies the padding factor to be used for the file's index. The number specified is the percentage of each index block which is not to be used by a subsequent run of the mass update utility ADAMUP. This padding area is reserved for future use if additional entries have to be added to the block by the Adabas nucleus. This avoids the necessity of having to relocate overflow entries to another block.

A value may be specified in the range 0 to 95.

A small padding factor (0 to 10) should be specified if little or no descriptor updating is expected. A larger padding factor (10 to 50) should be specified if there is a large amount of descriptor updating in which new descriptor values are created.

You can change the padding factor at a later time using the utility ADAORD.

The default padding factor is 5.

[NO]BT

```
[NO]BT
```

If NOBT is specified, this file will be a no-BT file, which means that modifications to this file are not made within normal transaction logic, and all modifications are kept in the database even if a transaction is backed out.

BT is the default.



Note: The following points should be considered if the nucleus crashes:

- All database modifications for a no-BT file issued before the last ET command are applied to the database.
- It is not defined whether database modifications for a no-BT file issued after the last ET command are applied to the database or not.

[NO]CIPHER

```
[NO]CIPHER
```

This option can be used to enable or disable data record ciphering.

Ciphering prevents the unauthorized analysis of Adabas container files. If ciphering is enabled, data records are ciphered when they are stored in a database by either the Adabas nucleus or by the mass update utility ADAMUP. The data records are then deciphered when they are requested by a user or application: this means that the ciphering is completely transparent to the user or application. See *Administration, Adabas Security Facilities* for further information about ciphering.

The default is NOCIPHER.

CONTIGUOUS

```
CONTIGUOUS = ( [AC] [,DS] [,NI] [,UI] )
```

This parameter is used to control ADAFDU's space allocations. If specified, ADAFDU ensures that only the first logical extent of the types specified is used.

By default, ADAFDU makes contiguous-best-try allocations.

DATAPFAC

```
DATAPFAC = number
```

This parameter specifies the padding factor to be used for the file's Data Storage. The number specified is the percentage of each data block which is not to be used when subsequently adding new records to the file with the mass update utility ADAMUP or with the Adabas nucleus. This padding area is reserved for future use if any record in the block requires additional space as a result of record updating by the Adabas nucleus. This avoids the necessity of having to relocate the record to another block.

A value in the range 0 to 95 may be specified.

A small padding factor (0 to 10) should be specified if there is little or no record expansion. A larger padding factor (10 to 50) should be specified if there is a large amount of record updating which will cause expansion.

You can change the padding factor at a later time using the utility ADAORD.

The default padding factor is 5.

DBID

```
DBID = number
```

This parameter selects the database to be used.

DSBLOCKSIZE

```
DSBLOCKSIZE = numberK
```

This parameter allows you to specify a block size for the allocation of the data storage extent.

Example:

```
dsblocksize = 6k
```

Data storage will be allocated with a block size of 6 kilobytes.

If the database does not contain enough space with this block size, ADAFDU aborts.

DSRABN

```
DSRABN = number
```

This parameter specifies the RABN at which the space allocation for Data Storage is to start.

This parameter can be used to allocate Data Storage to a given container file extent.

If this parameter is omitted, ADAFDU assigns the starting RABN.

DSSIZE

```
DSSIZE = number [B|M]
```

This parameter specifies the number of blocks or megabytes to be assigned to Data Storage. By default, the size is given in megabytes.

The value specified for DSSIZE determines the size of the logical extent allocated to Data Storage for the file.

A contiguous-best-try allocation is made unless CONTIGUOUS=DS has been specified.

This parameter is mandatory for ADAM files - the dependencies between the parameters ADAM_OVERFLOW and DSSIZE are described for the parameter ADAM_OVERFLOW.

For non-ADAM files, this parameter can be omitted; in this case Adabas calculates a reasonable number of blocks to be used for Data Storage. If the size that is actually required is larger, the file is automatically increased.

FDT

```
FDT
```

If this parameter is specified, the FDT contained in the sequential file FDUFDT is displayed.

FILE

```
FILE = number
```

This parameter is required when a base file is to be created; it specifies the file number to be assigned to the file.

The 'number' specified must not be currently assigned to another file in the database and must not exceed the maximum file number defined for the database. The maximum number that can be specified is 32000.

File numbers can be assigned in any sequence.

It is not possible to specify FILE if you specify ADD_LOBFILE, and vice versa.

[NO]FORMAT

```
[NO]FORMAT
```

This option is used to control whether the RABNs allocated for the file's index and Data Storage are to be formatted or not. The RABNs of the file's Address Converter are always formatted.

The default is NOFORMAT.

LOBFILE

```
LOBFILE = number [, LOBSIZE = number[B|M]]
```

If LOBFILE is specified, a LOB file with the specified number is generated and assigned to the base file to be created, and the base file is enabled for LOB processing. A LOB file with the specified file number must not already exist. The maximum number that can be specified is 32000. You should take the following into consideration:

- The block size for LOB file data blocks will be 32 KB.
- The block size for LOB NI and UI blocks will be < 16 KB.
- LOBSIZE specifies the size in Data storage of the LOB file, analogously to the parameter DSSIZE.
- Adabas calculates reasonable sizes for the Address converter, the normal and upper index of the LOB file. If you want to specify these values yourself, you should create the base file first without specifying LOBFILE, and then you should call ADAFDU again and add the LOB file with the ADD_LOBFILE parameter.

[NO]LOWER_CASE_FIELD_NAMES

[NO]LOWER_CASE_FIELD_NAMES

If LOWER_CASE_FIELD_NAMES is specified, Adabas field names are not converted to upper case. If NOLOWER_CASE_FIELD_NAMES is specified, Adabas field names are converted to upper case. The default is NOLOWER_CASE_FIELD_NAMES.

MAXISN

MAXISN = number

This parameter specifies the highest ISN expected in the file. The file definition utility ADAFDU uses this parameter to determine the amount of space to be allocated for the file's Address Converter (AC). The default value for MAXISN is 5000.

A contiguous-best-try allocation is made unless CONTIGUOUS=AC has been specified.



Note: The value is rounded up to the number of ISNs that fit into the Address converter blocks required to store MAXISN ISNs in the Address converter, the exact value used as MAXISN for the file is:
 $(\text{MAXISN specified} / (\text{Address converter block size} / 4) + 1) * (\text{Address converter block size} / 4) - 1$. For example, using an Address converter with a block size of 4KB, the default value of 5000 is increased to $(5000 / (4096 / 4) + 1) * (4096 / 4) - 1 = 5119$.

NAME

```
NAME {=|:} string
```

This parameter specifies the name to be assigned to the file. This name will appear together with data about this file in the database status report produced by the report utility ADAREP. A maximum of 16 characters are permitted. If you specify an equals sign, the value given for 'string' will be converted to upper case; if you specify a colon, no upper-case conversion is performed

The default value is FILE-n, where n is the file number.

NIBLOCKSIZE

```
NIBLOCKSIZE = numberK|(numberK,numberK)
```

This parameter allows you to specify a block size for the allocation of the Normal Index. Note that the Normal Index requires a block size ≥ 16 KB for large index values > 253 bytes, while a smaller block is allocated for descriptors with smaller descriptor values. The following must be taken into consideration:

- If you specify one block size, the file is created with all normal index blocks having this size.
- If you specify two block sizes, one value should be $< 16K$, and one value should be $\geq 16K$. You should also specify two values for NISIZE; the first value for NIBLOCKSIZE corresponds to the first value of NISIZE, and the second value for NIBLOCKSIZE corresponds to the second value of NISIZE.

Examples:

```
niblocksize = 6k
```

The normal index will be allocated with a block size of 6 kilobytes.

```
niblocksize = (8k,32k)  
nsize = (1000b,10m)
```

The normal index will be allocated with 1000 blocks of block size 8 KB and 10 MB of block size 32 KB.

If the database does not contain enough space with this block size, ADAFDU aborts.

NIRABN

```
NIRABN = number |(number, number)
```

This parameter specifies the RABN at which the space allocation for the Normal Index is to start. This parameter can be used to allocate the Normal Index to a given container file extent.

If two RABNs have been specified, one should have a block size < 16KB, and the other should have a block size of >= 16KB.

If this parameter is omitted, ADAFDU assigns the starting RABNs.

If both NIBLOCKSIZE and NIRABN are specified, the block sizes of the RABNs specified as NIRABN must be equal to the values specified as NIBLOCKSIZE.

NISIZE

```
NISIZE = number [B|M] |(number [B|M], number [B|M])
```

This parameter defines the number of blocks or megabytes to be assigned to the Normal Index. By default, the size is in megabytes.

If the block size cannot be derived from the NIBLOCKSIZE or the NIRABN parameter, the first value for NISIZE is used for blocks < 16KB, and the second value is used for blocks >= 16KB.

A contiguous-best-try allocation is made unless CONTIGUOUS=NI has been specified.

If this parameter is omitted, Adabas calculates a reasonable number of blocks to be used for the normal index.

Examples:

```
adafdu: nsize = 100b
```

If the block size cannot be derived from the NIBLOCKSIZE or NIRABN parameter, 100 blocks with block size < 16KB are allocated for the Normal Index.

```
adafdu: nsize = (10m,1000b)
```

If the block size cannot be derived from the NIBLOCKSIZE or NIRABN parameter, 10 MB of blocks with block size < 16KB and 1000 blocks of block size >= 16KB are allocated for the Normal Index.

[NO]PGM_REFRESH

```
[NO]PGM_REFRESH
```

If PGM_REFRESH is specified, the file can be refreshed by an E1 command (reset to a state of zero records loaded) when it is loaded.

The default is NOPGM_REFRESH.

REUSE

```
REUSE = ( keyword [,keyword] )
```

The REUSE parameter controls the reuse of Data Storage space or ISNs by Adabas.

REUSE = [NO]DS

NODS causes all newly-added records, together with records that have to be moved to another block (as a result of record expansion caused by updating) to be placed in the last used block in the Data Storage extent allocated to the file. If there is not sufficient space in this block, the next block is used.

If the DS keyword is specified, Adabas will scan the Data Storage Space Table (DSST) in order to locate a block with sufficient space. In this case, the first block found with sufficient space will be used.

The file control block for the specified file is modified to indicate the type of allocation to be used when adding new records or moving updated records.

The default value is DS.

REUSE = [NO]ISN

If REUSE is set to NOISN, Adabas does not reuse the ISN of a deleted record for a new record. Each new record will be assigned the next-highest unused ISN.

The ISN keyword specifies that Adabas may reuse the ISN of a deleted record.

The default value is NOISN.

Examples

```
adafdu: reuse = (isn, ds)
```

ISNs of deleted records can be reassigned to new records. The DSST is scanned for free space when a record is added to the database or when an updated record is moved in the database.

```
adafdu: reuse = isn
```

Reuse of data storage and ISNs is allowed.

```
adafdu: reuse = <cr>
```

Reuse of data storage and no reuse of ISNs is specified. This is the default setting.

SYFMAX

```
SYFMAX = number
```

This parameter specifies the maximum number of values generated for a system generated multiple-value field. There is no explicit maximum value, but you should bear in mind, that you can get a record overflow if the value is defined too high; the compressed data record should also fit into one DATA block if SYFMAX values are defined for system generated multiple-value fields.

The default value is 1.

UIBLOCKSIZE

```
UIBLOCKSIZE = numberK | (numberK, numberK)
```

This parameter allows you to specify a block size for the allocation of the Upper Index. Note that the Upper Index requires a block size ≥ 16 KB for large index values > 253 bytes, while a smaller block is allocated for descriptors with smaller descriptor values. The following must be taken into consideration:

- If you specify one block size, the file is created with all normal index blocks having this size.
- If you specify two block sizes, one value should be < 16 K, and one value should be ≥ 16 K. You should also specify two values for UISIZE; the first value for UIBLOCKSIZE corresponds to the first value of UISIZE, and the second value for UIBLOCKSIZE corresponds to the second value of UISIZE.

Examples:

```
uiblocksize = 6k
```

The upper index will be allocated with a block size of 6 kilobytes.

```
uiblocksize = (8k,32k)  
uisize = (1000b,10m)
```

The upper index will be allocated with 1000 blocks of block size 8 KB and 10 MB of block size 32 KB.

If the database does not contain enough space with this block size, ADAFDU aborts.

UIRABN

```
UIRABN = number|(number,number)
```

This parameter specifies the RABN at which the space allocation for the Upper Index is to start. This parameter can be used to allocate the Upper Index to a given container file extent.

If two RABNs have been specified, one should have a block size < 16KB, and the other should have a block size of >= 16KB.

If both UIBLOCKSIZE and UIRABN are specified, the block sizes of the RABNs specified as UIRABN must be equal to the values specified as UIBLOCKSIZE.

If this parameter is omitted, ADAFDU assigns the starting RABN.

UISIZE

```
UISIZE = number [B | M]
```

This parameter defines the number of blocks or megabytes to be assigned to the Upper Index. By default, the size is in megabytes.

If the block size cannot be derived from the UIBLOCKSIZE or the UIRABN parameter, the first value for UISIZE is used for blocks < 16KB, and the second value is used for blocks >= 16KB.

A contiguous-best-try allocation is made unless CONTIGUOUS=UI has been specified.

If this parameter is omitted, Adabas calculates a reasonable number of blocks to be used for the upper index.

Examples

Example:

```
adafdu: dbid = 1, file = 6, maxisn = 20000, dssize = 100B,  
adafdu: assopfac = 10, datapfac = 10,  
adafdu: uisize = 20b, nisize = 5
```

File 6 is to be loaded. The maximum number of expected records preset for the file is 20000. 100 blocks are allocated for Data Storage. The Associator and Data Storage block padding factors are both 10 percent. 20 blocks are allocated for the Upper Index and 5 megabytes for the Normal Index. The Normal Index ISN size is implicitly set to 2.

Example:

```
adafdu: dbid = 1, file = 7, maxisn = 350000,  
adafdu: assopfac = 5, datapfac = 15,  
adafdu: dssize = 100,  
adafdu: uisize = 2, nisize = 30
```

File 7 is to be loaded. The maximum number of expected records preset for the file is 350000. The Associator padding factor is 5 percent. The Data Storage padding factor is 15 percent. 100 megabytes are allocated for Data Storage. The Normal Index ISN size is implicitly set to 4.

Example:

```
adafdu: dbid = 1, file = 8,  
adafdu: maxisn = 10000, dssize = 20,  
adafdu: uisize = 10b, nisize = 50b
```

File 8 is to be loaded. The maximum number of expected records preset for the file is 10000. 20 megabytes are allocated to Data Storage. The padding factor for both the Associator and Data Storage is 5 percent (default).

Example:

```
adafdu: dbid = 1, file = 9, maxisn = 55000, dssize = 2000b, dsrabn = 30629,  
adafdu: uisize = 50b, nisize = 300b,  
adafdu: assopfac = 20, datapfac = 10
```

File 9 is to be loaded. The maximum number of expected records preset for the file is 55000. 2000 blocks are allocated for Data Storage. The Data Storage allocation will start at RABN 30629. 50 blocks are allocated for the Upper Index. 300 blocks are allocated for the Normal Index. The padding factor for the Associator is 20 percent. The padding factor for Data Storage is 10 percent.

Example:

```
adafdu: dbid = 1, file = 10, maxisn = 20000
```

File 10 is to be loaded. The maximum number of records expected for the file is set to 20000. All space allocation will be calculated by Adabas.

13

ADAFIN (File Information Report)

▪ Functional Overview	152
▪ Procedure Flow	153
▪ Checkpoints	154
▪ Control Parameters	154

This chapter describes the utility "ADAFIN".

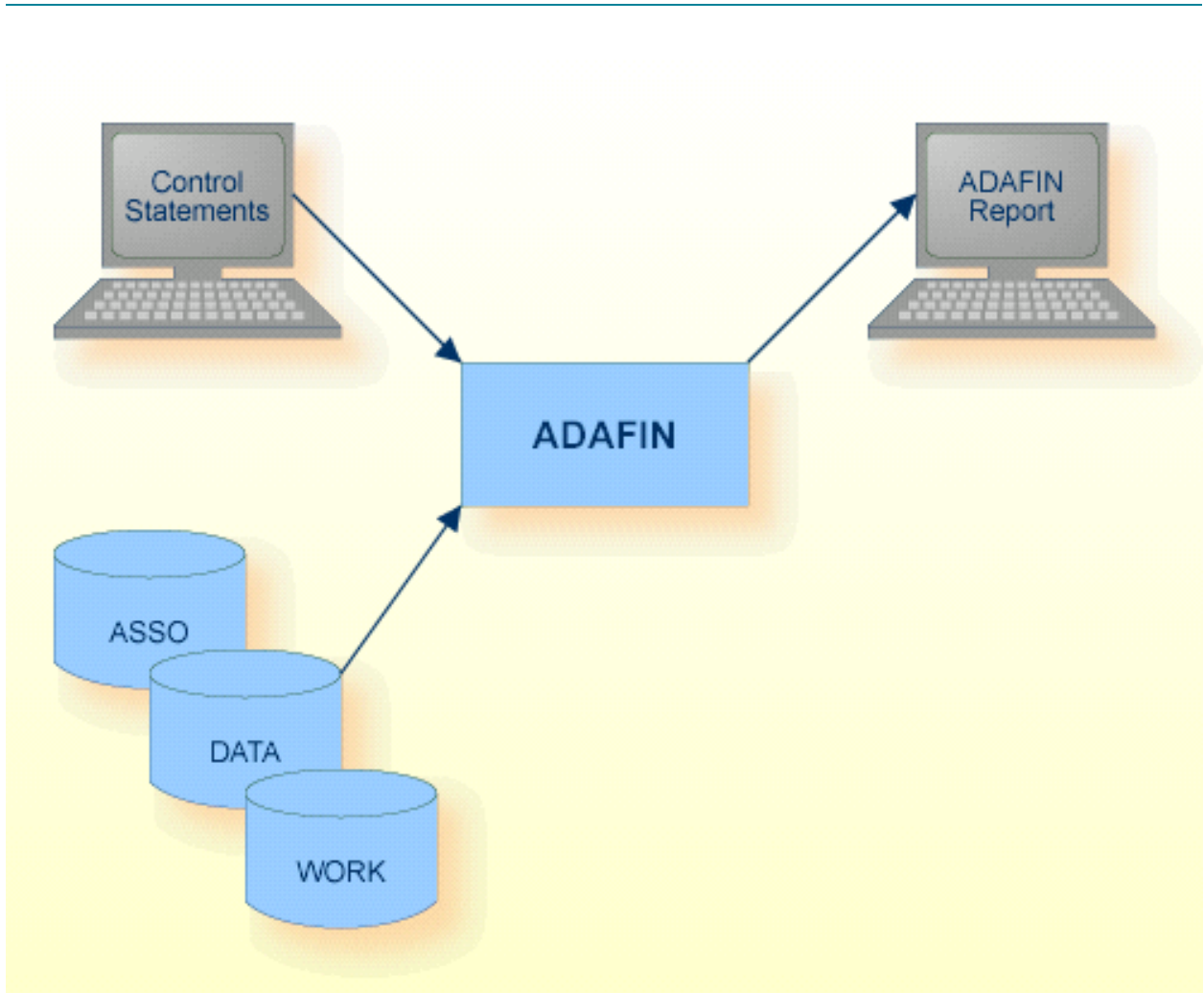
Functional Overview

The file information utility ADAFIN displays

- the FDT,
 - descriptor information, and
 - the number of blocks in the Data Storage, Normal Index or Upper Index and their usage
- of one or more selected files.

This utility is a multi-function utility.

Procedure Flow



Data Set	Environment Variable/ Logical Name	Storage Medium	Additional Information
Associator	ASSOx	Disk	
Data storage	DATAx	Disk	
Control statements	stdin/ SYS\$INPUT		Utilities Manual
ADAFIN messages	stdout/ SYS\$OUTPUT		Messages and Codes
Work	WORK1	Disk	

Checkpoints

The utility writes no checkpoints.

Control Parameters

The following control parameters are available:

```

ADAM_DS = keyword
M  DBID = number
   DESCRIPTOR = { = | : } { * | (string [,string]...) }
   FDT
M  FILE = { * | (number [-number] [,number [-number]]...) }
D  [NO]HISTOGRAM
   USAGE = (keyword [,keyword [,keyword]])

```

ADAM_DS

```
ADAM_DS = keyword
```

This parameter can be used in conjunction with USAGE=DS for ADAM files. It selects the data section of the ADAM file for which information is to be displayed. The following keywords can be used:

Keyword	Meaning
FULL	All of the DS space is selected
ADAM	Only the ADAM area is selected
OVERFLOW	Only the ADAM overflow area is selected

DBID

```
DBID = number
```

This parameter selects the database to be used.

DESCRIPTOR

```
DESCRIPTOR = { = | : } { * | (string [,string]...) }
```

This function defines the list of descriptors for which information is to be displayed. If more than one file is selected, information may only be requested for all descriptors (DESCRIPTOR = *).

The DESCRIPTOR function can only be executed if the selected files are not opened for update with the nucleus running. This function can only be selected in conjunction with the FILE parameter.

The DESCRIPTOR function does not synchronize against parallel updates (for example ADAINV REINVERT).

Examples

```
adafin: file=13, descriptor=ca
```

```
Database 76, File 13 (MISCELLANEOUS ) 27-OCT-2006 08:08:17
```

```
Descriptor CA , Format: A , Options: NU
```

	min	max	ave	
Length	1	233	20.59	
ISNs per value	1	2	1.08	
Values:	different:	86	total:	93
ASSO-Blocks:	NI:	2	UI:	1

```
adafin: file=(11,12), descriptor=*
```

```
Database 1, File 11 (EMPLOYEEES-NAT ) 27-OCT-2006 08:09:39
```

```
Descriptor AA , Format: A , Options: UQ
```

	min	max	ave	
Length	8	8	8.00	
ISNs per value	1	1	1.00	
Values:	different:	1,107	total:	1,107
ASSO-Blocks:	NI:	5	UI:	1

Descriptor AE , Format: A , Options: None

	min	max	ave
Length	3	17	6.78
ISNs per value	1	19	1.37
Values:	different: 804	total: 1,107	
ASSO-Blocks:	NI: 4	UI: 1	

Descriptor AH , Format: P , Options: NC

	min	max	ave
Length	4	4	4.00
ISNs per value	1	43	1.20
Values:	different: 921	total: 1,107	
ASSO-Blocks:	NI: 4	UI: 1	

Descriptor AJ , Format: A , Options: NU

	min	max	ave
Length	3	20	8.52
ISNs per value	1	141	3.60
Values:	different: 307	total: 1,107	
ASSO-Blocks:	NI: 3	UI: 1	

Descriptor A0 , Format: A , Options: None

	min	max	ave
Length	6	6	6.00
ISNs per value	1	99	6.62
Values:	different: 167	total: 1,107	
ASSO-Blocks:	NI: 2	UI: 1	

Descriptor AP , Format: A , Options: NU

	min	max	ave
Length	2	25	12.56
ISNs per value	1	75	4.67

Values: different: 237 total: 1,107
 ASSO-Blocks: NI: 3 UI: 1

Descriptor AZ , Format: A , Options: NU,MU

	min	max	ave
Length	3	3	3.00
ISNs per value	1	843	86.28

Values: different: 21 total: 1,812
 ASSO-Blocks: NI: 2 UI: 1

Super-Descriptor H1 , Format: B , Options: NU

Parent field(s): AU (1 - 2) U
 AV (1 - 2) U

	min	max	ave
Length	4	4	4.00
ISNs per value	1	93	4.17

Values: different: 259 total: 1,081
 ASSO-Blocks: NI: 2 UI: 1

Phonetic-Descriptor PH , Format: A , Options: None

Parent field(s): AE A

	min	max	ave
Length	3	3	3.00
ISNs per value	1	33	1.82

Values: different: 608 total: 1,107
 ASSO-Blocks: NI: 3 UI: 1

Sub-Descriptor S1 , Format: A , Options: None

Parent field(s): AO (1 - 4) A

	min	max	ave
Length	4	4	4.00
ISNs per value	1	208	85.15

Values: different: 13 total: 1,107
 ASSO-Blocks: NI: 2 UI: 1

Super-Descriptor S2 , Format: A , Options: None

Parent field(s): AO (1 - 6) A
 AE (1 - 20) A

	min	max	ave
Length	9	23	12.78
ISNs per value	1	5	1.05
Values:	different: 1,052	total: 1,107	
ASSO-Blocks:	NI: 6	UI: 1	

Super-Descriptor S3 , Format: A , Options: NU,PE

Parent field(s): AR (1 - 3) A
 AS (1 - 9) P

	min	max	ave
Length	12	12	12.00
ISNs per value	1	25	2.15
Values:	different: 1,567	total: 3,383	
ASSO-Blocks:	NI: 10	UI: 1	

Highest PE-occurrence: 5

Database 1, File 12 (VEHICLES) 10-OCT-2006 14:30:39

Descriptor AA , Format: A , Options: UQ,NU

	min	max	ave
Length	6	10	7.91
ISNs per value	1	1	1.00
Values:	different: 772	total: 772	
ASSO-Blocks:	NI: 4	UI: 1	

Descriptor AC , Format: A , Options: None

	min	max	ave
Length	1	8	7.74
ISNs per value	1	24	1.16
Values:	different: 662	total: 773	

ASSO-Blocks: NI: 3 UI: 1

Descriptor AD , Format: A , Options: NU

	min	max	ave
Length	2	14	6.63
ISNs per value	1	179	17.17
Values: different:	45	total:	773
ASSO-Blocks: NI:	1	UI:	1

Descriptor AF , Format: A , Options: NU

	min	max	ave
Length	3	10	4.95
ISNs per value	1	135	11.36
Values: different:	68	total:	773
ASSO-Blocks: NI:	1	UI:	1

Descriptor AH , Format: A , Options: FI

	min	max	ave
Length	1	1	1.00
ISNs per value	169	329	257.66
Values: different:	3	total:	773
ASSO-Blocks: NI:	1	UI:	1

Super-Descriptor A0 , Format: A , Options: NU

Parent field(s): AG (1 - 2) U
 AD (1 - 20) A

	min	max	ave
Length	4	16	8.63
ISNs per value	1	45	4.29
Values: different:	180	total:	773
ASSO-Blocks: NI:	2	UI:	1

Total of 18 descriptors

Information about all descriptors in the specified files is displayed.

FDT

FDT

This parameter displays the Field Definition Tables (FDTs) of the files selected with the FILE parameter. This function can only be selected in conjunction with the FILE parameter.

Example

```
adafin: file=9, fdt
Database 1, File      9 (EMPLOYEES      )          27-OCT-2006 08:11:42
```

Field Definition Table:

Level	I Name	I Length	I Format	I Options	I Flags	I Encoding
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 W	I NU	I	I
2	I AE	I 20	I W	I NU	I SP	I
2	I AD	I 20	I W	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 8	I U	I DE	I	I
1	I AI	I	I	I	I	I
2	I AJ	I 20	I W	I NU,MU	I	I
2	I AK	I 10	I A	I NU	I	I
2	I AL	I 3	I A	I NU	I	I
1	I A2	I	I	I	I	I
2	I AN	I 6	I A	I NU	I	I
2	I AM	I 15	I A	I NU	I	I
1	I AO	I 6	I A	I DE	I SB,SP	I
1	I AP	I 25	I W	I DE,NU	I	I
1	I AQ	I	I	I PE	I	I
2	I AR	I 3	I A	I NU	I SP	I
2	I AS	I 5	I P	I NU	I SP	I
2	I AT	I 5	I P	I NU,MU	I	I
1	I A3	I	I	I	I	I
2	I AU	I 2	I U	I	I SP	I
2	I AV	I 2	I U	I NU	I SP	I
1	I AW	I	I	I PE	I	I
2	I AX	I 8	I U	I NU	I	I
2	I AY	I 8	I U	I NU	I	I
1	I AZ	I 3	I A	I DE,NU,MU	I	I

Type	I Name	I Length	I Format	I Options	I Parent field(s)	Fmt
COLL	I CN	I 11,144	I	I NU,HE	I AE de__PHONEBOOK	I PRIMARY

SUPER	I	H1	I	4	I	B	I	NU	I	AU	(1	-	2)	U
	I		I		I		I		I	AV	(1	-	2)	U
SUB	I	S1	I	4	I	A	I		I	A0	(1	-	4)	A
SUPER	I	S2	I	26	I	A	I	NU	I	A0	(1	-	6)	A
	I		I		I		I		I	AE	(1	-	20)	W
SUPER	I	S3	I	12	I	A	I	NU,PE	I	AR	(1	-	3)	A
	I		I		I		I		I	AS	(1	-	9)	P

FILE

```
FILE = { * | (number [-number] [,number [-number]]...) }
```

This parameter selects one or more files from a database and displays information about these files in accordance with the following parameter. Specifying FILE = * selects all files.

[NO]HISTOGRAM

```
[NO]HISTOGRAM
```

If the HISTOGRAM option is selected, a graphical overview of the descriptor-value length distributions will be provided in all the information that is subsequently displayed by the DESCRIPTOR function.

If HISTOGRAM is used, it must be specified before the DESCRIPTOR parameter.

Using the HISTOGRAM option does not lead to additional I/Os on the data sets.

The default is NOHISTOGRAM.

Example (with HISTOGRAM)

```
adafin: file=9, histogram, descriptor=ap
Database 1, File 9 (EMPLOYEES ) 27-OCT-2006 08:12:44

Descriptor AP , Format: W , Options: NU

min max ave
-----
Length 2 26 12.71
ISNs per value 1 75 4.61

Values: different: 240 total: 1,107
ASSO-Blocks: NI: 3 UI: 1
```

Histogram of descriptor value length for descriptor AP

Length	25%	50%	75%	100%	Frequency
2	*				1
3	*				22
5	*				7
6	*				26
7	*****				124
8	****				83
9	*****				117
10	*****				119
11	***				67
12	****				83
13	*				23
14	**				47
15	**				46
16	**				46
17	*				29
18	*****				101
19	*				27
20	*				29
21	*				33
22	*				17
23	*				20
24	*				21
25	*				5
26	*				14

adafin:

The information that is displayed has the following meaning:

Keyword	Meaning
Length	Each value n shown in this column indicates that there is a descriptor value with a length of n bytes in the file. The range of values in this column lies between the minimum (column "min") and maximum (column "max") values shown in the table before the histogram.
Frequency	The value shown in this column indicates the number of descriptor values for the given descriptor length. The sum of the values in the frequency column is equal to the total number of values for the descriptor in question.

If all of the descriptor values are of the same length, the histogram will be of an unusual type, e.g.:

```
adafin: file=9, histogram, descriptor=aa
Database 1, File      9 (EMPLOYEES      )      27-OCT-2006 08:15:16
```

Descriptor AA , Format: A , Options: UQ

	min	max	ave
Length	8	8	8.00
ISNs per value	1	1	1.00
Values:	different: 1,107	total: 1,107	
ASSO-Blocks:	NI: 5	UI: 1	

Histogram of descriptor value length for descriptor AA

Length	25%	50%	75%	100%	Frequency
8	*****				1,107

This histogram shows that the file only contains descriptor values that have a length of 8 bytes. The file contains a total of 1107 values for the descriptor AA.

Example (with NOHISTOGRAM)

```
adafin: file=9, histogram, descriptor=ap
Database 1, File      9 (EMPLOYEES      )      27-OCT-2006 08:14:24
```

Descriptor AP , Format: W , Options: NU

	min	max	ave
Length	2	26	12.71
ISNs per value	1	75	4.61
Values:	different: 240	total: 1,107	
ASSO-Blocks:	NI: 3	UI: 1	

USAGE

USAGE = (keyword [,keyword [,keyword]])

Depending on the keyword specified, this parameter displays the percentage of used blocks in the file.

Keyword	Meaning
DS	Displays statistics of used blocks in the Data Storage;
NI	Displays statistics of used blocks in the Normal Index;
UI	Displays statistics of used blocks in the Main/Upper Index.

Example

```

adafin: file=13, usage=ds

Database 76, File    13  (MISCELLANEOUS  )           27-OCT-2006 08:16:18

DS - Blocks allocated =          50 , used =          49 , unused =          1

Records:  Number      =          179
         Length: max =      1,991 , min  =          260 , avg   =          997.47

  0%:                                0 blocks
  5%:                                0 blocks
 10%:                                0 blocks
 15%:                                0 blocks
 20%:                                0 blocks
 25%:                                0 blocks
 30%:                                0 blocks
 35%:                                0 blocks
 40%:                                0 blocks
 45%:                                0 blocks
 50%:                                0 blocks
 55%:                                0 blocks
 60%:                                0 blocks
 65%:                                0 blocks
 70%: ****                            2 blocks
 75%:                                0 blocks
 80%: **                              1 block
 85%: *****                          6 blocks
 90%: *****                          13 blocks
 95%: *****                          27 blocks
100%:                                0 blocks
    
```

Information about the used data blocks of file 13 in database 76 is displayed. 50 DS blocks are allocated, of which 49 are in use and 1 is unused. The total number of records is 179, with the record

length varying between a maximum of 1991 and a minimum of 260. The average record length is 997.47. The following lines give an overview of the number of blocks that are used up to a given level. The majority of the blocks (27) is used up to between 90% and 95%.

Example (for ADAM file)

```

adafin: file = 8
adafin: adam_ds = full
adafin: usage = ds

Database 30, File 8 (ADAM_FILE ) 11-OCT-2006 12:08:57

ADAM key = FF ADAM parameter = 5 ADAM_DS = FULL

DS - Blocks used for ADAM = 94
Total overflow blocks = 1, used = 1

Records: Number = 3863
          In ADAM area= 3860 , ovfl = 3
          Length: max = 9 , min = 9 , avg = 9.00

0%: ***** 10 blocks
5%: *** 4 blocks
10%: 0 blocks
15%: 0 blocks
20%: 0 blocks
25%: 0 blocks
30%: * 1 block
35%: 0 blocks
40%: 0 blocks
45%: 0 blocks
50%: * 1 block
55%: 0 blocks
60%: 0 blocks
65%: ***** 74 blocks
70%: 0 blocks
75%: 0 blocks
80%: 0 blocks
85%: 0 blocks
90%: 0 blocks
95%: *** 3 blocks
100%: * 2 blocks

```

Information about all data blocks of file 8, which is an ADAM file, is displayed. The ADAM parameter is set to 5. 94 blocks are used for the ADAM area, with 1 block reserved for overflow. The ADAM area contains 3860 records, and 3 records are in the overflow area.

14

ADAFRM (Format And Create A New Database)

▪ Functional Overview	168
▪ Procedure Flow	169
▪ Checkpoints	170
▪ Control Parameters	170
▪ Restart Considerations	174
▪ Control Statement Examples	174

This chapter describes the utility "ADAFRM".

Functional Overview

The utility ADAFRM creates the container files (ASSO, DATA, WORK) assigned to the database and establishes the database including the database system files. It can also be used to format the TEMP and SORT files.

For the raw device interface, the utility ADADEV can be used if placement control is required. Raw devices and files in the file system may be used for database container files.

You can create up to 255 ASSO and 255 DATA container files. You can add more containers later using ADADBMs ADD_CONTAINER function.

After ADAFRM creates the container files, it initializes the global Adabas blocks, inserts the 3 Adabas system files (checkpoint file, ET data file, security file) and allocates space for them. The checkpoint file is allocated 3000 records, the ET data file is allocated 3000 records and the security file is allocated 200 records.

Block sizes from 1 kilobyte to 32 kilobytes may be used for database container files.

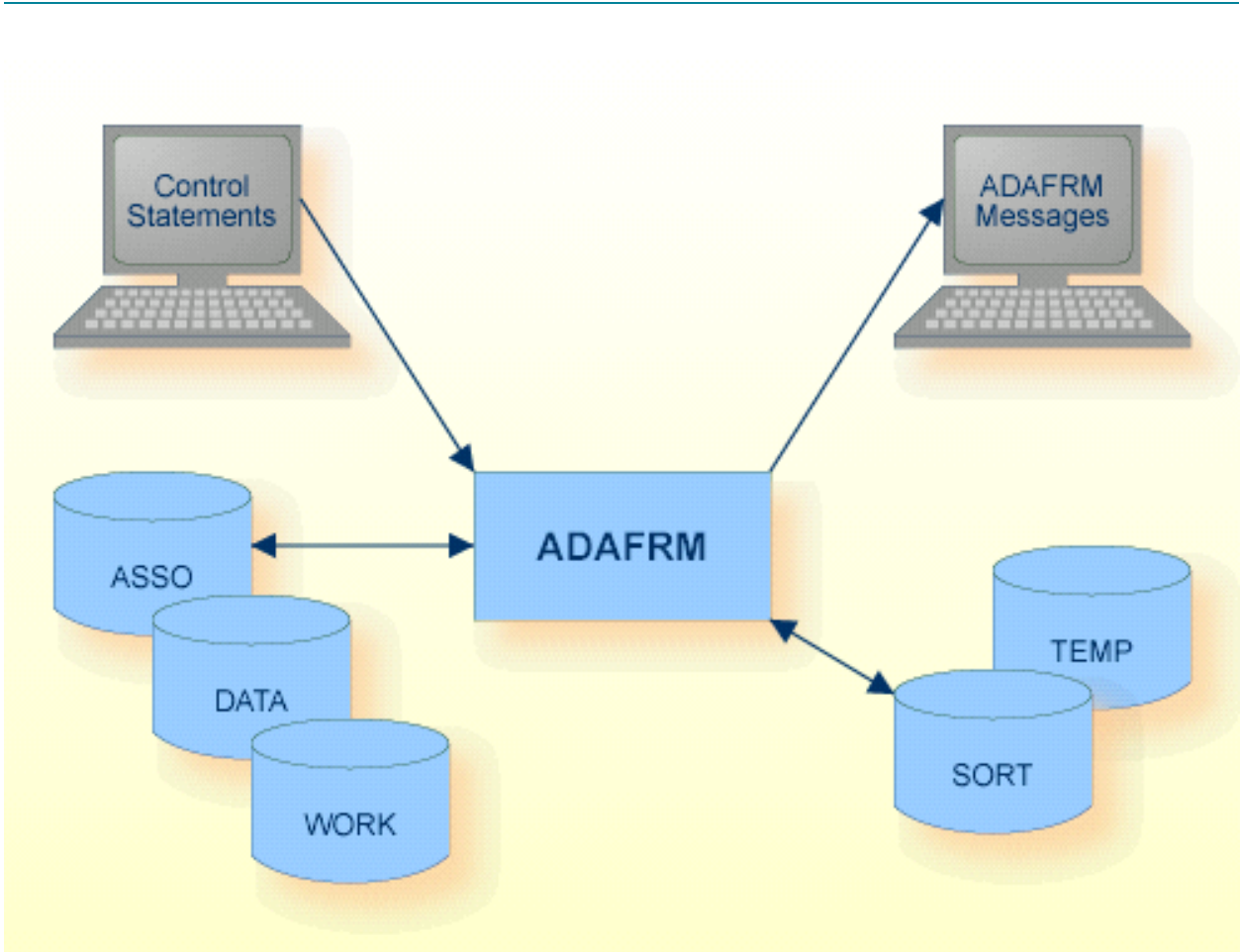
If you try to reformat a container file, the utility terminates with an error message. This ensures that the database is not accidentally overwritten.



Note: On OpenVMS, container files are allocated using the contiguous-best-try method. For this reason, you should ensure that your disk space is as defragmented as possible in order to avoid reduced performance.

This utility is a single function utility.

Procedure Flow



If a database is to be formatted:

Data Set	Environment Variable/ Logical Name	Storage Medium	Additional Information
Associator	ASSOx	Disk	
Data storage	DATAx	Disk	
Control statements	stdin/ SYS\$INPUT		Utilities Manual
ADAFRM messages	stdout/ SYS\$OUTPUT		Messages and Codes
Work	WORK1	Disk	

If a TEMP or SORT is to be formatted:

Data Set	Environment Variable/ Logical Name	Storage Medium	Additional Information
Sort storage	SORTx	Disk	
Control statements	stdin/ SYS\$INPUT		Utilities Manual
ADAFRM messages	stdout/ SYS\$OUTPUT		Messages and Codes
Temporary storage	TEMPx	Disk	

Checkpoints

The utility writes no checkpoints.

Control Parameters

The following control parameters are only used when establishing a new database:

```

D   ASSOBLOCKSIZE = (number[K] [,number[K]] ... )
M   ASSOSIZE = (number[B|M] [,number[B|M]]...)
D   DATABLOCKSIZE = (number[K] [,number[K]] ... )
M   DATASIZE = (number[B|M] [,number[B|M]]...)

DBID = number

D   NAME {=|:} string
M   SORTSIZE = (number[M] [,number[M]] ... )
D   SYSFILES = (number, number, number)
M   TEMPSIZE = (number[M] [,number[M]] ... )
D   WORKBLOCKSIZE = number[K]
M   WORKSIZE = number[M | B]
    
```

ASSOBLOCKSIZE

```
ASSOBLOCKSIZE = (number[K] [,number[K]] ... )
```

This parameter specifies the block sizes that are to be used for the Associator container file(s). The first block size corresponds to ASSO1, the second to ASSO2 etc.

If block sizes are not specified, the default of 4K will be used.

For ASSO1, only blocks sizes from 2K to 8K can be specified. For ASSO2 to ASSO_n, block sizes between 1K and 32K are permitted.



Note: The ASSOBLOCKSIZE parameter should be specified once for each ASSOSIZE that is specified, i.e. these parameters should be specified in pairs. If ASSOSIZE is specified more frequently than ASSOBLOCKSIZE, then the last specified block size will be used for the containers that do not have a block size specified. The default value will be used if ASSOBLOCKSIZE is not specified at all.

ASSOSIZE

```
ASSOSIZE = (number[M|B] [,number[M|B]]...) ↵
```

This parameter specifies the number of blocks or megabytes to be assigned to the Associator.

If the Associator is to be contained in more than one physical file, the size of each file must be specified.

If a 'B' is appended to the number, the size is in blocks, otherwise it is in megabytes.

DATABLOCKSIZE

```
DATABLOCKSIZE = (number[K] [,number[K]] ... )
```

This parameter specifies the block sizes that are to be used for the Data Storage container file(s). The first block size corresponds to DATA1, the second to DATA2 etc.

If block sizes are not specified, the default of 32K will be used.



Note: The DATABLOCKSIZE parameter should be specified once for each DATASIZE that is specified, i.e. these parameters should be specified in pairs. If DATASIZE is specified more frequently than DATABLOCKSIZE, then the last specified block size will be used for the containers that do not have a block size specified. The default value will be used if DATABLOCKSIZE is not specified at all.

DATASIZE

```
DATASIZE = (number[B|M] [,number[B|M]]...) ↵
```

This parameter specifies the number of blocks or megabytes to be assigned to the Data Storage. If the Data Storage is to be contained in more than one file, the size of each file must be specified. If a 'B' is appended to the number, the size is in blocks, otherwise it is in megabytes.

DBID

```
DBID = number
```

This parameter selects the database to be created.

The minimum value is 1 and the maximum value is 255.



Note: This parameter only needs to be set when formatting ASSO, DATA and WORK. It must not be entered when formatting only SORT or TEMP.

NAME

```
NAME {=|:} string
```

This parameter specifies the name to be assigned to the database. This name will appear in the title of the database status report produced by the report utility ADAREP. If you specify an equals sign, the value given for 'string' will be converted to upper case; if you specify a colon, no upper-case conversion is performed.

A maximum of 16 characters may be specified.

If this parameter is omitted, a default value of 'GENERAL-DATABASE' is assigned.

SORTSIZE

```
SORTSIZE = (number[M] [,number[M]] ... )
```

This parameter specifies the number of megabytes to be assigned to the SORT dataset.

If the SORT dataset consists more than one extent, the size of each extent must be specified. Up to 50 extents can be specified. The SORT dataset can be formatted independently.

SYSFILES

```
SYSFILES = (number, number, number)
```

This parameter specifies the file numbers to be reserved for the Adabas system files. These file numbers must not be used subsequently for user files.

The first value specifies the file number of the checkpoint file.

The second value specifies the file number of the security file.

The third value specifies the file number of the user data file.

The default setting is SYSFILES=(1, 2, 3).

TEMPSIZE

```
TEMPSIZE = (number [M] [,number[M]] ... )
```

This parameter defines the number of megabytes to be assigned to TEMPx.

If the TEMP dataset is to be contained in more than one physical file, the size of each file must be specified.

This component may be formatted independently.

WORKBLOCKSIZE

```
WORKBLOCKSIZE = number[K]
```

This parameter specifies the block size that is to be used for the WORK file.

If no block size is specified, the default of 16K will be used.

WORKSIZE

```
WORKSIZE = number [B|M]
```

This parameter defines the number of blocks or megabytes to be assigned to WORK1.

If a 'B' is appended to the number, the size is in blocks, otherwise it is in megabytes.

Restart Considerations

ADAFRM does not have a restart capability. An interrupted ADAFRM run must be restarted from the beginning. Associator, Data Storage and WORK must be formatted together.

The files created during an earlier run have to be deleted first.

Control Statement Examples

Example: Formatting a database

```
adafrm: dbid = 1, name = DATABASE_1
adafrm: assosize = (200M, 100M), assoblocksize = (2K, 4K)
adafrm: datasize = (500M, 500M, 2M), datablocksize = (4K, 16k)
adafrm: worksize = 50M, workblocksize = 16K
```

A new database is established with the number 1 and the name "DATABASE_1". Two ASSO container files are created: ASSO1 has a size of 200 megabytes and a blocksize of 2 kilobytes, and ASSO2 has a size of 100 megabytes and a blocksize of 4 kilobytes. There are three DATA containers. DATA1 and DATA3 have a blocksize of 4 kilobytes, DATA2 has a blocksize of 16 kilobytes. There is a single WORK container file with a block size of 16 kilobytes. The file numbers 1, 2 and 3 will be used for the 3 system files.

Example: Formatting SORT and TEMP

```
adafrm: sortsize = (10M,10M)
adafrm: tempsize = 10M
```

Explanation: Two container files, each 10 megabytes in length, are to be formatted as SORT1 and SORT2. A container file, 10 megabytes in length, is to be formatted as TEMP1.

15 ADAINV (Creating, Removing And Verifying Inverted Lists)

▪ Functional Overview	176
▪ Procedure Flow	177
▪ Checkpoints	178
▪ Control Parameters	179
▪ Restart Considerations	188
▪ Examples	189

This chapter describes the utility "ADAINV".

Functional Overview

The inverted list utility ADAINV creates, removes and verifies inverted lists for loaded files in a database. It does not require the Adabas nucleus to be active. The nucleus may, however, be active or shut down while ADAINV is running. The following functions are available:

- The INVERT function establishes new descriptors;
- The REINVERT function performs an implicit RELEASE and INVERT;
- The RELEASE function removes existing descriptors;
- The RESET_UQ function removes the unique status from descriptors;
- The SET_UQ function establishes a unique status for existing descriptors;
- The SUMMARY function displays the descriptor space summary for the specified descriptors and the required sizes to process these descriptors;
- The VERIFY function checks the integrity of inverted lists.

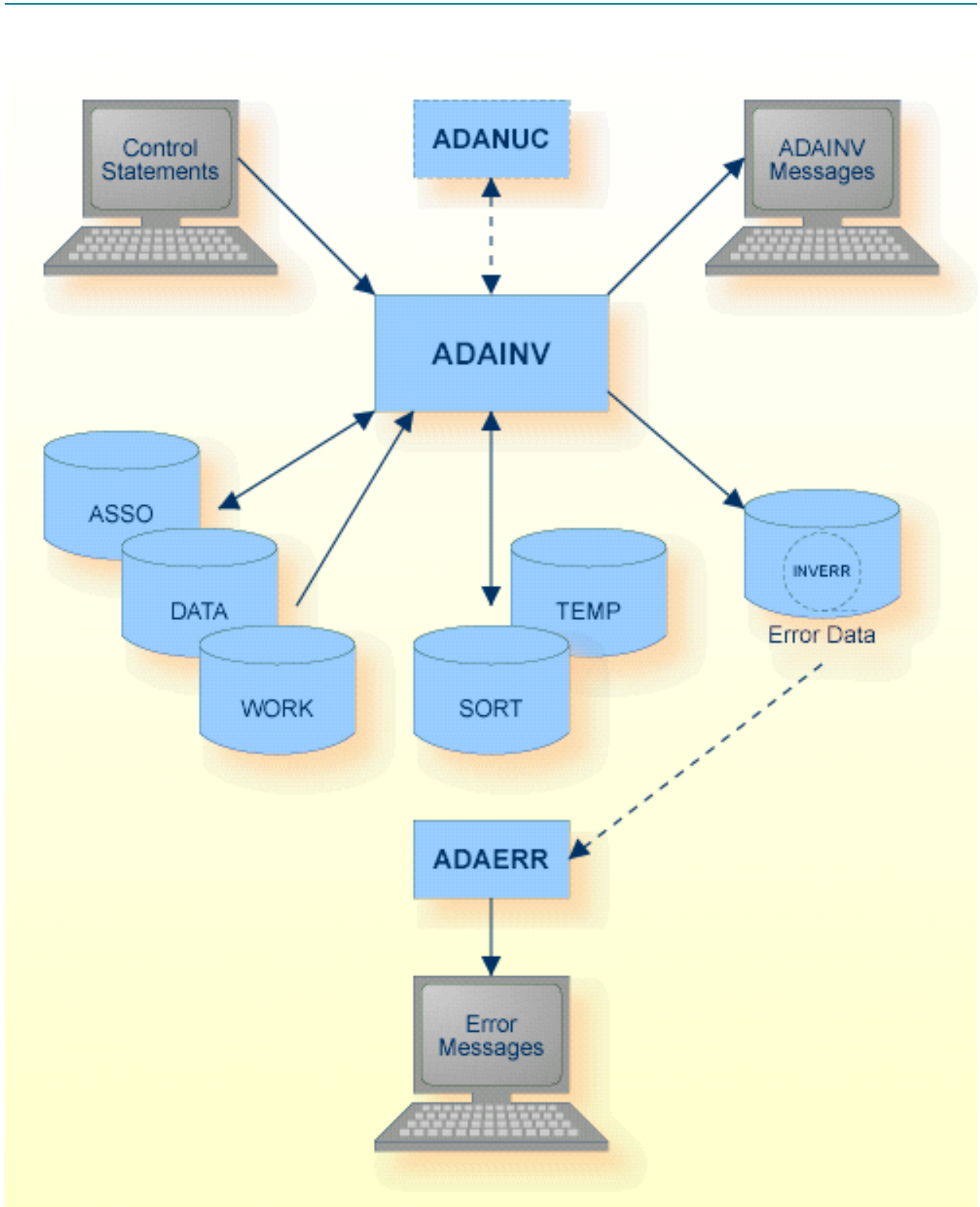
A LOB file can only be specified for the functions REINVERT, SUMMARY and VERIFY.

These functions are mutually exclusive and only one of them may be executed each time this utility is run.

If the utility writes records to the error file, it will exit with a non-zero status.


This utility is a single-function utility.

Procedure Flow



The sequential file INVERR can have multiple extents. For detailed information about sequential files with multiple extents, see *Administration, Using Utilities*.

Data Set	Environment Variable/ Logical Name	Storage Medium	Additional Information
Associator	ASSOx	Disk	
Data storage	DATAx	Disk	
Rejected data	INVERR	Disk, Tape (* see note)	output of ADAINV
Sort storage	SORTx TEMPLOCx	Disk	Administration Manual, temporary working space
Control statements	stdin/ SYS\$INPUT		Utilities Manual
ADAINV messages	stdout/ SYS\$OUTPUT		Messages and Codes
Temporary storage	TEMPx	Disk	
Work storage	WORK1	Disk	

 **Note:** (*) A named pipe can be used for this sequential file (not on OpenVMS, see Administration, Using Utilities for details).

In cases without an active nucleus and no pending AUTORESTART, the WORK may be used as TEMP by setting the environment variable/logical name TEMP1 to the path name or raw disk section of a WORK container.

Checkpoints

Checkpoints

The following table shows the nucleus requirements for each function and the checkpoints written:

Function	Nucleus must be active	Nucleus must NOT be active	Nucleus is NOT required	Checkpoint written	Nucleus operations allowed
INVERT			X	SYNP	R
REINVERT		X(* see note)	X	SYNP	
RELEASE			X	SYNP	R
RESET_UQ			X	SYNP	R
SET_UQ			X	SYNP	R
SUMMARY			X		W
VERIFY		X(* see note)	X	SYNX	R



Note: (*) When processing an Adabas system file.

R: read operations allowed for the processed file.

W: read und write operations allowed for the processed file.

Control Parameters

The following control parameters are available:

```

M  DBID = number

    INVERT = number,
        FIELDS {field_name [,UQ] [,TR] | derived_descriptor_definition | FDT},
        ... [END_OF_FIELDS]
        [,FDT]
D    [,LWP = number[K]]
D    [,UQ_CONFLICT = keyword]

D  [NO]LOWER_CASE_FIELD_NAMES

    REINVERT = number,
        {ALL_FIELDS | FIELDS {descriptor_name | FDT}, ... [END_OF_FIELDS]}
        [,FDT]
D    [,[NO]FORMAT]
D    [,LWP = number[K]]
D    [,UQ_CONFLICT = keyword]

    RELEASE = number,
        {ALL_FIELDS | FIELDS {descriptor_name | FDT}, ... [END_OF_FIELDS]}
        [,FDT]
D    [,[NO]FORMAT]

    RESET_UQ = number,
        {ALL_FIELDS | FIELDS {descriptor_name | FDT}, ... [END_OF_FIELDS]}
        [,FDT]

    SET_UQ = number,
        {ALL_FIELDS | FIELDS {descriptor_name | FDT}, ... [END_OF_FIELDS]}
        [,FDT]
D    [,UQ_CONFLICT = keyword]

    SUMMARY = number,
        {ALL_FIELDS | FIELDS
        {descriptor_name | derived_descriptor_definition | FDT},
        ... [END_OF_FIELDS]}
        [,FDT]
D    [,FULL]

```

```

    VERIFY = number,
            {ALL_FIELDS | FIELDS {descriptor_name | FDT}, ... [END_OF_FIELDS]}
D          [,ERRORS = number]
           [,FDT]
D          [,LWP = number[K]]

```

DBID

```
DBID = number
```

This parameter selects the database to be used.

INVERT

```

INVERT = number,
        FIELDS {field_name [,UQ] [,TR] | derived_descriptor_definition | FDT},
        ... [END_OF_FIELDS]
        [,FDT]
        [,LWP = number[K]]
        [,UQ_CONFLICT = keyword]

```

This function establishes new elementary, sub-, super-, hyper-, phonetic and collation descriptors at any time after a file has been initially loaded. `number` specifies the file containing the fields to be inverted. You are not allowed to specify the number of a LOB file.

FDT

This parameter displays the FDT of the selected file. This option may be specified before or within the field specification list.

FIELDS {field_name [,UQ] [,TR] | derived_descriptor_definition | FDT}, ... [END_OF_FIELDS]

This parameter specifies fields to be inverted. It can contain one or more

- field name,
- phonetic descriptor or
- sub-, super-, hyper- or collation descriptor

specifications, each starting on a separate line. See *Administration, FDT Record Structure* for valid specifications of field names, phonetic, sub-, super-, hyper- or collation descriptors.

The options UQ and TR are used to specify whether the field in question is a unique descriptor or whether index truncation will be performed. See *Administration, Definition Options* for further information about the UQ and TR options.



Note: Only fields for which the values are stored in the base file can be used as descriptors or parent fields of derived descriptors. For this reason, an invert function will be aborted if a field to be inverted or a parent field of a derived descriptor to be created has the LA or LB option and values are stored in the LOB file. LA and LB fields can be used as descriptors or parent fields of derived descriptors, but then all values are limited to 16 KB – 3, and the base record including these LA or LB field values must fit into one data block.

If the field definitions are terminated with the END_OF_FIELDS parameter, this parameter must be specified in upper case when the LOWER_CASE_FIELD_NAMES parameter is used. In addition, the FDT parameter must also be specified in upper case when the LOWER_CASE_FIELD_NAMES parameter is used.

LWP = number[K]

This parameter specifies the size of the Work Pool in bytes or in kilobytes (K) to be used for the sort while creating the inverted lists.

You can use the SUMMARY function to determine the required value for this parameter.

The minimum size is 0 bytes and the default size is 0 bytes.

UQ_CONFLICT = keyword

This parameter determines which action is to be taken when duplicate values are found for a unique descriptor. 'keyword' may take the values ABORT or RESET. If ABORT is specified, ADAINV terminates execution and returns an error status if duplicate UQ descriptor values are found. If RESET is specified, the UQ status of the descriptors in question is removed and processing continues.

The default is UQ_CONFLICT = ABORT.

[NO]LOWER_CASE_FIELD_NAMES

[NO]LOWER_CASE_FIELD_NAMES

If LOWER_CASE_FIELD_NAMES is specified, Adabas field names are not converted to upper case. If NOLOWER_CASE_FIELD_NAMES is specified, Adabas field names are converted to upper case. The default is NOLOWER_CASE_FIELD_NAMES.

This parameter must be specified before the FIELDS parameter.

REINVERT

```
REINVERT = number,  
          {ALL_FIELDS | FIELDS {descriptor_name | FDT}, ... [END_OF_FIELDS]}  
          [,FDT]  
          [, [NO]FORMAT]  
          [,LWP = number[K]]  
          [,UQ_CONFLICT = keyword]
```

This function performs an implicit RELEASE and INVERT. This reduces the probability of a typing error, especially for sub- and superdescriptors.



Note: The purpose of ADAINV REINVERT is to recreate a descriptor if the index tree becomes unbalanced as a result of a large number of updates, or if an index error occurred. Descriptors are always recreated with the same definition as before; if you want to change the definition of a descriptor, for example a superdescriptor, you must perform ADAINV RELEASE followed by ADAINV INVERT with the new descriptor definition.

ALL_FIELDS

This parameter specifies that all descriptors of the selected file are to be released/inverted.

FDT

This parameter displays the FDT of the selected file. This option may be specified before or within the fields specification list.

FIELDS {descriptor_name | FDT}, ... [END_OF_FIELDS]

This parameter specifies the descriptors to be released/reinverted. It can be followed by one or more field names, each starting on a separate line. See *Administration, FDT Record Structure* for a description of valid field name specifications.

If the field definitions are terminated with the END_OF_FIELDS parameter, this parameter must be specified in upper case when the LOWER_CASE_FIELD_NAMES parameter is used. In addition, the FDT parameter must also be specified in upper case when the LOWER_CASE_FIELD_NAMES parameter is used.

[NO]FORMAT

If a descriptor is released or reinverted, the new index created is generally smaller than the old index and requires less disk space. The FORMAT option can be used to format the blocks that are no longer used by the index but which are still allocated to the file.

The default is NOFORMAT.

LWP = number[K]

This parameter specifies the size of the Work Pool to be used for the sort. This is an internal sort to recover lost index blocks when rebuilding the upper index.

You can use the SUMMARY function to determine the required value for this parameter.

The minimum size is 0 bytes and the default size is 0 bytes.

UQ_CONFLICT = keyword

This parameter determines which action is to be taken when duplicate values are found for a unique descriptor. 'keyword' may take the values ABORT or RESET. If ABORT is specified, ADAINV terminates execution and returns an error status if duplicate UQ descriptor values are found. If RESET is specified, the UQ status of the descriptors in question is removed and processing continues.

The default is UQ_CONFLICT = ABORT.

RELEASE

```
RELEASE = number,
         {ALL_FIELDS | FIELDS {descriptor_name | FDT}, ... [END_OF_FIELDS]}
         [, FDT]
         [, [NO]FORMAT]
```

This function removes elementary, sub-, super-, hyper-, phonetic and collation descriptors from the file specified by 'number'. You are not allowed to specify the number of a LOB file.

ALL_FIELDS

This parameter specifies that all descriptors of the selected file are to be released.

FDT

This parameter displays the FDT of the selected file. This option may be specified before or within the fields specification list.

FIELDS {descriptor_name | FDT}, ... [END_OF_FIELDS]

This parameter specifies the descriptors to be released. It can be followed by one or more field names, each starting on a separate line. See *Administration, FDT Record Structure* for a description of valid field name specifications.

If the field definitions are terminated with the END_OF_FIELDS parameter, this parameter must be specified in upper case when the LOWER_CASE_FIELD_NAMES parameter is used. In addition, the FDT parameter must also be specified in upper case when the LOWER_CASE_FIELD_NAMES parameter is used.

[NO]FORMAT

If a descriptor is released or reinverted, the new index created is generally smaller than the old index and requires less disk space. The FORMAT option can be used to format the blocks that are no longer used by the index but which are still allocated to the file.

The default is NOFORMAT.

RESET_UQ

```
RESET_UQ = number,  
          {ALL_FIELDS | FIELDS {descriptor_name | FDT}, ... [END_OF_FIELDS]}  
          [,FDT]
```

This function removes the unique status from elementary, sub-, hyper-, super- and collation descriptors defined in the file specified by 'number'. You are not allowed to specify the number of a LOB file.

ALL_FIELDS

This parameter specifies that the unique status is to be removed from all unique descriptors in the specified file.

FDT

This parameter displays the Field Definition Table (FDT) of the selected file. This option may be specified before or within the fields specification list.

FIELDS {descriptor_name | FDT}, ... [END_OF_FIELDS]

This parameter specifies the descriptors that are to have unique status removed. It can be followed by one or more field names, each starting on a separate line. See *Administration, FDT Record Structure* for a description of valid field name specifications.

If the field definitions are terminated with the END_OF_FIELDS parameter, this parameter must be specified in upper case when the LOWER_CASE_FIELD_NAMES parameter is used. In addition, the FDT parameter must also be specified in upper case when the LOWER_CASE_FIELD_NAMES parameter is used.

SET_UQ

```
SET_UQ = number,
        {ALL_FIELDS | FIELDS {descriptor_name | FDT}, ... [END_OF_FIELDS]}
        [,FDT]
        [,UQ_CONFLICT = keyword]
```

This function establishes the unique status for elementary, sub-, hyper-, super- and collation descriptors defined in the file specified by 'number'. You are not allowed to specify the number of a LOB file.

ALL_FIELDS

This parameter specifies that the unique status is to be established for all elementary, sub-, hyper-, super- and collation descriptors defined in the specified file.

FDT

This parameter displays the FDT of the selected file. This option may be specified before or within the fields specification list.

FIELDS {descriptor_name | FDT}, ... [END_OF_FIELDS]

This parameter specifies the descriptors for which the unique status is to be established. It can be followed by one or more field names, each starting on a separate line. See *Administration, FDT Record Structure* for a description of valid field name specifications.

If the field definitions are terminated with the END_OF_FIELDS parameter, this parameter must be specified in upper case when the LOWER_CASE_FIELD_NAMES parameter is used. In addition, the FDT parameter must also be specified in upper case when the LOWER_CASE_FIELD_NAMES parameter is used.

UQ_CONFLICT = keyword

This parameter determines which action is to be taken when duplicate values are found for a unique descriptor. 'keyword' may take the values ABORT or RESET. If ABORT is specified, ADAINV terminates execution and returns an error status if duplicate descriptor values are found. If RESET is specified, the UQ status of the descriptors in question is not established and processing continues.

The default is UQ_CONFLICT = ABORT

SUMMARY

```
SUMMARY = number,  
          {ALL_FIELDS | FIELDS  
          {descriptor_name | derived_descriptor_definition | FDT},  
          ... [END_OF_FIELDS]}  
          [,FDT]  
          [,FULL]
```

This function displays the descriptor space summary (DSS) for the specified descriptors and the required sizes to process the descriptors.



Note: Processing the exact size would be too complicated. It may be that sizes a little smaller than those displayed are sufficient. If the file is updated during or after the SUMMARY function, the displayed values might also be too small.

ALL_FIELDS

This parameter specifies that all descriptors of the selected files are to be checked.

FDT

This parameter displays the FDT of the selected file. This option may be specified before or within the fields specification list.

FIELDS {descriptor_name | derived_descriptor_definition | FDT}, ... [END_OF_FIELDS]

This parameter specifies the descriptors for which the unique status is to be established. It can be followed by one or more field names, phonetic descriptors, subdescriptors, superdescriptors, hyperdescriptors or collation descriptors, each starting on a separate line. You can specify fields that are descriptors or fields that are not descriptors. See *Administration, FDT Record Structure* for a description of valid field name specifications.

If the field definitions are terminated with the END_OF_FIELDS parameter, this parameter must be specified in upper case when the LOWER_CASE_FIELD_NAMES parameter is used. In addition, the FDT parameter must also be specified in upper case when the LOWER_CASE_FIELD_NAMES parameter is used.

FULL

If this is specified, each descriptor is displayed along with the sizes that are required for the descriptor. This can be helpful if not all of the specified fields are to be processed.

VERIFY

```
VERIFY = number,
        {ALL_FIELDS | FIELDS {descriptor_name | FDT}, ... [END_OF_FIELDS]}
        [,ERRORS = number]
        [,FDT]
        [,LWP = number[K]]
```

This function checks the integrity of inverted lists of the file specified by 'number'.

ALL_FIELDS

This parameter specifies that all descriptors of the selected file are to be checked.

ERRORS = number

This parameter specifies the number of errors that have to be reported in order to terminate the verification of a descriptor.

The default is 20.

FDT

This parameter displays the FDT of the selected file. This option may be specified before or within the fields specification list.

FIELDS {descriptor_name | FDT}, ... [END_OF_FIELDS]

This parameter specifies the descriptor fields to be verified. It can be followed by one or more field names, each starting on a separate line. See *Administration, FDT Record Structure* for a description of valid field name specifications.

If the field definitions are terminated with the END_OF_FIELDS parameter, this parameter must be specified in upper case when the LOWER_CASE_FIELD_NAMES parameter is used. In addition, the FDT parameter must also be specified in upper case when the LOWER_CASE_FIELD_NAMES parameter is used.

LWP = number[K]

This parameter specifies the size of the Work Pool to be used for the sort while verifying the inverted lists.

You can use the SUMMARY function to determine the required value for this parameter.

The minimum size is 0 bytes and the default size is 0 bytes.

Restart Considerations

ADAINV has no restart capability. However, it may or may not be possible to re-start an abnormally terminated ADAINV from the beginning.

If ADAINV terminates abnormally, it can usually be restarted from the beginning. However, if ADAINV has modified the index, the following points have to be considered:

- The function REINVERT ... FIELDS is the same as the function RELEASE ... FIELDS followed by the function INVERT ... FIELDS. So if ADAINV has aborted in the INVERT phase, perform the function INVERT ... FIELDS to restart the operation.
- If ADAINV is performed offline, there is a very small amount of time where a few records that together form a logical unit are written to disk. If ADAINV terminates after the first of these records has been written and before the last has been written, ADAINV cannot be restarted. In this case, the function REINVERT ... ALL_FIELDS is required. This cannot happen if ADAINV is performed online.
- If ADAINV terminates abnormally, it can happen that some index blocks are lost. These index blocks can only be recovered by the function REINVERT ... ALL_FIELDS or by using the utility ADAORD or by using the utilities ADAULD and ADAMUP.

Examples

Example 1

```
adainv: dbid=1
adainv: invert=10, fields
adainv: HO
```

The elementary field HO in file 10 of database 1 is inverted.

Example 2

```
adainv: dbid=1
adainv: invert=10
adainv: lwp=600k
adainv: fields
adainv: ph=phon(na)
adainv: sp=na(1,3),yy(1,2),uq
adainv: bb,uq
```

Three new descriptors are established for file 10 in database 1. PH is a phonetic descriptor based on the field NA. SP is a unique superdescriptor derived from bytes 1 to 3 of field NA and bytes 1 to 2 of field YY. The elementary field BB is changed to descriptor status and the unique flag is set. The size of the work pool to be used for the sort is increased to 600 K.

Example 3

```
adainv: dbid=1
adainv: release=10
adainv: fields
adainv: ho
adainv: ph
```

The two descriptors HO and PH from the examples above are released.

Example 4

```
adainv: dbid = 1, verify = 10
adainv: errors = 5
adainv: fields
adainv: sp
adainv: na
adainv: end_of_fields
```

The descriptors SP and NA are verified. The descriptor value table entries generated for descriptor NA are checked against the decompressed values of this field. Verification is terminated if more than five errors are reported for each descriptor.

Example 5

```
adainv: dbid = 1, reinvert = 10
adainv: fields
adainv: na
```

The descriptor NA in file 10 of database 1 is to be reinverted (this may be necessary if errors are reported in example 4).

Example 6

```
adainv: db=12
adainv: reinvert=9
adainv: all_fields
```

The complete index is recreated for file 9 in database 12.

The following output is produced:

```
%ADAINV-I-FILE, file 9, EMPLOYEES

%ADAINV-I-UIUPD, upper index being modified

%ADAINV-I-SORTDESC, sorting descriptor KA
%ADAINV-I-LOADDESC, loading descriptor KA

%ADAINV-I-SORTDESC, sorting descriptor S3
%ADAINV-I-LOADDESC, loading descriptor S3

%ADAINV-I-SORTDESC, sorting descriptor S2
%ADAINV-I-LOADDESC, loading descriptor S2

%ADAINV-I-SORTDESC, sorting descriptor PA
%ADAINV-I-LOADDESC, loading descriptor PA

%ADAINV-I-SORTDESC, sorting descriptor FB
%ADAINV-I-LOADDESC, loading descriptor FB

%ADAINV-I-SORTDESC, sorting descriptor AA
%ADAINV-I-LOADDESC, loading descriptor AA

%ADAINV-I-SORTDESC, sorting descriptor BC
%ADAINV-I-LOADDESC, loading descriptor BC

%ADAINV-I-SORTDESC, sorting descriptor CN
%ADAINV-I-LOADDESC, loading descriptor CN

%ADAINV-I-SORTDESC, sorting descriptor JA
%ADAINV-I-LOADDESC, loading descriptor JA

%ADAINV-I-SORTDESC, sorting descriptor H1
%ADAINV-I-LOADDESC, loading descriptor H1

%ADAINV-I-SORTDESC, sorting descriptor EA
%ADAINV-I-LOADDESC, loading descriptor EA

%ADAINV-I-SORTDESC, sorting descriptor LC
%ADAINV-I-LOADDESC, loading descriptor LC

%ADAINV-I-SORTDESC, sorting descriptor S1
%ADAINV-I-LOADDESC, loading descriptor S1

%ADAINV-I-SORTDESC, sorting descriptor AC
%ADAINV-I-LOADDESC, loading descriptor AC

%ADAINV-I-NULLDISC, no values for descriptor IJ
%ADAINV-I-LOADDESC, loading descriptor IJ

%ADAINV-I-NULLDISC, no values for descriptor IB
%ADAINV-I-LOADDESC, loading descriptor IB
```

```
%ADAINV-I-NULDESC, no values for descriptor FI
%ADAINV-I-LOADDESC, loading descriptor FI

%ADAINV-I-UIUPD, upper index being modified
%ADAINV-I-DSPASSES, data storage passes : 17
%ADAINV-I-REMOVED, dataset SORT1, file C:\Program Files\Software AG\Adabas/db012
\SORT01_3664.012 removed
%ADAINV-I-IOCNT,      1 IOs on dataset SORT
%ADAINV-I-IOCNT,     85 IOs on dataset DATA
%ADAINV-I-IOCNT,     49 IOs on dataset ASSO
```



Notes:

1. The message NULDESC indicates that no descriptor values exist for this descriptor. This may happen for fields defined with option NU or NC if the field contains the null value/SQL null values for all records.
2. The message DSPASSES shows how often the data records of the file were read. In this case the number of data storage passes is 17, i.e. the data records were reread for each descriptor, because no TEMP container was defined where descriptor values can be saved. The number of data storage passes can be reduced by defining a TEMP container. This is recommended in particular for large files, because it reduces the number of required I/O operations significantly. The ADAINV parameter SUMMARY can be used to find out which size is useful for the TEMP container.
3. The message REMOVED shows that a temporary SORT container created by ADAINV was deleted. You can also use a persistent SORT container, which is not created and deleted by ADAINV (see [ADAFRM](#) for further details).

Example 7

```
adainv: dbid = 1, set_uq=10
adainv: fields
adainv: na
adainv: end_of_fields
adainv: uq_conflict=reset
```

The unique status is to be established for the descriptor NA in file 10 of database 1. If there is more than one ISN per descriptor value, the conflicting ISNs are written to the error log and the unique status is removed.

Example 8

```
adainv: dbid = 1, reset_uq=10
adainv: fields
adainv: sp
```

The unique status is to be removed from the descriptor SP in file 10 of database 1.

Example 9

```
adainv: db=33
adainv: summary=112
adainv: fields
adainv: ab
adainv: ae
adainv: s1=ap(1,1),aq(1,1),ar(1,1)
adainv: s2=ac(1,3),ad(1,8),ae(1,9)
adainv: s3=ao(2,3)
```

This produces the following output:

```
Descriptor summary:
```

```
=====
```

```
Descriptor AB :          1,194,469 bytes,          581,209 occ
Descriptor AE :          3,605,545 bytes,          538,769 occ
Descriptor S1 :          1,566,501 bytes,          581,209 occ
Descriptor S2 :          1,520,169 bytes,           72,389 occ
Descriptor S3 :          1,340,949 bytes,          446,983 occ
```

```
Required sizes to process these descriptors:
```

```
=====
```

```
  - SORTSIZE (LWP=          0 KB)          =          8 MB
  - LWP for incore sort                    =        13,230 KB
  - TEMPSIZE (1 pass)                      =          24 MB
  - TEMPSIZE (2 passes)                   =          13 MB
  - TEMPSIZE (recommended minimum size) =          5 MB
```

```
%ADAINV-I-IOCNT,      1710 IOs on dataset DATA
%ADAINV-I-IOCNT,         3 IOs on dataset ASSO
%ADAINV-I-TERMINATED,  24-NOV-2006 14:15:06, elapsed time: 00:04:03
```


16 ADAMON (Monitoring The Database Nucleus)

▪ Functional Overview	196
▪ Procedure Flow	197
▪ Checkpoints	197
▪ Control Parameters	198

This chapter describes the utility "ADAMON".



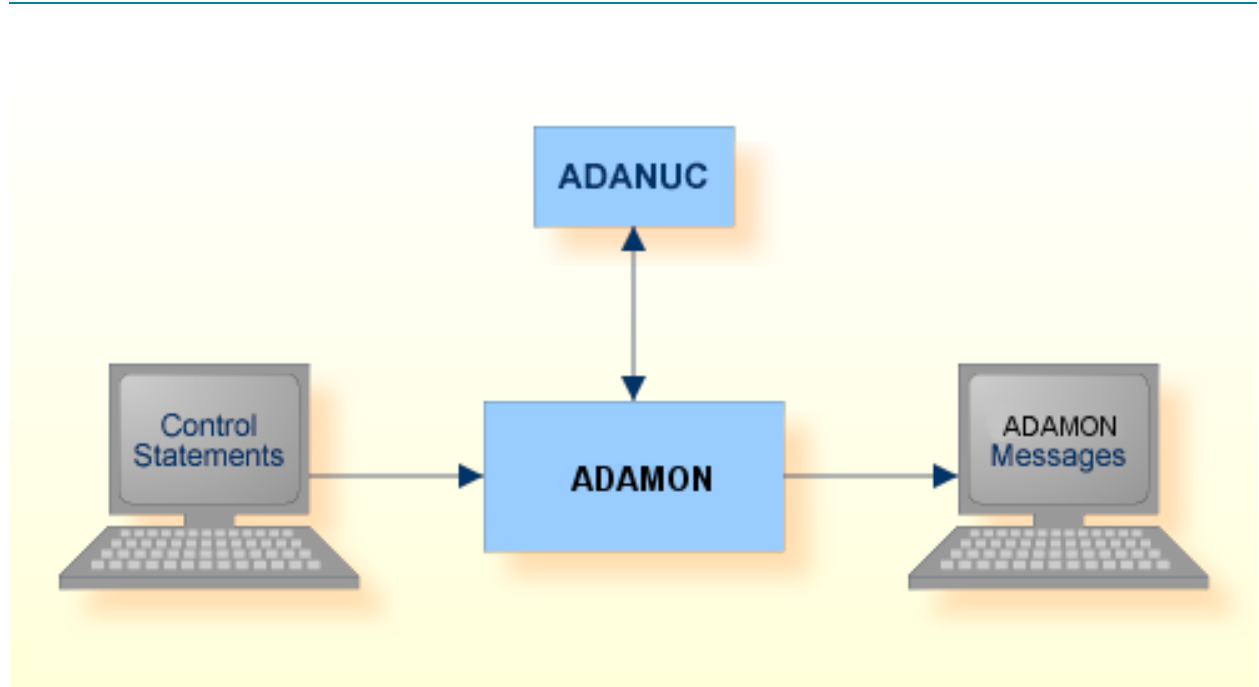
Note: This utility is not available on Windows platforms.

Functional Overview

The ADAMON utility is used to monitor an Adabas session with the aim of collecting performance data. The type of information collected is determined by the setting of the DISPLAY parameter; the information is usually displayed on a "per second" basis. The information collected can be presented as a set of numbers or as a basic graphical output. An ADAMON session is terminated by typing CTRL/C, or when the value specified for the LOOP parameter has been reached - then a statistical summary of the monitored session is displayed.

This utility is a multi-function utility.

Procedure Flow



Data Set	Environment Variable/ Logical Name	Storage Medium	Additional Information
Control statements	stdin/ SYS\$INPUT		Utilities Manual
ADAMON messages	stdout/ SYS\$OUTPUT		Messages and Codes

Checkpoints

The utility writes no checkpoints.

Control Parameters

The following control parameters are available:

```
D   [NO]DATETIME
M   DBID = number
D   DISPLAY = keyword
D   [NO]GRAPHICAL
D   INTERVAL = number
D   LOOPS = number
```

DATETIME

```
[NO]DATETIME
```

If this parameter is set to **DATETIME**, each monitoring line with non-graphical output will be preceded by the current date and time. The default is **NODATETIME**.

DBID

```
DBID = number
```

This parameter selects the database to be used. The database must be active for all functions with the exception of **DISPLAY = BACKUP**.

DISPLAY

```
DISPLAY = keyword
```

This parameter displays database information in accordance with the keyword specified. The display is refreshed at intervals specified by the parameter **INTERVAL** (default of 3 seconds). Please refer to the **DISPLAY** examples in the section **ADAOPR** for explanations of the information displayed.

The following keywords are available:

Keyword	Meaning
ACTIVITY	Displays the throughput of a database, for example the number of commands per second. This keyword is the default if the nucleus is active.
BACKUP	Displays a graph which monitors the execution of an ADABCK DUMP or RESTORE function. The values displayed are normalized to full block sizes, which can be different from the real backup/restore space because of 'used size compression'. The output is always graphically oriented. This keyword can also be used if the nucleus is not active. This keyword is the default if the nucleus is not active.
HIGH_WATER	Displays some important highwater values. The output is always graphically oriented. The bold line gives the current value in percent, the dashed line shows the highwater value. If there is only a bold line, the current and the high water values are identical. The 'Write Limit' line shows the number of modified blocks in percent until the flush limit is reached - at 100%, usually a buffer flush will be started. The number given within the line shows the modified space in bytes. The 'WP1 Flush' line shows the number of modified WORK part 1 blocks from the most recent buffer flush record - if the 100% point is reached, a buffer flush will be started. The 'Hit rate' lines show the overall hit rate (dashed line) and the current rate that occurred during the measured interval (bold line). The 'ASSO' and 'DATA' lines show the ratio between used and total allocated container space. The 'PLOG' line shows the ratio between the used and allocated space for the protection log. The numbers in these data set lines are either given in KB, MB or GB units. Note that the PLOG on file system always shows 100%
INDEX	Displays some counters and exceptions that occurred during index update (for internal reasons).
IO	Displays the number of physical I/Os of the specific Adabas container files, per second. For each container type (ASSO, DATA) only the first 10 extents can be displayed. I/Os to upper extents are collected in ASSOx or DATAx.

Furthermore, if an exceptional situation is detected during collection of the data, additional information is displayed on the screen. In the non-graphical mode it is displayed in the final column, in the graphical mode, the status is shown in the base line. The following status information can be detected:

BF_ACTIVE

A buffer flush is in progress.

SPACE_WAIT

Threads waiting for work pool space on complex commands.

ET_SYNC

The nucleus is in ET_SYNC mode, which means no new transactions will be started.

HYX

The nucleus is executing a hyperexit.

UEX

The nucleus is executing a user exit.

LARGE_DWP

The internal work pool is so large that it extends into the buffer pool.

SHUTDOWN-P

Nucleus shutdown in progress.

SHUTDOWN-C

Nucleus shutdown completed.

CRASHED

Nucleus abnormally terminated.

If an AUTORESTART is executing, ADAMON can monitor and display the phase number (1, 2, 3 or 4) and the number of processed blocks. Usually, phase 3 takes the most time, and the percentage of processed blocks is displayed. This is done independently of the selected function. When the AUTORESTART completes, it PT_RETs to the function requested.

GRAPHICAL

```
[NO]GRAPHICAL
```

Setting this option to GRAPHICAL switches the output to the graphical format. For the display functions BACKUP and HIGH_WATER, only the graphical format is supported. The default is NOGRAPHICAL.

INTERVAL

```
INTERVAL = number
```

This parameter specifies the data-collection sampling interval in seconds.

The default interval is 3 seconds.

LOOPS

```
LOOPS = number
```

This parameter limits the number of data collection loops.

By default, ADAMON loops continuously. Data collection can be terminated with CTRL/C.

Examples

Example 1:

```
adamon dbid=1
```

Display the activity of database 1.

Example 2:

```
adamon dbid=2  
adamon display=high_water
```

Display the high-water marks of database 2.

Example 3:

```
adamon dbid=3  
adamon display=io graphical
```

Display the I/O activity of database 3 with graphical output.

Example 4:

```
adamon dbid=4  
adamon interval=10
```

Display the activity of database 4, and refresh the content of the screen every 10 seconds.

17 ADAMUP (Mass Add And Delete)

▪ Functional Overview	204
▪ Procedure Flow	205
▪ Checkpoints	208
▪ Control Parameters	209
▪ Restart Considerations	215
▪ SORT Data Set Placement	215
▪ TEMP Data Set Placement	215
▪ Examples	216

This chapter describes the utility "ADAMUP".

Functional Overview

The mass update utility ADAMUP adds records to, or deletes records from a file in a database. It does not require the Adabas nucleus to be active.

The output files produced by the compression utility ADACMP or the unload utility ADAULD may be used as input for a mass add.



Note: The ADAMUP ADD function can process MUPDTA/MUPDVT files created with earlier Adabas versions, but not MUPDTA/MUPDVT files created with later Adabas versions.

Input files produced by ADACMP or ADAULD with the SINGLE_FILE option or from a previous run of ADAMUP using the DELETE function with the LOG option can also be used.

Input files produced without descriptor value tables (SHORT option in ADAULD or LOG=SHORT option in ADAMUP) can be processed if the database file to be processed does not contain any descriptors.

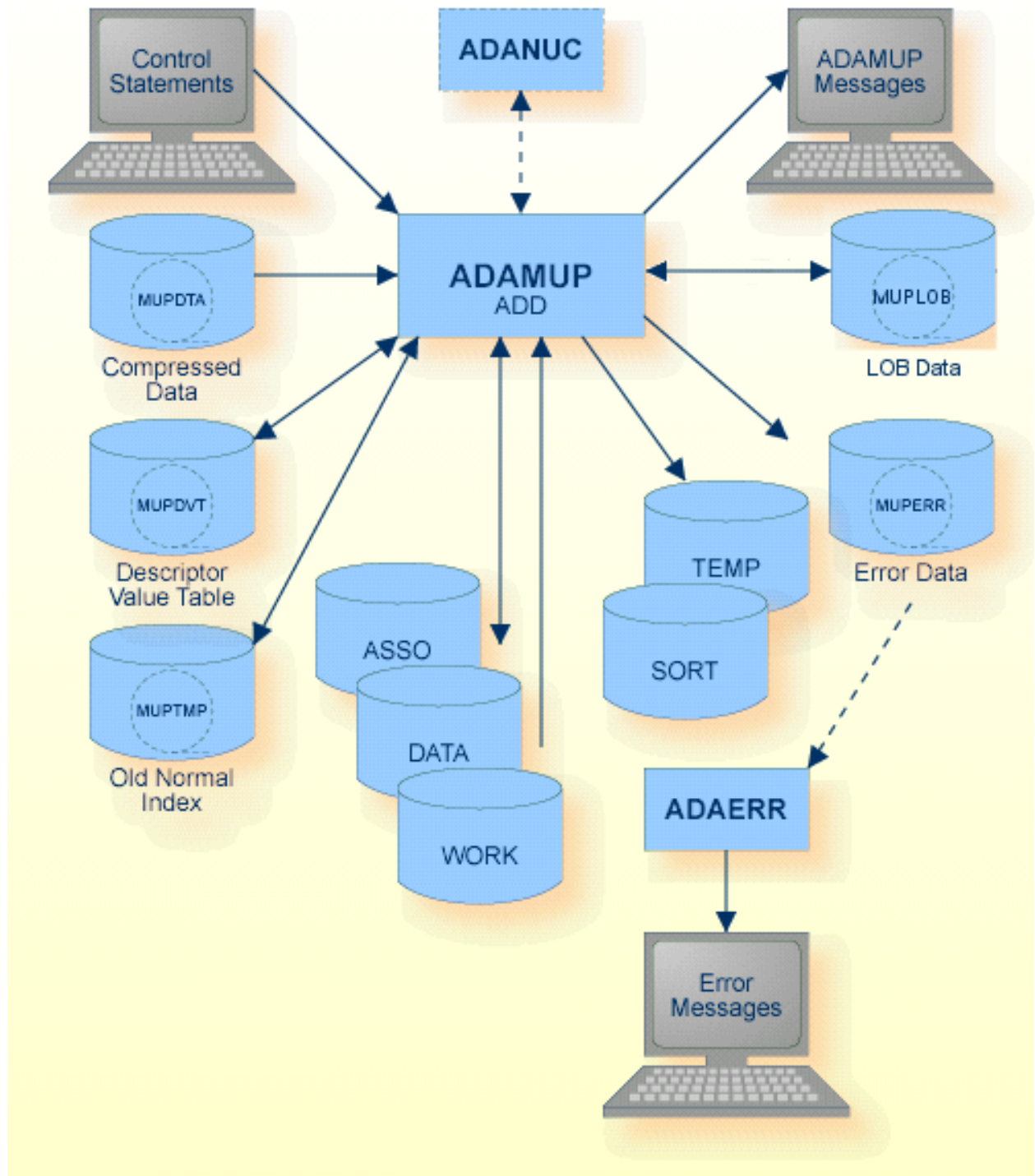
The input for the DELETE function is provided in an input file. Each record contains one or more ISNs or ISN ranges.

Records may be both added to and deleted from a database file during a single run of ADAMUP.

If the utility writes records to the error file, it will exit with a non-zero status.

This utility is a single-function utility.

Procedure Flow



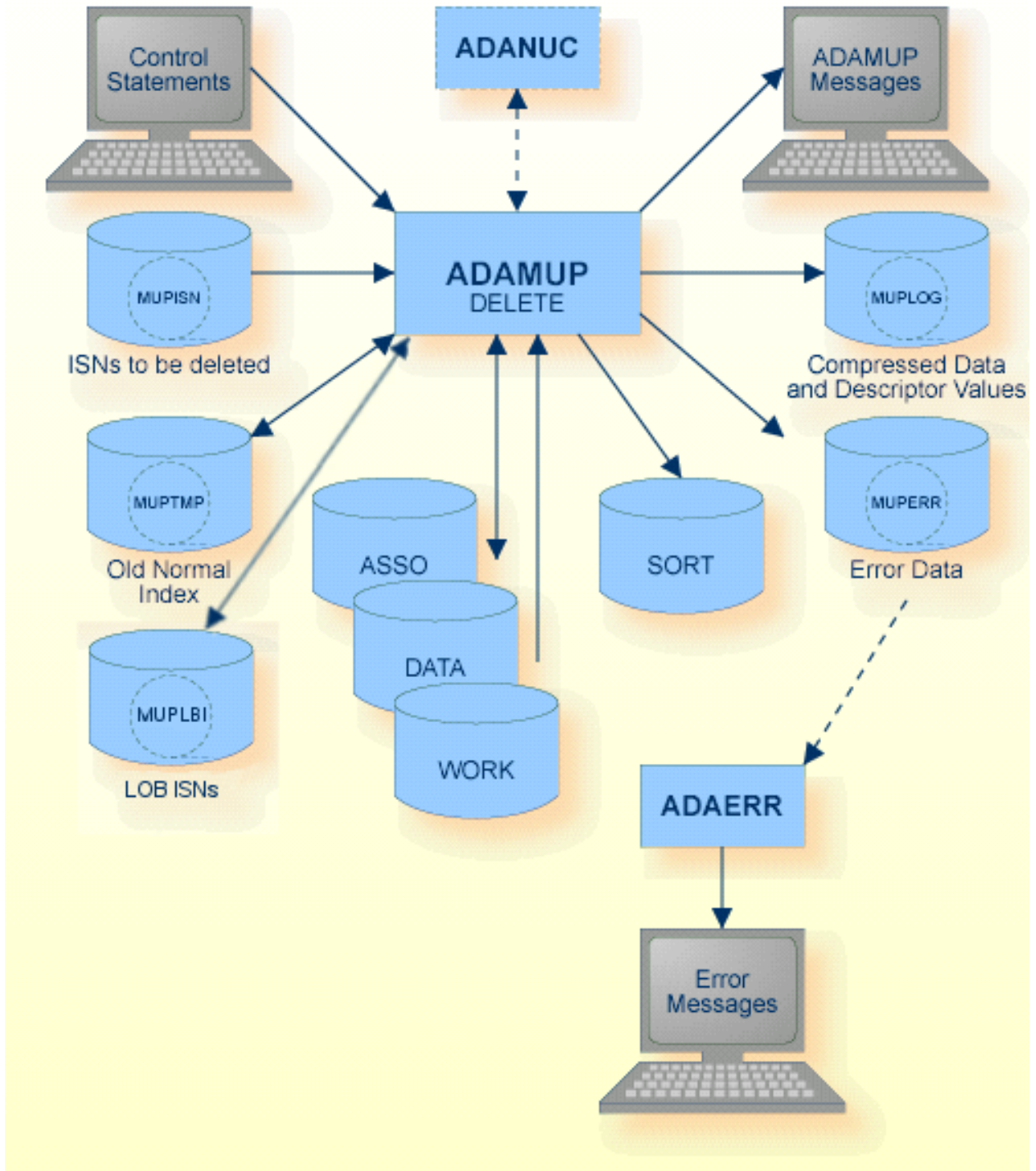
The sequential files MUPDTA, MUPDVT, MUPTMP, MUPLOB and MUPERR can have multiple extents. For detailed information about sequential files with multiple extents, see *Administration, Using Utilities*.

Data Set	Environment Variable/ Logical Name	Storage Medium	Additional Information
Associator	ASSOx	Disk	
Data storage	DATAx	Disk	
Compressed input data	MUPDTA	Tape, Disk	
Descriptor values	MUPDVT	Tape, Disk	
Rejected data	MUPERR	Disk, Tape (* see note)	
LOB data	MUPLOB	Disk, Tape	Temporary working space, will be deleted again when ADAMUP terminates
Normal index	MUPTMP	Disk, Tape	Temporary working space, will be deleted again when ADAMUP terminates
Sort storage	SORTx TEMPLOCx	Disk	Administration Manual, temporary working space
Control statements	stdin/ SYS\$INPUT		Utilities Manual
ADAMUP messages	stdout/ SYS\$OUTPUT		Messages and Codes
Temporary storage	TEMPx	Disk	
Work	WORK1	Disk	



Note: (*) A named pipe can be used for this sequential file (not on OpenVMS, see *Administration, Using Utilities* for details).

In cases without an active nucleus and no pending AUTORESTART, the WORK may be used as TEMP by setting the environment variable/logical name TEMP1 to the same value as WORK1.



The sequential files MUPTMP, MUPLBI, MUPLOG and MUPERR can have multiple extents. For detailed information about sequential files with multiple extents, see *Administration, Using Utilities*.

Data Set	Environment Variable/ Logical Name	Storage Medium	Additional Information
Associator	ASSOx	Disk	
Data storage	DATAx	Disk	
Rejected data	MUPERR	Disk, Tape (* see note)	
ISNs to be deleted	MUPISN	Disk, Tape	
LOB ISNs	MUPLBI	Disk, Tape	Temporary working space, will be deleted again when ADAMUP terminates
Compressed data	MUPLOG	Disk, Tape	
Normal index	MUPTMP	Disk, Tape	Temporary working space, will be deleted again when ADAMUP terminates
Sort storage	SORTx TEMPLOCx	Disk	Administration Manual, temporary working space
Control statements	stdin/ SYS\$INPUT		Utilities Manual
ADAMUP messages	stdout/ SYS\$OUTPUT		Messages and Codes
Work	WORK1	Disk	



Note: (*) A named pipe can be used for this sequential file (not on OpenVMS, see *Administration, Using Utilities* for details).

Checkpoints

The following table shows the nucleus requirements for each function and the checkpoints written:

Function	Nucleus must be active	Nucleus must NOT be active	Nucleus is NOT required	Checkpoint written
FDT			X	-
UPDATE	X(* see note 1)	X(* see note 2)	X(* see note 3)	SYNP
SUMMARY			X	-



Notes:

1. When deleting records in a file with LOB data.
2. When updating an Adabas system file.
3. Except when deleting records in a file with LOB data.

Control Parameters

The following control parameters are available:

```
M   DBID = number

    FDT

    SUMMARY

    UPDATE = number [,FDT]
            [ADD [,add_keywords]]
            [DELETE [,delete_keywords]]
D     [, [NO]FORMAT]
D     [LWP = number[K]]
```

DBID

```
DBID = number
```

This parameter selects the database to be used.

FDT

```
FDT
```

This parameter displays the Field Definition Table (FDT) of the selected file in the database. If records are to be added to a file, the FDT of the sequential input file containing these records can also be displayed. This parameter may also be used in an ADD/DELETE specification.

Depending on the context in which the FDT parameter is used, the Field Definition Table contained in the sequential input file MUPDTA and/or the Field Definition Table contained in the selected database file is displayed.

SUMMARY

```
SUMMARY
```

This parameter displays the Descriptor Space Summary (DSS) on the sequential input file that contains the compressed records. This display is identical to the one at the end of the ADACMP, ADAULD or ADAMUP run which generated this input file, and can be used to estimate the space required in the index.

Because the information has to be obtained from the last block of the input file on magnetic tape, a mechanical fast wind is required and some delay should be expected.

Additionally, the following information is displayed:

- required SORT size (for default LWP)
- recommended TEMP size (the size required to do the index update in one pass)
- current size of SORT (if present)
- LWP needed for memory-resident sort
- Recommended size of LWP and SORT (if LWP is large enough to allow a smaller SORT size to be used).



Note: If the default LWP is large enough to do a memory-resident sort, SORT sizes are not displayed.

UPDATE

```
UPDATE = number [,FDT]
        [ADD [,add_keywords]]
        [DELETE [,delete_keywords]]
        [, [NO]FORMAT]
        [LWP = number[K]]
```

This function specifies the file to which records are to be added/deleted. Since ADAMUP requires exclusive control of the file, it cannot be used for an Adabas system file while the nucleus is active. You are not allowed to specify a LOB file.

ADD

```
ADD
  [,DE_MATCH = keyword]
  [,FDT]
  [, [NO]NEW_FDT]
  [,NUMREC = number]
  [,SKIPREC = number]
  [,UQ_CONFLICT = keyword]
  [,RI_CONFLICT = keyword]
  [, [NO]USERISN]
```

This parameter indicates that records are to be added to the file specified by the UPDATE parameter.

The input for mass add is produced by the compression utility ADACMP, the unload utility ADAULD or by a previous run of the mass update utility ADAMUP using the DELETE function with the LOG option set.

ADAMUP compares the FDT in the sequential input file that contains the compressed records with the FDT of the database file specified. The FDTs must have identical layouts and must use the same field names, formats, lengths and options.

Descriptors in the database file can be a subset of the descriptors defined in the FDT in the sequential input file, but the input file must contain descriptor value table (DVT) entries for all descriptors defined in the database file. Therefore, input files produced without descriptor value tables (SHORT option) can only be processed if there are no descriptors currently defined in the database file to be updated.

If the input for mass update contains LOB data, the Adabas file must have an assigned LOB file.

DE_MATCH = keyword

This parameter is used to indicate which action is to be taken if a descriptor provided with the input data is not a descriptor in the actual FDT of the file. If keyword = IDENTICAL, ADAMUP terminates processing and returns an error message. If keyword = SUBSET, ADAMUP ignores a descriptor which is in the input file, but which has been removed from the database file.

The default is DE_MATCH=IDENTICAL.

[NO]NEW_FDT

If NEW_FDT is specified, the FDT of the file is replaced by the FDT of the MUPDTA file. NEW_FDT can only be specified if the file is empty when ADAMUP is started.

NEW_FDT must be specified if the FDT of the file in the database and the FDT of the MUPDTA file are different - a mass update is not possible if the FDTs are different and the file is not empty.

The default is NONEW_FDT.

NUMREC = number

This parameter specifies the number of records to be added. If NUMREC is specified, ADAMUP terminates after adding the predefined number of records, unless an end-of-file condition on the input file causes ADAMUP processing to end. If NUMREC is omitted and SKIPREC is not specified, all records in the input file are added.

SKIPREC = number

This parameter specifies the number of records in the input file to be skipped before starting to add records.

UQ_CONFLICT = keyword

This parameter is used to indicate which action is to be taken if duplicate values are found for a unique descriptor. 'keyword' may take the values ABORT or RESET. If ABORT is specified, ADAMUP terminates execution and returns an error status if duplicate UQ descriptor values are found. If RESET is specified, conflicting ISNs are written to the error log, the UQ status of the descriptors in question is removed and processing continues.

The default is UQ_CONFLICT=ABORT.

RI_CONFLICT

This parameter is used to indicate which action is to be taken if referential integrity is violated. 'keyword' may take the values ABORT or RESET. If ABORT is specified, ADAMUP terminates execution and returns an error status. The index is marked as not accessible. If RESET is specified, the violated constraint is removed. In both cases the violating ISNs are stored in the error log.

The default is RI_CONFLICT=ABORT.

[NO]USERISN

This option indicates whether the ISN to be assigned to each record is to be taken from the input file or not.

This option should be set to USERISN if the user wants to control ISN assignment for each record added to the database file. Each ISN provided must be:

- a four-byte binary number immediately preceding each data record;
- within the current limit (MAXISN) for the file - the file's Address Converter is not automatically extended;
- unique within the specified file.

Otherwise ADAMUP terminates execution and returns an error message.

Note that problems could arise if this option is set to USERISN for an input file created by an unload that is based on a descriptor which is a multiple-value field. This is because the same record may have been unloaded more than once. Please refer to the ADAULD utility, SORTSEQ parameter for more information.

If this option is set to NOUSERISN, the ISN for each record is assigned by ADAMUP. However, the ISN of a DVT record that has been previously re-vectored by a hyperexit will not be changed by ADAMUP.

The default is NOUSERISN.

DELETE

```
DELETE
  [,DATA_FORMAT = keyword]
  [,FDT]
  [,ISN_NOT_PRESENT = keyword]
  [,LOG = keyword | ,NOLOG]
```

This parameter indicates that records are to be deleted from the file specified by the UPDATE parameter. The ISNs of the records to be deleted are given in an input file.

DATA_FORMAT = keyword

This parameter defines the data type of the records in the input file containing the ISNs to be deleted. Each record contains one or more ISNs or ISN ranges.

Valid ISNs are within the range 1...MAXISN.

In accordance with the formats supported, 'keyword' may take the following values:

Keyword	Meaning
BINARY	<p>A single ISN is contained in a 4 byte binary value, an ISN range is contained in two consecutive binary values, with the high-order bit set in the second value.</p> <p>Blocks in this file start with 2 byte exclusive length field.</p> <p>Note: ISNs $\geq 2^{31}$ (2147483648) cannot be deleted with DATA_FORMAT=BINARY.</p>
DECIMAL	<p>Each record has the following layout:</p> <pre>[number[-number] [,number[-number]]...] [;comment]</pre> <p>where 'number' is decimal number with 1 to 10 digits.</p>

ADAMUP validates all input records in a first step. ADAMUP displays the line number and the offset for each error that is detected. If an error is detected, ADAMUP terminates execution once the input file has been completely parsed.

The default is DATA_FORMAT = BINARY.

ISN_NOT_PRESENT = keyword

This parameter indicates the action to be taken when an ISN given in the input file of records to be deleted is:

- not within the current limit (MAXISN) for the file;
- not in the file's Address Converter.

'keyword' may take the following values:

Keyword	Meaning
ABORT	ADAMUP aborts execution and returns an error message if a conflicting ISN is detected.
IGNORE	ADAMUP writes the conflicting ISNs to the error log and continues processing.

The default is ISN_NOT_PRESENT=IGNORE

LOG = keyword

NOLOG

LOG=keyword indicates that the deleted records are logged in a sequential file. The records are written in compressed format and are identical to those produced by the compression utility ADACMP and the unload utility ADAULD. Because each data record is preceded by its ISN, these ISNs can be used as user ISNs when reloading or mass-adding this file (see the USERISN option described above).

'keyword' may take the following values:

Keyword	Meaning
FULL	The descriptor values which are required to build the index, are included in the output file.
SHORT	The descriptor values which are required to build the index, are omitted from the output file.

ADAMUP writes both the compressed data records and the descriptor values generated to a single file.

The default is NOLOG.

[NO]FORMAT

This option may be used to format blocks at the end of the file's Normal Index (NI) and Upper Index (UI) extents if the new index (after the modifications have been made) requires less space than the old index did. This may be the result of deletions within the index, recovery of lost index blocks or re-establishing the padding factor.

Because these blocks are returned to the file's unused blocks, there are no side-effects if the data stored in these blocks is not deleted. If this option is set to FORMAT, ADAMUP overwrites these blocks with binary zeros.

The default is NOFORMAT.

LWP = number[K]

This parameter specifies the size of the Work Pool to be used for the sort during ADAMUP execution.

The minimum size and the default size is 0 bytes.

Restart Considerations

ADAMUP has no restart capability. An abnormally terminated ADAMUP must be rerun from the beginning.

If the Data Storage space becomes exhausted, ADAMUP will not abort, but will attempt to build the index for the records that have already been loaded; this means that the file is consistent, and the remaining records can then be loaded with the SKIPREC option after additional Data Storage space has been allocated.

SORT Data Set Placement

It is recommended that the SORT data set does not reside on the same volume as the Associator and the input file that contains the Descriptor Value Tables.

The SORT data set may be omitted when adding only small amounts of data. ADAMUP then performs an in-core sort.

Use the SUMMARY function to get information about the required SORT and LWP sizes.

TEMP Data Set Placement

It is recommended that the TEMP data set does not reside on the same volume as the input file that contains the Descriptor Value Tables and the SORT. Although the size of TEMP is closely related to the performance when loading the Normal/Main Index, successful execution does not depend on a given size or the presence of a TEMP.

Use the SUMMARY function to display the recommended TEMP size.

Examples

Example 1:

```
adamup: dbid=1
adamup: update=10
adamup: add, userisn
```

File 10 of database 1 is updated by adding new records. The ISN given with each input record is used.

Example 2:

```
adamup: dbid=1
adamup: update=10
adamup: delete
```

The records identified by the ISNs provided on the input file are to be deleted from file 10 of database 1. The ISNs to be deleted are in binary format.

Example 3:

```
adamup: dbid=1
adamup: update=10
adamup: add, skiprec=1000
adamup: delete, data_format=decimal, log=full
```

New records are to be added while old ones are deleted from file 10 of database 1. The first thousand records found on the input file are not added. The ISN for each record added is assigned by ADAMUP. The ISNs of the records to be deleted are supplied in decimal format on the input file. All records which have been deleted are logged on an output file. The values required to re-create the inverted list when reloading are included in the log.

18 ADANUC (Starting The Database, Defining Nucleus Parameters)

▪ Functional Overview	218
▪ Procedure Flow	220
▪ Checkpoints	222
▪ Control Parameters	222
▪ Summary of ADANUC Parameters	241

This chapter describes the utility "ADANUC". ADANUC is the database nucleus task.

Functional Overview

The nucleus parameters are used to define the Adabas nucleus runtime environment.

The nucleus parameters are set during nucleus startup.

The nucleus parameters provide the following information:

- The database to be used;
- The setting of various Adabas session parameters, such as the maximum Adabas buffer size and the transaction and user non-activity time limits;
- The type and amount of command data to be logged during the Adabas session. These parameters apply to statistical information and not to the logging of database updates on the Adabas data protection log.

This utility is a single-function utility.

Notes

1. If ADANUC terminates with a stop error during nucleus startup, e.g. STP055 or STP997, the reason is probably that there are not enough operating system resources (for example memory) available in order to start the nucleus with the specified parameters. You can prevent the stop error from occurring by reducing the values of some nucleus parameters (for example NT or LBP).
2. At the start of the first nucleus session after a database has been created or restored, the Adabas nucleus initializes all of the blocks of the WORK container. For a large WORK container this may take a few minutes. The database will only be available after the initialization of the WORK container has finished.
3. The default values for the nucleus parameters can be used if there are not more than 20 users who perform Adabas calls with relatively small Adabas buffers. The following hints describe conditions under which it may be necessary to use other nucleus parameter values:
 - If you temporarily have bad response times for Adabas commands (during an Adabas buffer flush), or if you get I/O errors during asynchronous I/Os even though your hardware is OK, consider setting BFIO_PARALLEL_LIMIT.
 - If Adabas commands with large adabas buffers are performed, e.g. for LOB processing, consider increasing the value for LAB and LBP.
 - If there are multi-threaded applications performing Adabas calls, consider increasing the NCL parameter.
 - If the number of parallel Adabas sessions is greater than 20, increase the NU parameter.

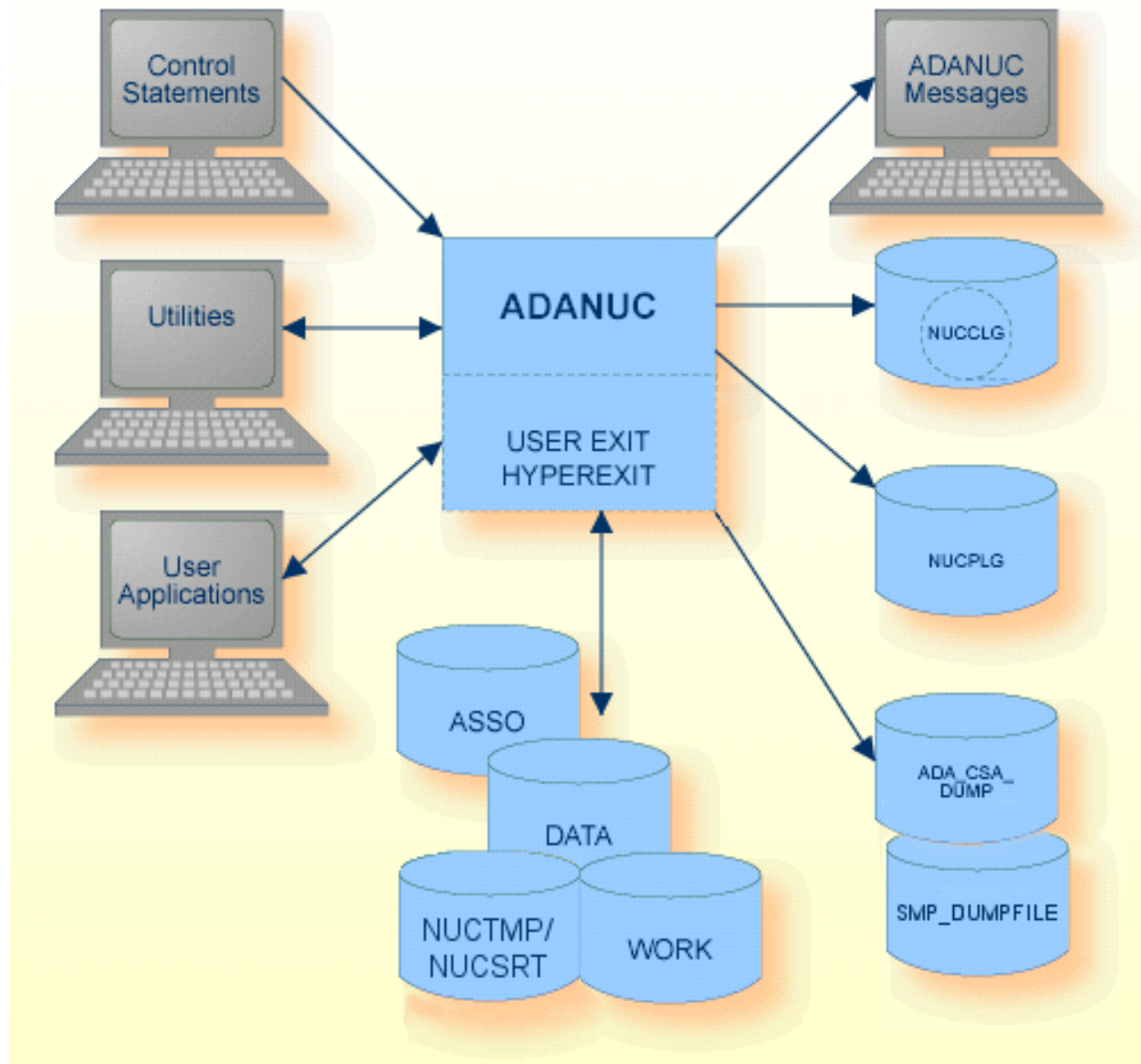
- If it is important that the autorestart time after a nucleus crash is short, set `WRITE_LIMIT` to a small positive value, especially if you are using a large buffer pool.
4. You can run a database in read-only mode by either removing the write permissions from the Adabas containers or by specifying `OPTIONS=READ_ONLY`. If you run the database in read-only mode, the temporary working space on disk is only created if you specify its location explicitly (environment variables `TEMPLOCn` or entries `TEMPORARY_LOCATION` in the `DBnnn.INI` file). For further information, see *Administration, Temporary Working Space*.
 5. If the previous Adabas session was not terminated normally with `SHUTDOWN` or `CANCEL`, Adabas performs an Autorestart: All transactions that were active when the nucleus crashed, are rolled back, and all missing database updates are written to `ASSO` and `DATA`. For this purpose, all update operations have been logged on the `WORK` container. Additionally, all update operations are logged in the `NUCPLG` file, which is required to recover the current state of the database, if one or more of the database containers has been corrupted, for example, because of a disk failure. In case of a nucleus crash, it is necessary that both logs contain the same information, otherwise the database could contain additional transactions, or transactions could be lost if you perform a restore/recover later. In order to check this, the `PLOG` file must still be available with the same name when an autorestart is performed. If you have renamed the `PLOG` file, or moved it to another location, you get the following warning:

```
%ADANUC-W-PLNOF, Last plog not found, so consistency check is not possible. New ↵
backup required.
```

If you get this warning, your database is still consistent, but if you perform a restore/recover later, it may be that your database then becomes inconsistent. If you create a new backup to be used as a base for restore/recover, the consistency of the restore/recover is guaranteed again.

6. You can define the nucleus parameters in the Adabas INI files via the DBA Workbench (not on OpenVMS). However, these parameters are not used when you execute `ADANUC` directly. If you want to start the nucleus with the nucleus parameters that are defined in the Adabas INI files, without using the DBA Workbench, you can use the Adabas Extended Operation command `adastart` (see *Adabas Extended Operations, Administration Commands*). On Windows platforms, there is also the utility `ADABAS` for starting `ADANUC` with the nucleus parameters in the Adabas INI files.
7. On UNIX platforms, the IPC resources allocated by the Adabas nucleus are not removed if the nucleus is not shut down normally with `ADAOPR SHUTDOWN` or `ADAOPR CANCEL`. The nucleus can only be restarted after these resources have been removed, and for this reason the nucleus executes the command `showipc -k <dbid>`. Important: if Adabas was not installed as recommended by user "sag", group "sag", it is necessary to set the environment variable `SIPUSER` and/or `SIPGROUP`. For further information, see *Administration, showipc*. On Windows platforms, the operating system automatically removes IPC resources that are no longer used.

Procedure Flow



The sequential files NUCPLG and NUCCLG can have multiple extents. For detailed information about sequential files with multiple extents, see *Administration, Using Utilities*.

Data Set	Environment Variable/ Logical Name	Storage Medium	Additional Information
Associator	ASSOx	Disk	
Command log	NUCCLG	Disk, Tape	Utilities Manual, ADACLPL
Data storage	DATAx	Disk	
Protection log	NUCPLG	Disk	Utilities Manual, ADAPLP
Control statements	stdin/ SYS\$INPUT		Utilities Manual
ADANUC messages	stdout/ SYS\$OUTPUT		Messages and Codes
Work	WORK1	Disk	
CSA dump	ADA_CSA_DUMP	Disk	Adabas CSA dump (see ADAOPR RESPONSE_CHECK)
SMP dump	SMP_DUMPFILE	Disk	SMP dump
Temporary working space (NUCTMPx, NUCSRTx)	TEMPLOCx	Disk	Administration Manual (see Temporary Working Space)



Notes:

1. In the environment variable/logical name ADA_CSA_DUMP, the directory in which the Adabas CSA dump is to be created must be specified.
2. The default directory for the Adabas CSA dump file is the database directory. For the layout of the file names for Adabas CSA dumps, see ADAOPR, parameter CSA.
3. For further information about the Adabas CSA dumps, see also the ADAOPR parameters ABORT and RESPONSE_CHECK.
4. In the environment variable/logical name SMP_DUMPFILE, the name of the SMP_DUMPFILE itself (not a directory) must be specified. An existing file is overwritten.
5. If the SMP_DUMPFILE is not specified, the SMP dump is written into the Adabas CSA dump directory with the file name SAGSMP.xxx.hh:mm:ss (UNIX), SAGSMP.xxx.hh-mm-ss (Windows) and SAGSMP-xxx-hh-mm-ss (OpenVMS) where xxx is the database ID and hh:mm:ss/hh-mm-ss is the nucleus start-time.

Checkpoints

The following table shows the checkpoints written by the nucleus:

Function	Checkpoint written
Nucleus startup	SYNC
Nucleus termination	SYNC

Control Parameters

The following control parameters are available:

```
D  ADABAS_ACCESS = {ALL | GROUP}
D  AR_CONFLICT = keyword
   BFIO_PARALLEL_LIMIT = number
D  [NO]BI
D  CLOGBMAX = number [K | M]
D  CLOGLAYOUT = [5 | 6]
M  DBID = number
D  LAB = number [K | M]
D  LABX = number [K | M]
D  LBP = number [K | M]
D  LOGGING = (keyword [,keyword]...)
D  LPXA = number
D  LWP = number [K | M]
D  MGC = number
D  NCL = number
D  NISNHQ = number
```

```

D   NT = number
D   NU = number
D   OPTIONS = (keyword[,keyword]...)
D   [NO]PLOG
D   READ_PARALLEL_LIMITS = (records,blocks,total)
D   TNAA = number
D   TNAE = number
D   TNAX = number
D   TT = number
D   TZ = ['timezone']
D   UNBUFFERED = ALL | CLEAR | (keyword [, keyword [, keyword]])
D   USEREXITS = (number[,number]...)
D   WRITE_LIMIT = number

```

ADABAS_ACCESS

```
ADABAS_ACCESS = {ALL | GROUP}
```

If ADABAS_ACCESS = ALL is specified, all users may perform Adabas calls.

If ADABAS_ACCESS = GROUP is specified, only those users that belong to the group of the user starting Adabas, for example *sag*, may perform Adabas calls.

The default is ALL.

Please refer to *Administration, Database Security Overview, Using the UNIX Group Concept* for further information about using the ADABAS_ACCESS parameter.



Note: This parameter only applies to local Adabas calls in UNIX environments.

AR_CONFLICT

AR_CONFLICT = keyword

This parameter specifies the action to be taken if a restart detects that the last system crash was during a buffer flush. The following keywords can be used:

Keyword	Meaning
ABORT	no restart is performed
CONTINUE	the system tries to perform a restart

It is recommended to keep the setting AR_CONFLICT=ABORT. Only if the nucleus does not come up due to an interrupted buffer flush, then temporarily AR_CONFLICT=CONTINUE should be set.



Note: You should be aware of the possibility of inconsistencies in the case of a restart with AR_CONFLICT=CONTINUE. Consistency should be checked with the ADAINV VERIFY function in this case.

The default is ABORT.

BFIO_PARALLEL_LIMIT

BFIO_PARALLEL_LIMIT = number

This parameter is used to limit the number of parallel I/O requests by a buffer flush and to allow earlier processing of concurrent I/Os from other threads. A large buffer flush, for example, can cause the I/O queue to be very busy, and other I/Os (such as buffer pool read I/Os and WORK I/Os) can be enqueued for a long time, slowing down command throughput and possibly causing applications to stall if a buffer flush is active.

If BFIO_PARALLEL_LIMIT is specified, the buffer flush sets up the specified number of I/Os and waits until these have been processed before issuing the next packet. The maximum value for 'number' is defined by the operating system, for example by the UNIX kernel parameter AIO_LISTIO_MAX. Specifying a value of 0 is equivalent to specifying the maximum value allowed. The default value is 50.



Note: If the value of BFIO_PARALLEL_LIMIT is too high (or equal to 0), this can result in an I/O error during asynchronous IO (utility error message: ADRERR). The reason for this is that the memory available within the operating system for asynchronous I/O is exhausted. You can imagine that the required memory is at least the size of the blocks to be written to the database plus some additional space. The maximum value for BFIO_PARALLEL_LIMIT for which you can be sure that this ADRERR error will not occur, depends on the operating system configuration and on the other processes that are active on the same machine. 50 seems to be a useful value for BFIO_PARALLEL_LIMIT; if the ADRERR error occurs nevertheless, you should check your operating system configuration.

Examples

```
adanuc: bfio_parallel_limit = 20
```

Twenty parallel I/O requests are permitted for a buffer flush.

```
adanuc: bfio_parallel_limit = 0
```

The number of buffer flush I/Os is unlimited.

[NO]BI

```
[NO]BI
```

This option is used to specify whether before images are written to the PLOG (BI) or not (NOBI).

The before images on the PLOG are used during a regenerate in order to verify the data consistency (for example, whether the appropriate PLOG is being used). If NOBI is set, the PLOG is smaller, but the consistency verifications cannot be performed.

The default is BI.

Note that if you specify NOBI, this will reduce the amount of consistency checking possible when using the ADAREC REGENERATE function. See *ADAREC* for more details.

CLOGBMAX

```
CLOGBMAX = number[K | M]
```

This parameter specifies the maximum Adabas buffer length that is logged in the command log. If an Adabas buffer is larger than the value specified, the buffer is truncated in the command log. 0 means that the complete buffers are always logged.

The default is 0 (the complete buffers are logged).

Example

```
adanuc: clogbmax = 4k
```

The logging of Adabas buffers is limited to the first 4 kilobytes of each buffer.

CLOGLAYOUT

```
CLOGLAYOUT = [5|6]
```

If you specify CLOGLAYOUT=5, the command log has the same layout as for Adabas Version 5 when command logging is enabled. Since this layout does not support the ACBX interface, only the subset of information that is also available in the old ACB interface is logged for ACBX calls. The command log can be evaluated with the ADACLP utility that already existed in old versions of Adabas.

If you specify CLOGLAYOUT=6, the command log is generated in the new layout that is supported by Adabas Version 6. For evaluating the command log, an example program `pri logc` is provided – for details see [Appendix B](#).

The default value is 5.

DBID

```
DBID = number
```

This mandatory parameter selects the database to be used. 'number' is in the range from 1 to 255.

Example:

```
adanuc: dbid = 2
```

Database 2 is used.

LAB

```
LAB = number[K | M]
```

This parameter specifies the size of the attached buffer area to be used during the Adabas session. The attached buffer area is used for allocating the user buffers during command execution.

Adabas rounds up the given value to the next multiple of 32 kilobytes to allocate the attached buffer area.

The high water mark for this parameter can be displayed using the DISPLAY parameter in ADAOPR.

The minimum value is 1 megabyte, the default value is 1 megabyte, however, if the value specified is less than the value of the NCL parameter in kilobytes, the value for LAB is automatically increased to that value.

**Notes:**

1. The attached buffers must be large enough to contain, for each local client thread, the maximum total size of user buffers for an Adabas call performed by this thread. Additional space may be required for the administration of the user buffers and for waste resulting from the different sizes required for different threads.
2. Attached buffers > 64 KB are allocated in the separate attached buffer extension area (see parameter **LABX**).

LABX

```
LABX = number[K | M]
```

This parameter specifies the size of the attached buffer extension area to be used for LOB processing or for other Adabas calls that use large Adabas buffer areas. While the regular attached buffers remain allocated during the whole Adabas session of the users, the attached buffer extensions are allocated before each Adabas call that requires more than 64 KB of attached buffer space, and they are released again at the end of the call.

The attached buffer extension area was introduced because storing large buffers in the regular attached buffer area requires much more space because of fragmentation.

Adabas rounds up the given value to the next multiple of 32 kilobytes to allocate the attached buffer area.

The high water mark for this parameter can be displayed using the DISPLAY parameter in ADAOPR.

The minimum value is 1 megabyte, the default value is 20 megabytes.

LBP

```
LBP = number[K | M]
```

This parameter specifies the size in bytes of the Adabas buffer pool during the session.

The high water mark for this parameter can be displayed using the DISPLAY parameter in ADAOPR.

The minimum value is 16 megabytes, the default value is 100 megabytes.



Note: The actual size of the buffer pool also depends on the NT parameter: if the size specified for LBP is less than NT * 2M, it is automatically increased to NT * 2M.

There are the following reasons for using a buffer pool in Adabas:

- If the same database block is accessed more than once, physical I/Os for rereading the block can be avoided;
- Commands that update the database can be finished before all of the corresponding database blocks have been written to the disk.

If the buffer pool is too small, you have a larger number of I/Os than you would when performing the same database operations with a sufficient buffer pool size. In some cases, it may even occur that Adabas cannot successfully process an Adabas command, in which case the command will get a response code 162 (buffer pool too small).

In particular, if you process LOBs, it is recommended that you increase the size of the buffer pool; it should be possible to put the LOBs into the buffer pool without removing too many other database blocks. On the other hand, the buffer pool should not be too large, since there is no advantage in having buffer pool I/Os replaced by operating system paging I/Os.

You can use the ADAOPR parameter DISPLAY=BP_STATISTICS in order to help you find the optimum size for the buffer pool.

Example:

```
adanuc: lbp=16m
```

16 megabytes are allocated to the Adabas buffer pool.

LOGGING

```
LOGGING = (keyword [,keyword]...)
```

This parameter specifies logging of the buffers as defined in the keyword list.

The following keywords may be specified in the keyword list:

Keyword	Meaning
CB	log Adabas control block
FB	log format buffers
RB	log record buffers
SB	log search buffer
VB	log value buffer
IB	log ISN buffer
ABD	log Adabas buffer descriptions
IO	log I/O activity
OFF	perform no logging

Logging parameters may also be specified while a session is executing by using operator commands as described in the section on the utility ADAOPR.

Example

```
adanuc: logging=(cb)
```

Command logging is performed for the current Adabas session; all of the Adabas control blocks will be logged.

LPXA

```
LPXA = number
```

This parameter specifies the number of blocks reserved for data protection information on the Adabas WORK in the XA context if OPTIONS=XA is specified. These blocks are used to save protection information for transactions which run in the XA context and are in the pending state (prepared but not yet committed) at nucleus shutdown or for a long time. Each such transaction uses at least one of these blocks. If there are no free blocks in this area, Adabas will heuristically commit such transactions.

The minimum value is 1, and the default value is 10.

LWP

```
LWP = number[K | M]
```

This parameter specifies the size of the Adabas work pool, which is a work area in memory to be used for the Adabas nucleus session.

The Adabas work pool area is used to store the following:

- Descriptor value table (DVT);
- WORK I/O areas during command execution.

The high water mark for this parameter can be displayed using the DISPLAY parameter in ADAOPR.

The default value is 16 M, the minimum value is 500 K.

Example:

```
adanuc: lwp=750k
```

The size of the Adabas work pool is 750 Kbytes.

MGC

```
MGC = number
```

This parameter specifies the maximum group-commit count. This defines the maximum limit of ET command grouping before the PLOG buffers are written back to disk. If this limit is reached, and the final I/O has been performed, all remaining users will be posted.

The high water mark for this parameter can be displayed using the DISPLAY parameter in ADAOPR.

The minimum value is 1, the default value is 50 and the maximum value is 500.

Example:

```
adanuc: mgc=80
```

The maximum group-commit count is 80.

NCL

```
NCL = number
```

This parameter specifies the number of client threads locally accessing the database.

The minimum value is 2. The default value is the value for the NU parameter but at least 50. There is no fixed maximum value for this parameter, it is only limited by the IPC resources that are available for the operating system.

Note that Net-Work Version 7 also uses several threads to access the database; please refer to the Net-Work documentation for further details.



Note: The value specified for the NCL parameter should not be too high. When Adabas sessions are terminated correctly with a close command, the associated message queues are removed, but if the sessions are terminated abnormally without a close command, the message queues are not removed. A cleanup of the message queues is only performed when the number of client message queues is equal to the value specified for the NCL parameter - therefore, it may happen that Adabas attempts to create more message queues than it is allowed to do by the system, if the value for the NCL parameter is too high, even though

the actual number of active Adabas users is very small. If this happens, the Adabas commands receive a response 255 (see *Messages and Codes* for further information).

NISNHQ

```
NISNHQ = number
```

This parameter specifies the maximum number of records that can be placed into hold at any time by a single user.

If a user attempts to place more records in the hold status than permitted, he will receive a non-zero response code, even though there may still be space in the hold queue.

The high water mark for this parameter can be displayed using the DISPLAY parameter in ADAOPR.

The minimum value is 0 - where 0 means unlimited, the default value is 0 (unlimited).

Example

```
adanuc: nishq=50
```

The maximum number of records which can be in the hold status for a single user is 50.

NT

```
NT = number
```

This parameter specifies the number of threads to be established for the Adabas session.

Each Adabas command is assigned to a thread. A thread is released when the command has been processed.

The high water mark for this parameter can be displayed using the DISPLAY parameter in ADAOPR.

The minimum value is 1, the maximum value 100 and the default value is 6.



Notes:

1. It is strongly recommended to use an NT parameter value greater than 1, because some internal commands can only be executed if NT > 1. If NT is 1, some utilities or applications, such as SQL Gateway, which use these internal commands, may fail.
2. NT = 1 only makes sense in some special situations, for example, if support requests an ADANUC trace and you want to have the trace of all Adabas commands in one file.

3. Increasing the NT parameter usually implies an overhead, which may reduce the performance. Therefore, it usually doesn't make sense to use an NT parameter value that is significantly higher than the number of hardware threads; if you have a large number of hardware threads, the performance may even be better if NT is less than the number of hardware threads, because the Adabas nucleus is not the only program running on the machine and using CPU time.
4. One reason to increase the value of the NT parameter is if there are several complex commands in parallel that require a long time to complete. If all nucleus threads are blocked by such commands, it can happen that short running commands cannot be scheduled for a long time, and as a result the overall performance of Adabas becomes very low.
5. With previous versions of Adabas, it was possible that short running commands could no longer be scheduled because complex, long-running commands blocked all threads, which resulted in very poor overall performance. In order to avoid this situation, now only NT/2 threads can be used to process such long-running commands in parallel. As a consequence of this, it may be that you can see several free threads even though there are sufficient commands in the command queue waiting to be scheduled.

Example:

```
adanuc: nt=8
```

Eight threads are established for the session.

NU

```
NU = number
```

This parameter specifies the number of user queue elements to be established for the Adabas session.

A user queue element is assigned to each active Adabas user. A user queue element is assigned when the user issues an OP command or when the first Adabas command is issued. A user queue element is released when the user issues a CL command or when the user queue element is deleted on a timeout.

The high water mark for this parameter can be displayed using the DISPLAY parameter in ADAOPR.

The minimum value is 2, the default value is 20.

Example:

```
adanuc: nu=100
```

The Adabas user queue consists of 100 elements.



Note: The NU parameter is used for two different purposes:

- the number of user queue entries
- the number of communication blocks

For single-threaded applications, the number of required communication blocks is not larger than the number of user queue elements, but for multi-threaded applications you need one communication block for each thread of an application that performs Adabas calls. Therefore the required value of the NU parameter may be much higher than the number of Adabas users.

OPTIONS

```
OPTIONS = (keyword[,keyword]...)
```

This parameter is used to define the mode(s) in which the nucleus is started.

The following keywords are permitted:

Keyword	Meaning
AUTO_EXPAND	If AUTO_EXPAND is selected, the database will increase the existing container extents or create a new container extent if there is no free space available for increasing an existing Adabas file or for adding a new Adabas file. See the section <i>Container Files</i> in the <i>Administration</i> section for further information.
AUTORESTART_ONLY	The AUTORESTART_ONLY keyword shuts down the nucleus immediately after its startup sequence has completed. If an autorestart is pending, the autorestart will be performed. No user commands or utility calls will be accepted by the nucleus.
FAULT_TOLERANT_AR	When FAULT_TOLERANT_AR is selected, the nucleus behaviour in the event of an Adabas error during autorestart can be controlled. If an error occurs for a given file, a detailed entry is made in the nucleus log, but the autorestart continues. When the autorestart completes, the DBA can restore and regenerate the file by using ADABCK RESTORE and ADAREC REGENERATE for the affected file. If however the error occurred for the index of a file, it is sufficient to rebuild the file's index by using the REINVERT function of ADAINV. If FAULT_TOLERANT_AR is not selected, Adabas aborts an autorestart if an error is detected, and this can cause the database to be in an inconsistent state.
LOCAL_UTILITIES	If LOCAL_UTILITIES is selected, ADANUC rejects all remote utility calls, i.e. the Adabas utilities cannot be run from a remote node across a network. This setting is recommended if security is important in your operating environment.

Keyword	Meaning
	LOCAL_UTILITIES and UTILITIES_ONLY can be dynamically enabled or disabled without having to shut the database down (see <i>ADAOPR</i> for further information).
OPEN_REQUIRED	If OPEN_REQUIRED is selected, an open (OP) command must be issued as the first command of a user session. This option should be set if lnk_set_adabas_id is used when calling Adabas from application servers, and also when using Net-Work, otherwise in these cases Adabas cannot guarantee transaction integrity following an ADANUC restart.
READONLY	The READONLY option causes ADANUC to run in read-only mode. Refer to the Administration Manual for more details.
TRUNCATION	The TRUNCATION keyword controls the truncation of alphanumeric fields. If TRUNCATION is set, alphanumeric fields are truncated if necessary and response code 0 is returned. If TRUNCATION is not set, response code 55 is returned if truncation occurred.
UTILITIES_ONLY	If UTILITIES_ONLY is selected, all calls other than for utilities will be rejected, giving the DBA exclusive control over all of the files in the database. Note, however, that this restriction only applies to new users; users who were already active when OPTIONS=UTILITIES_ONLY was specified can continue processing normally. If you want exclusive utility control over files or the entire database, use the LOCK function of ADAOPR instead. LOCAL_UTILITIES and UTILITIES_ONLY can be dynamically enabled or disabled without having to shut the database down (see <i>ADAOPR</i> in this manual for further information).
XA	The keyword XA indicates that the server will support distributed transaction processing according to the X/Open XA specification. If the Adabas XA interface is to be used by an application, OPTION=XA must be used. See <i>Administration, XA Support</i> for further information.

The default is that no option is set.

Example:

```
adanuc: options = utilities_only
```

All non-utility calls are rejected and the DBA has exclusive control over all database files.

[NO]PLOG

[NO]PLOG

PLOG specifies which protection log is to be switched on.

The database cannot be regenerated if the disk is physically damaged and if there is no protection log. In this case, the database has to be restored using the last database dump. All updates made since this last dump was taken are then lost.

PLOG is the default.

READ_PARALLEL_LIMITS

READ_PARALLEL_LIMITS = (records,blocks,total)

For sequential commands, Adabas tries to increase the performance by reading the database blocks required for the following records in advance and in parallel. This lets you take advantage of disk striping, and the sequence of I/Os can be optimized. Significant performance improvements can only be expected when the physical I/O rate is high. A typical case where the READ_PARALLEL_LIMITS parameter can improve the performance is the execution of a batch program in which a complete large file is read, and this file contains a large number of database blocks that are not in the buffer pool when the program is started. However, this prefetching of database blocks can also have a negative impact on the performance:

- Checking which blocks are required for the next records requires a certain amount of CPU time, and this is superfluous if all blocks are already in the buffer pool.
- If you don't read the next records of the command sequence, the overhead for reading blocks for this command sequence is superfluous.
- If I/Os for too many blocks have been initiated, an I/O required for another command performed in parallel may be delayed significantly.

The READ_PARALLEL_LIMITS parameter lets you control the read-ahead behaviour. You can specify 3 numbers with the following meanings:

records

The maximum number of records to be processed next in a command sequence, that are checked for blocks that are not yet in the buffer pool. The maximum value that can be specified is 65,535.

blocks

The maximum number of blocks read in advance for one command sequence. The number specified for blocks must be \leq the number specified for records, the maximum value is operating-system dependent.

total

The maximum total number of blocks read in advance at the same time for the whole database. The number specified for total must be \geq the number specified for blocks, the maximum value is operating-system dependent.

The default is (0,0,0), i.e. no read-ahead is performed.



Note: If the value of the number specified for `total` is too high, this can result in an I/O error during asynchronous IO (utility error message: ADRERR). The reason for this is that the memory available within the operating system for asynchronous I/O is exhausted. You can imagine that the required memory is at least the size of the blocks to be written to the database plus some additional space. The maximum value for `total` for which you can be sure that this ADRERR error will not occur, depends on the operating system configuration and on the other processes that are active on the same machine.

Example

```
adanuc: read_parallel_limits = (100,20,50)
```

When sequential commands are processed, the next up to 100 records of a command sequence are checked for required blocks that are not yet in buffer pool. The search for blocks to be read is stopped when 20 blocks are found, or when the number of blocks found plus the number of blocks currently read by other commands is 50. Then a read I/O for these blocks is initiated.

TNAA

```
TNAA = number
```

This parameter specifies the maximum elapsed time (in seconds) that an access-only user may be active without issuing an Adabas command. This value can be changed dynamically with the ADAOPR utility.

The OP command allows you to override this value. See *Command Reference, OP command* for details.

See *Command Reference, Time Limits* for a table with timeout conditions.

Note that the figure you specify for this parameter is only approximate. In any particular instance, the actual amount of time can vary from this value by up to 10 seconds.

The minimum value is 20, the default value is 900 and the maximum value is 2592000.

Example:

```
adanuc: tnaa=180
```

The non-activity time limit for access-only users is 180 seconds.

TNAE

```
TNAE = number
```

This parameter specifies the maximum elapsed time (in seconds) that an ET logic user may be active without issuing an Adabas command. This value can be changed dynamically with the ADAOPR utility.

The OP command allows you to override this value. See *Command Reference, OP command* for details.

See *Command Reference, Time Limits* for a table with timeout conditions.

Note that the figure you specify for this parameter is only approximate. In any particular instance, the actual amount of time can vary from this value by up to 10 seconds.

The minimum value is 20, the default value is 900 and the maximum value is 2592000.

Example:

```
adanuc: tnae=180
```

The non-activity time limit for ET logic users is 180 seconds.

TNAX

```
TNAX = number
```

This parameter specifies the maximum elapsed time (in seconds) that an exclusive update user who does not use ET logic may be active without issuing an Adabas command. This value can be changed dynamically with the ADAOPR utility.

The OP command allows you to override this value. See *Command Reference, OP command* for details.

See *Command Reference, Time Limits* for a table with timeout conditions.

Note that the figure you specify for this parameter is only approximate. In any particular instance, the actual amount of time can vary from this value by up to 10 seconds.

The minimum value is 20, the default value is 900 and the maximum value is 2592000.

Example:

```
adanuc: tmax=180
```

The non-activity time limit for exclusive update users is 180 seconds.

TT

```
TT = number
```

This parameter specifies the maximum elapsed time (in seconds) permitted for a logical transaction issued by an ET logic user. This value can be changed dynamically with the ADAOPR utility.

The OP command allows you to override this value. See *Command Reference, OP command* for details.

The time measurement for a logical transaction starts when the first command that places a record in hold status is issued, and terminates when an ET, BT, or CL command is issued. Adabas takes the following action when this time limit is exceeded:

1. All database updates made during the transaction are backed out;
2. All records held during the transaction are released;
3. All Command IDs for the user are released;
4. Response code 9 is returned on the next call issued by the user.

This time limit does not apply to non-ET logic users. The value specified for TT directly influences the required size of the Adabas Work data set.

The high water mark for this parameter can be displayed using the DISPLAY parameter in ADAOPR.

Note that the figure you specify for this parameter is only approximate. In any particular instance, the actual amount of time can vary from this value by up to 10 seconds.

The minimum value is 20, the default value is 300 and the maximum value is 2592000.

Example:

```
adanuc: tt=50
```

The transaction time limit for ET logic users is 50 seconds.

TZ

```
TZ = ['timezone']
```

The specified time zone must be a valid time zone name that contained in the time zone database known as the Olson database (<http://www.twinsun.com/tz/tz-link.htm>). If a time zone has been specified, this time zone is used for time zone conversions of date/time fields with the option TZ, unless a user has specified his own time zone in the OP command.

The default is none. If no time zone is specified, a time zone conversion can only be done for users who specified a time zone in the OP command - other users get an Adabas response code 48.

UNBUFFERED

```
UNBUFFERED = ALL | CLEAR | (keyword [, keyword [, keyword]]) ←
```

This parameter is relevant only for UNIX platforms, and if database containers or the protection log are stored in a file system. Usually, write I/Os are buffered by the file system. However, if you specify the O_DSINC option for the corresponding open call, write I/O operations are performed unbuffered. This may improve the performance, in particular, if specified for WORK and PLOG, as it is required that the corresponding log information is really on disk at the end of a transaction. With the parameter UNBUFFERED, the usage of the O_DSINC operation may be defined.



Note: Usage of this parameter only has impact on the performance of Adabas; which actual setting of the UNBUFFERED parameter results in the best performance depends on the operating system and the storage system used - the integrity of the database is guaranteed for all values of the UNBUFFERED parameter.

The following keywords may be specified in the keyword list:

Keyword	Meaning
DATABASE	ASSO and DATA containers in a file system are opened with the option O_DSINC.
NODATABASE	ASSO and DATA containers in a file system are not opened with the option O_DSINC.
WORK	A WORK container in a file system is opened with the option O_DSINC.
NOWORK	A WORK container in a file system is not opened with the option O_DSINC.
PLOG	A protection log in a file system is opened with the option O_DSINC.
NOPLLOG	A protection log in a file system is not opened with the option O_DSINC.

The keyword ALL is equivalent to (DATABASE, WORK, PLOG).

The keyword CLEAR is equivalent to (NODATABASE, NOWORK, NOPLLOG).

The default is (NODATABASE, WORK, PLOG).

Example:

```
adanuc: unbuffered=(nowork)
```

O_DSYNC is not used for ASSO and DATA containers because NODATABASE is the default.

O_DSYNC is not used for WORK container because NOWORK has been specified.

O_DSYNC is used for protection logs because PLOG is the default.

USEREXITS

```
USEREXITS = (keyword [,keyword]...)
```

This parameter, used in conjunction with the user exit facility, specifies one or more user exits. The specified user exit(s) must be loadable at execution time.

The keyword can take the values 1, 2, 4, 11 and 14.



Notes:

1. User exit 1 and 11 are mutually exclusive.
2. User exit 4 is only activated if CLOGLAYOUT=5 is specified. User exit 14 is only activated if CLOGLAYOUT=6 is specified.

See *Administration, User Exits and Hyperexits* for further information about the user exits available.

WRITE_LIMIT

```
WRITE_LIMIT = number
```

This parameter specifies the percentage of modified blocks permitted in the buffer pool before an implicit buffer flush is taken.

A number in the range from 0 to 70 may be specified; 0 means that Adabas will dynamically choose an appropriate value.



Note: If you have a large buffer pool, it is useful to specify a small value for WRITE_LIMIT in order to reduce the time required for an auto restart of the nucleus: during an auto restart, ADANUC must read the update log in the WORK container and reapply all changes in the database that had not yet been written to disk. A smaller value for WRITE_LIMIT means a smaller amount of data not yet written to disk, and therefore less updates to be made during the auto restart.

Example:

```
adanuc: write_limit=40
```

An implicit buffer flush is taken when 40% of the blocks in the buffer pool have been modified.

Summary of ADANUC Parameters

Parameter	Use	Min. Value	Max. Value	Default	Dynamic (see note 3)
ADABAS_ACCESS	Restrict user access for making Adabas calls				
AR_CONFLICT	Restart after crash during buffer flush			ABORT	No
BFIO_PARALLEL_LIMIT	Limit of parallel buffer flush IOs	0	None (see note 4)	0	Yes
[NO]BI	Write before image to PLOG			BI	No
CLOGBMAX	Length of Adabas buffers logged			0 (complete)	No
CLOGLAYOUT	Select layout of CLOG			5	No
DBID	Database	1	255	None	No
LAB	Attached buffer area	1MB	None	1MB	
LABX	Attached buffer extension area	1MB	None	20MB	
LBP	Maximum Adabas Buffer Pool Size	16MB	None (see note 1)	100MB	No
LOGGING	Command Logging			OFF	Yes
LPXA	XA data protection area size	1 block	65535 blocks	10 blocks if the XA option is set	No
LWP	Length of Adabas Work Pool	16 MB	None	500 KB	No
MGC	Maximum Group-Commit Count	1	500	50	No
NCL	Number of local client threads	2	None	=NU, but at least 50	No
NISNHQ	Maximum Number of ISNs in Hold per User	0	None	0	Yes

Parameter	Use	Min. Value	Max. Value	Default	Dynamic (see note 3)
NT	Number of Threads	1	100	6	No
NU	User Queue Size	2 users	None	20 users	No
OPTIONS	Various Options			No option	Some
[NO]PLOG	Protection Logging			PLOG	No
READ_PARALLEL_LIMITS	Optimize read-ahead I/O performance	(0,0,0)	65535 for records. The maximum values for blocks and total depend on the operating system.	(0,0,0)	Yes
TNAA	Non-Activity Time Limit (ACC Only Users)	20	2592000	900 seconds	Yes
TNAE	Non-Activity Time Limit (ET Logic Users)	20	2592000	900 seconds	Yes
TNAX	Non-Activity Time Limit (EXU,EXF Users)	20	2592000	900 seconds	Yes
TT	Transaction Time Limit	20	2592000	300 seconds	Yes
UNBUFFERED	Setting of the O_DSYNC option (UNIX platforms only)			NODATABASE, WORK, PLOG	No
USEREXITS	User Exit(s) to be used			None	No
WRITE_LIMIT	Buffer Pool Modification Limit	0	70	None (see note 2)	Yes



Notes:

1. A limit may be given by the operating system, for example via the maximum size of a shared memory segment.
2. If not specified, Adabas dynamically chooses an appropriate value.
3. If the value of a dynamic parameter is changed using ADAOPR, the new value takes effect immediately, without the nucleus having to be restarted. In the case of non-dynamic parameters, the nucleus must first be stopped and restarted for the new value to take effect.

4. A value of 0 means there is no limit.

19 ADAOPR (Operator Utility)

▪ Functional Overview	246
▪ Procedure Flow	247
▪ Checkpoints	248
▪ Control Parameters	248

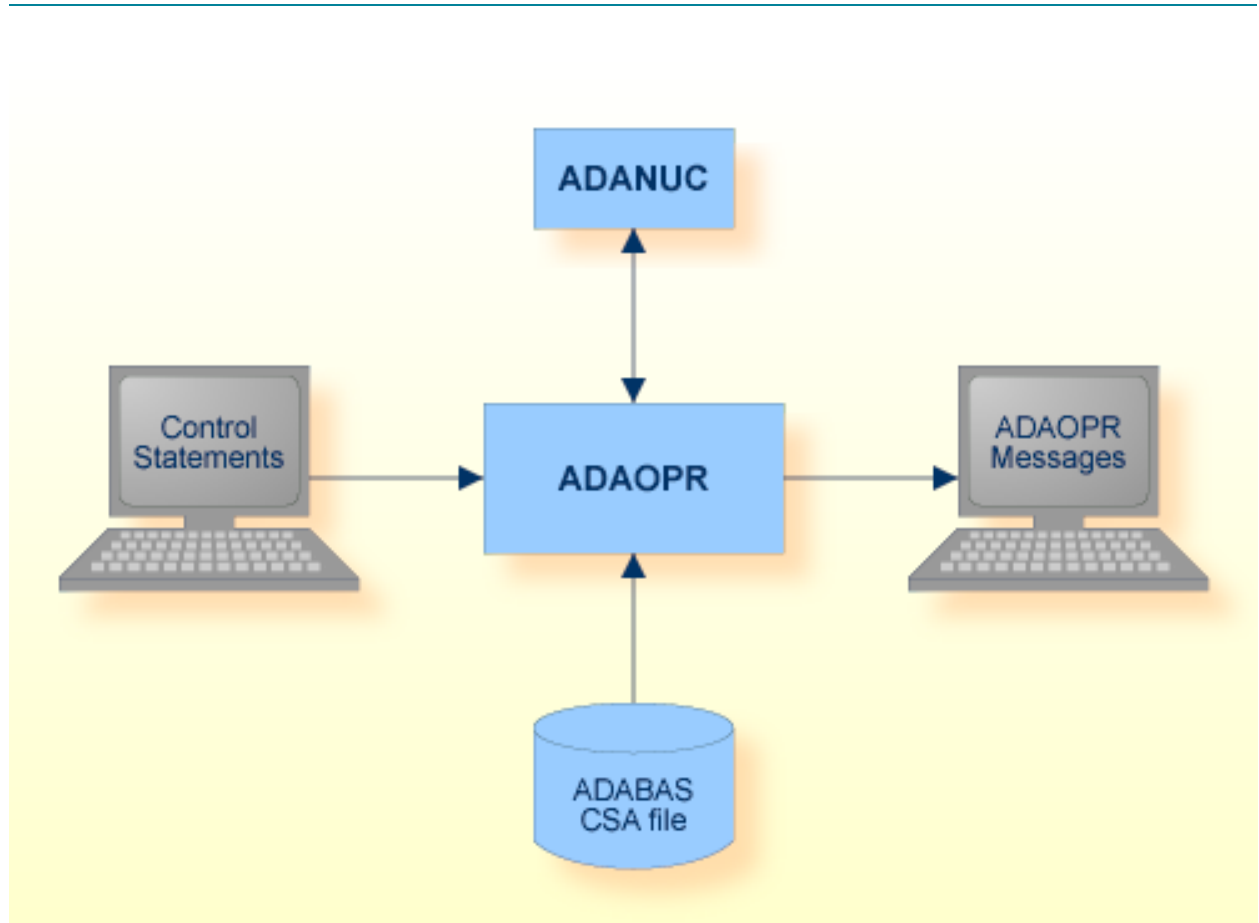
This chapter describes the utility "ADAOPR".

Functional Overview

The DBA uses this utility to operate the Adabas nucleus.

This utility is a multi-function utility.

Procedure Flow



Data Set	Environment Variable/ Logical Name	Storage Medium	Additional Information
Control statements	stdin/ SYS\$INPUT		Utilities Manual
ADAOPR messages	stdout/ SYS\$OUTPUT		Messages and Codes

Checkpoints

The following table shows the nucleus requirements for each function and the checkpoint written:

Function	Nucleus must be active	Nucleus must NOT be active	Nucleus is NOT required	Checkpoint written
FEOF=PLOG				SYNC

Control Parameters

The following control parameters are available:

```

ABORT

BFIO_PARALLEL_LIMIT = number

CANCEL

CLEAR_FILE_STATS = (number [- number] [, number [- number] ] ... )

CSA = string

DBID = number

DISPLAY = (keyword [,keyword]...)

ES_ID = number

D [NO]JET_SYNC

EXT_BACKUP = [PREPARE | CONTINUE | ABORT]

FEOF = (keyword [,keyword])

FILE = number

FREE_CLQ = number

ID = number

D [NO]IO_TIME

ISN = ( number [- number] [,number [- number] ] ... )
    
```



```
[UN]LOCK = (number [,number]...)  
LOGGING = (keyword [,keyword]...)  
LOGIN_ID = number  
MGC = number  
NISNHQ = number  
NODE_ID = string  
OPTIONS = (keyword [,keyword]...)  
READ_PARALLEL_LIMITS = (records,blocks,total)  
RESET = keyword  
D [NO]RESPONSE_ABORT  
RESPONSE_CHECK = (number[-number][,number[-number]]...)  
SET_FILE_STATS = (number[-number][,number[-number]]...)  
SHUTDOWN  
STATUS = (keyword [,keyword]...)  
STOP = (number[-number][,number[-number]]...)  
TNAA = number  
TNAE = number  
TNAX = number  
TT = number  
USER_ID = string  
WRITE_LIMIT = [number]  
XA_RESPONSE_CHECK = (keyword [,keyword]...)
```

ABORT

ABORT

This function terminates the Adabas session immediately. All command processing is immediately stopped. The session is terminated abnormally with a pending AUTORESTART.

ABORT causes the following files to be written to the databases's default directory:

- The CSA dump file, which contains status information from the adabas nucleus. The name of the file is ADABAS.xxx.hh:mm:ss (UNIX), ADABAS.xxx.hh-mm-ss (Windows) or ADABAS-xxx-hh-mm-ss (OpenVMS), where xxx is the database ID and hh:mm:ss (or hh-mm-ss) is the time at which the file was created. ADAOPR can also display the same information that you can get for a running nucleus for a CSA dump file if you specify the CSA parameter.
- The SMP dump file, which contains some diagnostic information. The name of the file is SAGSMP.xxx.hh:mm:ss (UNIX), SAGSMP.xxx.hh-mm-ss (Windows) or SAGSMP-xxx-hh-mm-ss where xxx is the database ID and hh:mm:ss (or hh-mm-ss) is the time at which the file was created.

BFIO_PARALLEL_LIMIT

BFIO_PARALLEL_LIMIT = number

This function sets the number of parallel I/O requests by a buffer flush, allowing earlier processing of concurrent I/Os from other threads. A large buffer flush, for example, can cause the I/O queue to be very busy, and other I/Os (such as buffer pool read I/Os and WORK I/Os) can be enqueued for a long time, slowing down command throughput and possibly causing applications to stall if a buffer flush is active.

If BFIO_PARALLEL_LIMIT is specified, the buffer flush sets up the specified number of I/Os and waits until these have been processed before issuing the next packet. The maximum value for 'number' is defined by the Adabas system, If a value of 0 is specified, the number of buffer flush I/Os is unlimited.

CANCEL

CANCEL

This function terminates the Adabas session immediately. A BT command is issued for each active ET user and the session is terminated.

The communication link to the database is cut but the shared memory is still held. In this case, display functions are still possible with ADAOPR but parameter modification commands are no longer permitted.

CLEAR_FILE_STATS

```
CLEAR_FILE_STATS = (number [- number] [, number [- number] ] ... )
```

This function disables the collection of I/O statistics enabled by SET_FILE_STATS for the specified file(s).

CSA

```
CSA = string
```

'string' is a file specification of a file containing status information from an Adabas nucleus, a so-called CSA dump file. This file may be created by an ADAOPR ABORT function, by an abnormal termination of Adabas, or by response check trapping (refer to the RESPONSE_CHECK function for further information).

The following naming conventions are used for the file:

UNIX

```
ADABAS.xxx.hh:mm:ss
ADABAS.xxx.RSPyyy.hh:mm:ss
```

Windows

```
ADABAS.xxx.hh-mm-ss
ADABAS.xxx.RSPyyy.hh-mm-ss
```

OpenVMS

```
ADABAS-xxx-hh-mm-ss
ADABAS-xxx-RSPyyy.hh-mm-ss
```

(with the NORESPONSE_ABORT option set), where

- 'xxx' is the three digit database ID;
- 'yyy' is the trapped three digit response code;
- 'hh:mm:ss' is the time the file was created (UNIX),
- 'hh-mm-ss' is the time the file was created (Windows and OpenVMS)

For example, if the database ID is 5, and the file creation was initiated by a trapped response code 113, the file name will start with ADABAS.005.RSP113, and then the time of creating will be appended, e.g. ADABAS.005.RSP113.12:16:50 (UNIX) or ADABAS.005.RSP113.12-16-50 (Windows) or ADABAS-005-RSP113.12-16-50 (OpenVMS).

The file will be created in the directory that is pointed to by the environment variable/logical name ADA_CSA_DUMP. The default is the directory from which the nucleus was started. If a file with the same name already exists in this directory, it will be overwritten.

The DBID and CSA parameters are mutually exclusive.

DBID

DBID = number

This parameter selects the database to which all subsequent ADAOPR commands apply. Multiple DBIDs are supported within one session.

The DBID and CSA parameters are mutually exclusive.

Example:

```
adaopr: dbid=1
adaopr: shutdown
adaopr: dbid=2
adaopr: shutdown
adaopr: dbid=3
adaopr: shutdown
adaopr: quit
```

DISPLAY

DISPLAY = (keyword [,keyword]...)

This parameter displays various information during an Adabas session.

The following keywords can be used:

Keyword	Meaning
ACTIVITY	Database activities display.
BP_STATISTICS	Buffer pool statistics display.
COMMANDS	Command table display.
CQ	Command queue display.
DYNAMIC_PARAMETERS	Dynamic nucleus parameters display.
FILE_IO	File I/O display.
FP_STATISTICS	Format pool statistics display.
HIGH_WATER	High water marks display.
HQ	Hold queue display.
ICQ	Internal command queue display.

Keyword	Meaning
IO_TIMES	Container I/O times display.
PLOG_STATISTICS	Protection log statistics.
REPLICATIONS	Adabas - Adabas replications.
STATIC_PARAMETERS	Static nucleus parameters display.
TT	Thread table display.
UCB	Utility communication block.
UQ	User queue display.
UQ_FILES	User file list display.
UQ_FULL	Full information about user queue element.
UQ_TIME_LIMITS	User time limits display.

The following examples show the information produced by the various keywords, together with explanations of the information that is displayed.

Some of the following displays include percentages. The corresponding values are always truncated. An undefined value (divided by 0) is specified with "%*" and an overflow with "***%".

Example: DISPLAY=ACTIVITY

```

adaopr: display=activity

                ADANUC Version <version number>
                Database 76          Activity          on 22-JAN-2010 13:19:30

I/O Activity          Total    Throwbacks          Total
-----
Buffer Pool          5,440    Waiting for UQ context      87
WORK Read            728     Waiting for ISN             53
WORK Write           647     ET Sync                     0
PLOG Write           194     DWP Overflow                 0
NUCTMP              1,600
NUCSRT               531

Pool Hit Rate          Total    Interrupts          Current          Total
-----
Buffer Pool          99.6%    WP Space Wait           0                0
Format pool          98%
    
```

The information has the following meaning:

- I/O ACTIVITY shows the total numbers of:
 - physical buffer pool I/Os (physical read I/Os + physical write I/Os);

- read and write I/Os for WORK and PLOG.
- I/Os for NUCTMP and NUCSRT
- INTERRUPTS shows the current and total number of workpool space waits;
- POOL HIT RATE shows:
 - the buffer pool hit rate. This is the relationship between the logical read I/Os and the physical read I/Os. The buffer pool hit rate is calculated using the following formula:

$$\text{hit rate (in \%)} = \frac{((\text{logical read I/Os} - \text{physical read I/Os}) * 100)}{\text{logical read I/Os}}$$

- the format pool hit rate. This is the relationship between the number of format buffer requests (required FBs) and the required format buffers already translated in the format pool (translated FBs). The format pool hit rate is calculated using the following formula:

$$\text{hit rate (in \%)} = \frac{(\text{translated FBs} * 100)}{\text{required FBs}}$$

- THROWBACKS shows:
 - the number of commands waiting for session context because internal commands were running;
 - the number of commands waiting because ISNs are held by another user;
 - the number of commands waiting for ET synchronization;
 - the number of commands thrown back because of dynamic work pool overflow.

EXAMPLE: DISPLAY=BP_STATISTICS

```
adaopr: display=bp_statistics

          Database 76      ADANUC Version <version number>
                          Buffer Pool Statistics   on 19-JAN-2010 14:58:10

Buffer Pool Size      :   524,288,000

Pool Allocation
-----
Current      ( 14%) :   73,517,056   ASSO      :   11,909
Highwater    ( 14%) :   73,517,056   DATA     :         3
Internal     (  3%) :   15,728,640   WORK      :   2,766
Workpool     (  0%) :    881,664     NUCTMP    :         0
                                         NUCSRT    :         0

I/O Statistics
-----
Logical Reads      :         98,201   Buffer Flushes
Physical Reads     :         11,914   Total      :         1
                                         To Free Space :         0
```

```
Pool Hit Rate      :           87.9%
Physical Writes   :           2,711
Write Limit ( 5%): 26,214,400
Modified ( 2%):  11,329,536
```

The information is interpreted as follows:

- POOL ALLOCATION shows:
 - the size in bytes and percentage of the buffer pool that is currently in use;
 - the size in bytes and percentage of the buffer pool high water mark (see also the display for DISPLAY=HIGH_WATER).
- RABNs PRESENT shows:
 - the number of ASSO, DATA and WORK RABNs currently in the buffer pool.
- I/O STATISTICS shows:
 - the total number of logical and physical buffer pool read I/Os (both numbers are required in order to calculate the buffer pool hit rate);
 - the buffer pool hit rate (please refer to the example for DISPLAY=ACTIVITY for the buffer pool hit-rate formula);
 - the total number of physical buffer pool write I/Os.
- BUFFER FLUSHES shows:
 - the total number of buffer flushes;
 - the total number of buffer flushes that were made in order to get free space;
 - the size in bytes and percentage of the buffer pool write limit. If the modified bytes in the buffer pool reach this limit, an automatic buffer flush is started. The buffer pool write limit is automatically adjusted if not explicitly set in ADANUC or ADAOPR;
 - the size in bytes and percentage of the currently modified bytes in the buffer pool.

Example: DISPLAY=COMMANDS

```
adaopr: display=commands
          ADANUC Version <version number>
Database 76      Commands      on 19-JAN-2010 14:58:10

ADABAS Commands:          9,884

A1          892          L2          553          OP          25
BT          736          L3          1,124        RC          89
C1           40          L4           569        RE           0
C3           0           L5           420        RI           0
C5           10          L6           436        S1          1,511
```

CL	32	L9	456	S2	81
E1	1,006	LF	20	S4	12
ET	72	MC	0	S8	230
HI	0	N1	877	S9	50
L1	643	N2	0		

This command displays the total numbers of Adabas commands issued in the current session. For MC commands, the value displayed is the number of MC calls plus the number of single Adabas commands contained in the MC calls.

A read command that is issued while the multifetch option is set is counted as a single command.

Updates made by utilities are not included in the display.



Note: The command counts can be reset by ADAOPR RESET=COMMANDS.

Example: DISPLAY=CQ

```

adaopr: display=cq

                ADANUC Version <version number>
      Database 76      Command Queue      on 19-JAN-2010 14:58:10

No  Node Id  Login Id      ES Id  Cmd File  Status
--  -
1  sunxxx01  miller        21243798  E1  12  Ready to run
2  sunxxx01  jones         21231788  E1  12  Running
3  sunxxx01  smith         21230756  L2  12  Ready to run
4  sunxxx01  miller        21227398  S1  12  Running
5  sunxxx01  smith         21247630  S1  12  Running
6  sunxxx01  jones         21246388  A1  12  Ready to run
7  sunxxx01  jones         21219466  S1  11  Ready to run
8  sunxxx01  smith         21237160  L6  11  Ready to run
9  sunxxx01  miller        21246610  A1  11  Running
10 sunxxx01  miller        21246896  S1  11  Running
11 sunxxx01  dba           21244730  U0  0   Running

Selected: 11, Used: 11, Queue Size: 20
    
```

This display shows the current command-queue entries:

- NODE ID shows the node identification string.
- LOGIN ID shows the login user identification string;
- ES ID shows the environment-specific identification (for example, the process ID);
- CMD shows the command string;
- FILE shows the file number;

- STATUS shows the status of the command-queue entry.

The final line of the display shows how many command queue entries were selected according to the currently active selection criteria, and how many entries are used in total in the command queue.

The possible status values are shown in the following table:

Status	Meaning
Completed	Command processing completion;
Marked For Deletion	Command is marked for delete, user is no longer active;
New	Command is ready to be inserted in the scheduling queue;
Ready To Run	Placed in queue and ready for scheduling;
Running	Running in a thread (see DISPLAY=TT);
Waiting For Complex	Complex command is waiting to run;
Waiting For Et Sync	Waiting for ET synchronization;
Waiting For Group Commit	Waiting for group ET. No entry in thread table;
Waiting For Isn <isn>	Waiting for ISN in file shown in column "File" in the display. No entry in thread table;
Waiting For Space	Waiting for working space. No entry in thread table.
Waiting For Uqe	Waiting for user queue entry. The required entry is locked by an active internal command;



Note: The display may show command codes such as "U0", which are only used internally by Adabas (for example, during a utility run).
The "RUNNING" and "COMPLETED" values may differ even if the user has not specified an explicit selection criterion.

Example: DISPLAY=DYNAMIC_PARAMETERS

```

adaopr: display=dynamic_parameters

                ADANUC Version <version number>
Database 76      Dynamic Parameters      on 19-JAN-2010 14:58:10

Resources:      NISNHQ      :           100      WRITE_LIMIT:      -
Time Slices:    TNA        :           900      TNAX           :           900
                TNAE        :           900      TT             :           300
Group Commit:   MGC        :           50
Logging:        CLOG       : OFF
    
```

```
Read limits:          200, 10, 30
Response check with ABORT : 84,160,164-182,243,251-252
```

This display shows the current values of the dynamic nucleus parameters.

Example: DISPLAY=FILE_IO

```
adaopr: display=file_io

                ADANUC Version <version number>
      Database 76      File I/O      on 19-JAN-2010 14:58:10

      File      Logical Reads      Physical Hit      Writes
      ----      -
      11      145,341      180      99%      2,869
      12      99,070      148      99%      2,149
```

This display shows the logical and physical reads, their hit rate and the writes the buffer pool manager has made for every file since the file I/O statistics for the file in question were enabled (ADAOPR SET_FILE_STATS) - files for which the I/O statistics have not been enabled or for which no I/Os were performed are not displayed.

 **Notes:**

1. The formula for the hit rate value is given in the description of DISPLAY=ACTIVITY.
2. A write operation is only counted if the block was not yet marked as modified. This means that the physical write I/Os either already done in a previous buffer flush or still pending to be performed in the next buffer flush are counted.

Example: DISPLAY=FP_STATISTICS

```
adaopr: display=fp_statistics

                ADANUC Version <version number>
      Database 76      Format Pool Statistics      on 19-JAN-2010 14:58:10

Maximum Local Pool Size:      251,656
Maximum Global Pool Size:      251,656

Pool Allocation      Pool Contents
-----
```

```

Local Current ( 22%) :          57,540   Local Format Buffers:          162
Local Highwater ( 27%) :         70,000   Global Format Buffers:           1

Global Current (  0%) :             84
Global Highwater (  0%) :            84
    
```

Pool Statistics	Local	Global
-----	-----	-----
Scans	11,780	3
Hits	11,547	2
Hit Rate	98%	66%
Replacements	0	0
Overflows	0	0

This display shows the format pool statistics:

- POOL ALLOCATION shows:
 - the size in bytes and percentage of the local and global format pools that are currently in use;
 - the size in bytes and percentage of the local and global format pool high water marks.
- POOL STATISTICS shows:
 - the total number of scans and hits of valid format buffers in the format pool (both numbers are required in order to calculate the format pool hit rate);
 - the format pool hit rate (please refer to the example DISPLAY=ACTIVITY for the format pool hit-rate formula);
 - the total number of valid format buffers that are overwritten in the format pool (replacements).
 - Overflows. This is the number of times that a format buffer exceeded the format pool size, resulting each time in a response 42.
- POOL CONTENTS shows:
 - the number of valid local format buffers in the format pool;
 - the number of valid global format buffers in the format pool.

Example: DISPLAY=HIGH_WATER

```

adaopr db=076 display=high_water

          ADANUC Version <Version number>
Database 76          High Water Marks          on 19-JAN-2010 10:52:08

Area/Entry          Size          In Use  High Water  %          Date/Time
-----
User Queue          100             14           14    14 19-JAN-2010 10:50:48
Command Queue          -              11           11    - 19-JAN-2010 10:50:54
    
```

Hold Queue	-	100	100	-	19-JAN-2010	10:46:04
Client Queue	50	1	5	5	19-JAN-2010	10:46:13
HQ User Limit	100	-	83	83	19-JAN-2010	10:47:26
Threads	6	6	6	100	19-JAN-2010	10:45:54
Workpool	16,777,216	0	4,194,320	25	19-JAN-2010	10:46:04
ISN Sort	2,097,152	-	2,095,784	99	19-JAN-2010	10:51:58
Complex Search	2,097,152	-	126,324	6	19-JAN-2010	10:45:54
Attached Buffer	98,304	67,584	67,584	68	19-JAN-2010	10:52:17
Buffer Pool	6,291,456	1,627,136	1,627,136	25	19-JAN-2010	10:52:08
Protection Area	1,000					
Active Area	300	-	146	48	19-JAN-2010	10:50:58
Group Commit	50	0	1	2	19-JAN-2010	10:45:46
Transaction Time	300	-	158	52	19-JAN-2010	10:50:28

This display shows the high water marks for the current session:

- **SIZE** shows the size in bytes of pools and buffers. For queues, threads and hold queue user limit, it shows the number of entries.
- **IN USE** shows the size in bytes or number of entries currently in use.
- **HIGH WATER** shows the maximum quantity required simultaneously for the given area/entry.
- **%** shows the relationship between the high water mark and the size. If the high water mark exceeds the size, the value in this column can be larger than 100 %. For example, this can occur if the value is decreased by ADAOPR, or if the original area has been dynamically increased. This is normal Adabas behaviour, and no changes of Adabas parameters are required.
- **DATE/TIME** shows the date/time at which the high water mark occurred for the first time. There is no output in this column if the high water mark is 0.

The entries in the column AREA/ENTRY correspond to the ADANUC parameters NU (user queue), NCL (client queue), NISNHQ (hold queue user limit), NT (threads), LWP (workpool), LBP (buffer pool), LAB (attached buffer), MGC (group commit), TT (transaction time). The hold queue and the command queue have no predefined size and are increased dynamically if required.

The entry "ACTIVE AREA" is the largest part of WORK part 1 that can be used by a single transaction. If a transaction's protection information spans more space than allowed by "Active Area", it receives a response 9 (LP), the nucleus displays a PLOVFL message and a value of more than 100 in the "%" column of the highwater display.

Users who have set user-specific timeout values in their OP call are not included in the values for Transaction Time.



- Note:**
1. Values for Attached Buffer and Command Queue are not displayed correctly if the nucleus cannot be contacted by ADAOPR (for example, if the ADAOPR parameter CSA is used).
 2. Threads are used in a round-robin manner. Therefore, the high water mark for threads will be the same as the value shown in the Size column in most cases.

Example: DISPLAY=HQ

```

adaopr: file=11, display=hq

          ADANUC Version <version number>
Database 76          Hold Queue          on 19-JAN-2010 14:58:10

  Id Node Id  Login Id      ES Id User Id  File          ISN Locks  Flg
  ---- -
  15 sunxxx01 miller      6974 *adatst   11          2,222  X      M
  19 sunxxx01 smith       7056 *adatst   11           2      X

```

Selected: 2, Used: 8, Queue Size: 160

This display shows the current hold-queue entries:

- ID shows the internal user identification of the user holding the ISN;
- NODE ID shows the node identification string. The local node is represented by an empty string;
- LOGIN ID shows the login user identification string;
- ES ID shows the environment-specific identification (for example, process ID);
- USER ID shows the user identification. Adabas utilities use the utility name preceded by an asterisk as the USER ID;
- FILE shows the number of the Adabas file in which the ISN is located;
- ISN shows the number of the ISN in hold;
- LOCKS shows the kind of lock for the ISN, where X = exclusive lock , S = shared lock.



Note: S is displayed for shared locks starting with Adabas version 6.3 SP 1; in previous releases R is displayed.

- An M for FLG indicates that the record has been modified.

The final line of the display shows how many hold queue entries were selected according to the currently active selection criteria, and how many entries are used in total.

Entries are displayed in unsorted sequence.

Example: DISPLAY=ICQ

```

adaopr: display=icq

                ADANUC Version <version number>
Database 76      Internal Command Queue   on 19-JAN-2010 14:58:10

      Id  Node Id  Login Id      ES Id  Command  Status
      --  -
00000002          *system  00000000  SHUT    Running

Selected: 1, Used: 1, Queue Size: 101
    
```

This display shows the internal command queue:

Command	Meaning
AR	Autorestart
BT	Back out transaction
BTCL	Back out open transaction and close user
CANCEL	Cancel nucleus
DELUQE	Release file list and delete user queue element
ETSYNC	Start an ET-SYNC status check after a global transaction has received a timeout
SHUT	Shut down nucleus
STOP	STOP from ADAOPR
TIMEOUT	Non-activity timeout

The status of internal commands can be READY TO RUN, RUNNING, WAITING FOR ET SYNC or WAITING FOR UQE.

The final line of the display shows how many internal command queue entries were selected according to the currently active selection criteria, and how many entries are used in total.

Example: DISPLAY=IO_TIMES

```

adaopr: display=io_times

                ADANUC Version <version number>
Database 76      IO Statistics           on 19-NOV-2010 12:16:48

      Number of IOs      Maximum IO time      Average IO time
      -----
ASSO Read      :           735574           14397           1
    
```

ASSO Write	:	12136	2	1
DATA Read	:	2023257	13910	1
DATA Write	:	444	1	1
WORK Read	:	4	1	1
WORK Write	:	660	2	1
NUCSRT Read	:	4060	940	1
NUCSRT Write	:	4060	1	0
NUCTMP Read	:	30	1	1
NUCTMP Write	:	896	1	1

The number of IOs shows the number of physical read and write I/O accesses to ASSO, DATA, WORK, NUCSRT and NUCTMP.

The maximum IO time shows the maximum duration of a single I/O read and write access to ASSO, DATA, WORK, NUCSRT and NUCTMP in microseconds.

The average IO time shows the average time of a single I/O access to ASSO, DATA, WORK, NUCSRT and NUCTMP in microseconds.

Logging of I/O times is only available if ADAOPR IO_TIME is enabled..

Example: DISPLAY=PLOG_STATISTICS

```

adaopr: display=plog_statistics

                ADANUC Version <version number>
Database 76      PLOG Statistics      on 19-JAN-2010 14:59:41

PLOG Environment
-----
NUCPLG      (active) : /FS/fsxxxx/sag/ada6180102/ada/db076/NUCPLG

Active PLOG
-----
Session Number      : 37
Extent              : 2

Active Since        : 19-JAN-2010 14:59:41
Duration            : 00:00:01

Allocated Space     : 24,683 KB
Used Space ( 0%)   : 32 KB
Average Growth Rate : 115,200 KB/h

```

Example: DISPLAY=REPLICATIONS

```

adaopr: display=replications
                ADANUC Version <version number>
Database 34      Replications      on 27-JUN-2012 09:47:48

ID  From FNR  To DB  To FNR  Status      Remark
---  ---  ---  ---  ---  ---
  1   111    37   111  Inactive
 86   86    37   86   Active


      2 transactions pending:
      -----

To DB  Transactions
-----  -----
  37           2

      5 commands pending:
      -----

From FNR      Commands
-----  -----
  86           5
 111           0
    
```

This display shows the Adabas - Adabas replications currently defined. This is only relevant for customers who are using the Adabas Event Replicator with Adabas - Adabas replication.

 **Note:** Replications to other replication targets, for example SQL databases, are not displayed. Such replications can only be displayed with the administration tools of the event replication.

The display shows the following information:

- "ID" is the ID of the replication that is also used in the replication administration.
- "From FNR" is the file number of the file to be replicated to another Adabas file.
- "To DB" and "To FNR" are the database ID and file number of the target file for the replication.
- "Status" can have the following values and meanings:

Status	Meaning
Inactive	Currently no data are replicated to the target file, and at the moment no activities have been made to initiate the replication.
Prepare	This status indicates that it is planned to perform the initial state processing for the replication. This status is the prerequisite for creating a backup of files to be replicated via ADABCK with parameter REPLICATION.
Initialization	This status indicates that ADABCK with parameter REPLICATION is running and creates a backup containing the initial state of files to be replicated.
Recording	Adabas is currently recording the update transactions within the replication command file and the replication transaction file, but currently does not replicate the update operations to the target database.
Active	The replication is active; all modifications of the source file are replicated to the target file.
Error	An unexpected error occurred during replication. In order to continue replication, a new initial state processing is required.

- “Pending Transactions” is the number of transactions that have not yet been replicated to the target file.



Notes:

1. The number contains both transactions that have already been committed but not yet replicated to the target database, and transactions that are still open and which can only be replicated after an end of transaction.
 2. If a transaction contains commands to be replicated to more than one target database, the transaction is counted only once, independent of the number of target databases. Therefore the total number of pending transactions can be smaller than the sum of the transactions for the different target databases.
- “Pending Commands” is the number of commands that have not yet replicated to the target file.



Notes:

1. The number contains both commands belonging to transactions that have already been committed but not yet replicated to the target database, and commands belonging to transactions that are still open and which can only be replicated after an end of transaction.
2. If a file is replicated to more than one target file, database modification commands of the source file are counted only once, independent of the number of target files to which a command has to be replicated.

If ADAOPR DISPLAY=REPLICATIONS is executed in non-interactive mode, ADAOPR returns one of the following exit status values:

Value	Meaning
0	At least one replication has been defined, and no replication is in status Error.
12	There is a replication in status Error.
15	Replication has not been activated, or no replication has been defined.

Example: DISPLAY=STATIC_PARAMETERS

```
adaopr: display=static_parameters

                ADANUC Version <version number>
      Database 76      Static Parameters      on 19-JAN-2010 14:58:10

Resources:      LAB      :      98,304      NT      :      6
                LBP      :      6,291,456    NU      :      100
                LWP      :      170,000    NCL      :      50

Logging:        PLOG, BI
Options:        TRUNCATION

Cloglayout:    5
```

This display shows the static nucleus parameters.

Example: DISPLAY=TT

```
adaopr: display=tt

                ADANUC Version <version number>
      Database 76      Thread Table      on 19-JAN-2010 14:58:10

No      Cmd Count  File  Cmd  Status
--      -
1       10,566    0     U0   Update , active
2       21,475    11    S1   Complex, active
3       10,382    12    S1   Simple , active
4       2,516     12    S1   Simple , active
5       3,782     11    A1   Update , active
6       1,713     12    E1   Update , active
```

This display shows the entries in the thread table. The number of displayed entries is simultaneously the high water mark for threads.

- **CMD COUNT** shows the total number of Adabas commands processed from the corresponding thread context. The sum of these counts will normally differ from the sum shown by **DISPLAY=COMMANDS**, because internal commands are also counted.
- **FILE** shows the file number of the Adabas command that is currently being processed from the corresponding thread context. The file number is 0 if the corresponding thread context is not active, or if the command is a global one which is not linked to a particular file.
- **CMD** shows the command string of the Adabas command that is currently being processed from the corresponding thread context. There is no output in this column if the corresponding thread context is not active.
- **STATUS** shows the command type and the status of the corresponding thread context.

Possible command types are:

- Update
- Simple
- Complex

Possible entries for the thread status are shown in the following table:

Status	Meaning
free	available for allocation
ready	ready to run
active	running
waiting for io <rabn>/<block type>	waiting for I/o completion of block <rabn>
waiting for <rabn>/<block type>	waiting for access/update synchronization of block <rabn>
waiting for space <size> bytes	waiting for <size> bytes of work pool space

The block type can be ASSO, DATA, WORK, FILE or PLOG.

Example: DISPLAY=UCB

```

adaopr: display=ucb

                ADANUC Version <version number>
Database 76           UCB           on 19-JAN-2010 14:59:45

Date/Time           Entry Id  Utility  Mode Files
-----
19-JAN-2010 14:59:41      42    adaopr  UTO   13
    
```

This display shows the utility communication block.

- DATE/TIME shows the date and time on which the given files were locked.
- ENTRY ID shows the allocated identification of the entry.
- UTILITY shows the name of the utility.
- MODE shows the mode in which the files are being accessed. The possibilities are:
 - ACC open for access
 - UPD open for update
 - EXU open for exclusive update (parallel access allowed)
 - UTO open for utilities only
 - UTI open for exclusive access (no parallel access or update allowed)
- Files shows the file numbers of the files that are locked.

Example: DISPLAY=UQ

```

adaopr: display=uq

                ADANUC Version <version number>
Database 76      User Queue          on 19-JAN-2010 14:58:10

  Id  Node Id  Login Id      ES Id  User Id  Type  Status
  --  -
  26  sunxxx01 dba          4473  *adaopr  UT
  23  sunxxx01 smith        3075           ET      E
  20  sunxxx01 jones        3178           ET      I
  19  sunxxx01 jones        1946           ET      IE
  18  sunxxx01 smith        4689           ET
  16  sunxxx01 smith        4661           ET
  17  sunxxx01 jones        4638  #####      T
  14  sunxxx01 miller      4379           ET      R
  13  sunxxx01 dba          3967  *adatst  AC
  12  sunxxx01 dba          3651  *adatst  EX,ET    E
  11  sunxxx01 dba          4025  DBADMIN  EX      RU

Selected: 11, Used: 11, Queue Size: 100
    
```

This display shows the current user queue entries.

- ID shows the internal user identification;
- NODE ID shows the node identification string;
- LOGIN ID shows the login identification string;
- USER ID shows the user identification;

- TYPE shows the user type:
 - AC access only user
 - ET ET user
 - EX exclusive update user
 - EX,ET exclusive update user with ET logic
 - UT utility user.
- STATUS shows the status of the user:
 - E user at ET status
 - G global timeout (XA)
 - I user session started with an implicit OPEN
 - P pending ET (XA)
 - R restricted file list
 - T user has received a time-out
 - U user specific timeout interval value

The final line of the display shows how many user queue entries were selected according to the currently active selection criteria, and how many entries are used in total.

Example: DISPLAY=UQ_FILES

```
adaopr: display=uq_files

                                ADANUC Version <version number>
Database 76                      User Files                on 19-JAN-2010 14:58:10

  Id  Type  Mode  Files
  --  -
  26  UT
  23  ET    UPD  11-12
  20  ET    UPD  11-12
  19  ET    UPD  11-12
  18  ET    UPD  11-12
  16  ET    UPD  11-12
  14  ET    UPD  11-12
  13  AC
  12  EX,ET EXU   14
  11  EX    ACC   11
      EXU   13

Selected: 10, Used: 11, Queue Size: 100
```

This display shows the file lists for active users.

- ID shows the internal user identification;
- TYPE shows the user type (please refer to the DISPLAY=UQ example for more information).
- MODE shows the mode in which the files are being accessed:
 - ACC open for access
 - EXF open for exclusive access (no parallel access or update allowed)
 - EXU open for exclusive update (parallel access allowed)
 - UPD open for update
 - UTI open for exclusive access (no parallel access or update allowed)
 - UTO open for utilities only
- FILES shows the Adabas file list of the user entry. If the list is too large to be displayed in one line, several lines will be used: file numbers are not omitted.

The final line of the display shows how many user queue entries were selected according to the currently active selection criteria, and how many entries are used in total.

Example: DISPLAY=UQ_FULL

```

adaopr: file=13, display=uq_full

          ADANUC Version <version number>
Database 34      Full User Queue Entry   on 9-MAR-2010 11:20:43

User Entry:  Id       : 7                ES Id       : 14251
             Node Id  : sunxxx05         Login Id    : dba
             User Id   : *adaopr

             User Type : UT              User Status :

Time Stamps: Session Start : 9-MAR-2010 11:20:43
             Trans. Start  :
             Last Activity :

Time Limits: TT          :                0   TNA          :                0

Resources:  ISN Lists   :                0   ISNs Held     :                0
             Open Files  :                0

Activity:   ADABAS Calls :                1   Transactions  :                0

Settings:  User Encoding : UTF-8
-----
    
```

```

User Entry: Id      : 6          ES Id      : 13462
            Node Id   : sunxxx05  Login Id   : smith
            User Id   : SMITH001

            User Type  : ET          User Status :

Time Stamps: Session Start : 9-MAR-2010 11:20:23
            Trans. Start  : 9-MAR-2010 11:20:23
            Last Activity  : 9-MAR-2010 11:20:23

Time Limits: TT          :          3,600  TNA          :          900

Resources:  ISN Lists    :          0  ISNs Held    :          3
            Open Files   :          1

Activity:   ADABAS Calls :          21  Transactions :          1

Settings:  User Encoding : UTF-8
            Time zone    : Europe/Berlin
    
```

This display shows detailed information about user queue elements.

Example: DISPLAY=UQ_TIME_LIMITS

```
adaopr: display=uq_time_limits
```

```

                ADANUC Version <version number>
      Database 76      User Time Limits      on 19-JAN-2010 14:58:10

TNAI Interval   :          00:15:00  TNAX Interval   :          00:15:00
TNAE Interval   :          00:15:00  TT   Interval   :          00:05:00

      Id St Limit   Timeout Interval   Remaining Time   Start Date/Time
      --- -- ---
      23  TNAE      00:15:00      00:15:00  19-JAN-2010 14:58:10
           TT      00:05:00
      22  TNAE      00:15:00      00:15:00  19-JAN-2010 14:58:10
           TT      00:05:00
      21  TNAE      00:15:00      00:15:00  19-JAN-2010 14:58:10
           TT      00:05:00
      20  TNAE      00:15:00      00:15:00  19-JAN-2010 14:58:10
           TT      00:05:00
      19  TNAE      00:15:00      00:15:00  19-JAN-2010 14:58:10
           TT      00:05:00
      18  TNAE      00:15:00      00:15:00  19-JAN-2010 14:58:10
           TT      00:04:50
      17  TNAI      00:15:00      00:15:00  19-JAN-2010 14:58:10
      16  TNAE      00:15:00      00:15:00  19-JAN-2010 14:58:10
           TT      00:05:00
    
```

14	TNAE	00:15:00	00:15:00	19-JAN-2010	14:58:10
	TT	00:05:00	00:05:00	19-JAN-2010	14:58:10
13	TNAA	00:15:00	00:10:01	19-JAN-2010	14:53:11
12	TNAE	00:15:00	00:10:01	19-JAN-2010	14:53:11
	TT	00:05:00			
11	U TNAX	00:40:00	00:34:57	19-JAN-2010	14:53:07

Selected: 12, Used: 14, Queue Size: 100

This display shows the current timeout limits for the user queue entries.

- ID shows the internal user identification;
- ST shows the status of the entry. Possible values are:
 - U user specific timeout value
 - T a timeout is pending, response 9 has not been collected yet by the client.
- LIMIT describes the timeout type;
- TIMEOUT INTERVAL shows the current active timeout intervals.
- REMAINING TIME shows the amount of time remaining until the next timeout mark.
- START DATE/TIME shows the starting date and time of the entry.

The final line of the display shows how many user queue entries were selected according to the currently active selection criteria, and how many entries are used in total.

ES_ID

```
ES_ID = number
```

This function influences the output of the DISPLAY options CQ, HQ, ICQ, UQ, UQ_FILES, UQ_FULL, UQ_TIME_LIMITS. Only entries with the specified environment-specific ID are displayed.

[NO]ET_SYNC

```
[NO]ET_SYNC
```

This option controls the behaviour of the FEOF=PLOG function. It must be specified before specifying FEOF=PLOG. Refer to the FEOF=PLOG function for more information.

The default is NOET_SYNC.

EXT_BACKUP

```
EXT_BACKUP = [PREPARE | CONTINUE | ABORT]
```

This function is used to backup a database using an external backup system, which can be considerably faster with very large databases than using ADABCK.

The keyword PREPARE prepares the database for backup. During this phase, the following restrictions apply:

- new transactions will be stalled
- no updating utility functions (e.g. ADADBDM) can be started
- the functions SHUTDOWN, CANCEL, LOCK, STOPUSER, UNLOCK and FEOF=PLOG are not permitted once the EXT_BACKUP = PREPARE call has finished processing
- all non-activity timeout checks are disabled

The keyword CONTINUE is used to resume normal database operations following completion of the external backup. The following actions are performed:

- open a new PLOG with a new session number
- re-enable non-activity timeout checks
- re-enable update utilities
- wake up all waiting users (start of new transactions)

The keyword ABORT is used to abort an external backup for which a PREPARE has already been issued. In this case, the PLOG isn't switched and no checkpoint is written.

Example

The following scenario shows a backup and restore using a third-party backup tool (tar is not a real alternative, and is used for demonstration purposes only):

Dumping the database

```
% adaopr dbid=37 ext_backup=prepare
%ADAOPR-I-STARTED,      14-NOV-2012 16:18:30, T-Version 6.3.99.00 (Solaris 64Bit)

Database 37, startup at 14-NOV-2012 16:18:10
ADANUC T-Version 6.3.99.00, PID 15245

%ADAOPR-I-EXTBPREP, preparing for external backup, 14-NOV-2012 16:18:30

%ADAOPR-I-TERMINATED,  14-NOV-2012 16:18:30, elapsed time: 00:00:00
% tar cvf $BACKUPDIR/backup.tar ASSO* DATA* # external dump
<external backup output>
% adaopr dbid=1 ext_backup=continue
%ADAOPR-I-STARTED,      14-NOV-2012 16:18:45, T-Version 6.3.99.00 (Solaris 64Bit)
```

```
Database 37, startup at 14-NOV-2012 16:18:10
ADANUC T-Version 6.3.99.00, PID 15245
During ET Sync (phase 2), for external backup

%ADAOPR-I-EXTBCONT, continue from external backup, 14-NOV-2012 16:18:45

%ADAOPR-I-TERMINATED, 14-NOV-2012 16:18:45, elapsed time: 00:00:00
```

Restoring and recovering the database

```
% tar xvf $BACKUPDIR/backup.tar # external restore
% adastart 37
% setenv RECPLG plog.0017 # Set RECPLG for ADAREC (C shell)
% adarec dbid=37 regenerate=* plog=17
```

The external backup is logged in the ADANUC log file

```
%ADANUC-I-DBSTART, Database 37, session 16 started, 14-NOV-2012 16:17:10
%ADANUC-I-EXTBPREP, preparing for external backup, 14-NOV-2012 16:18:30
%ADANUC-I-DBSTART, Database 37, session 17 started, 14-NOV-2012 16:18:45
%ADANUC-I-PLOGCRE, plog NUCPLG, file 'plogs/plog.0017' created
%ADANUC-I-EXTBCONT, continue from external backup, 14-NOV-2012 16:18:45
```

FEOF

```
FEOF = (keyword [,keyword])
```

In accordance with the keywords specified, the log file(s) are closed and a new log file is created.

Keyword	Meaning
CLOG	closes command log file.
PLOG	closes protection log file. This depends on the [NO]ET_SYNC option: If ET_SYNC is specified: The current protection log file (PLOG) will be closed when all currently active ET logic users have come to ET status, and a new PLOG is created with the next higher PLOG number. If NOET_SYNC is specified: The current PLOG extent will be closed when the next PLOG block is written, and a new extent of the same PLOG will be created. The PLOG number is not incremented and the users do not have to be synchronized at ET status. Example (PLOG is on raw device): if the current PLOG is PLG.5, then the command "adaopr db=1 et_sync feof=plog" results in the PLOGs PLG.5 and PLG.6, whereas the command "adaopr db=1 noet_sync feof=plog" results in the PLOGs PLG.5(1) and PLG.5(2). Example (PLOG is in file system): if the current PLOG is NUCPLG.0005, then the command "adaopr db=1 et_sync feof=plog" results in the PLOGs NUCPLG.0005 and NUCPLG.0006, whereas the

Keyword	Meaning
	command "adaopr db=1 noet_sync feof=plog" results in the PLOGs NUCPLG.0005(1) and NUCPLG.0005(2).

The FEOF command will be rejected if the keyword PLOG is used while running ADAREC RE-GENERATE = * (see *ADAREC* for more detailed information).

FILE

FILE = number

This influences the output of the DISPLAY options HQ, ICQ, UQ, UQ_FILES, UQ_FULL and UQ_TIME_LIMITS. Only entries related to the specified file number are displayed.

FREE_CLQ

FREE_CLQ

Normally, obsolete entries in the client queue are released automatically when the client queue is full. With ADAOPR FREE_CLQ, you can enforce the client queue cleanup before the client queue becomes full.

ID

ID = number

This function influences the output of the DISPLAY options CQ, HQ, ICQ, UQ, UQ_FILES, UQ_FULL and UQ_TIME_LIMITS. Only entries related to the specified internal ID are displayed.

[NO]IO_TIME

[NO]IO_TIME

The parameter IO_TIME enables logging of the I/O times for the ASSO, DATA, WORK, NUCSRT and NUCTMP containers. The times are given in microseconds.

If logging of I/O times is already enabled, enabling it again resets all I/O time and I/O counter statistics.

The default is NOIO_TIME.

ISN

ISN = (number [- number] [,number [- number]] ...)

This function influences the output of the DISPLAY option HQ. Only entries related to the specified ISNs are displayed.

[UN]LOCK

[UN]LOCK = (number [,number]...)

The file(s) specified by the file number(s) are locked or unlocked. The specified files are locked for all non-utility use; Adabas utilities can use the file(s) normally.

For users who have one or more files to be locked in their open file list, a STOP <user-ID> command is issued internally. Refer to the description of the ADAOPR STOP parameter for more details.



Notes:

1. You can also lock non-existent file numbers; if you subsequently create files with these numbers, the files are locked.
2. Locking a LOB file does not prevent users from storing LOB data in the LOB file; disabling the access to LOB data in the LOB file is part of locking the corresponding base file. Locking a LOB file is only useful if you plan to use this file number for a base file at some time in the future.

LOGGING

LOGGING = (keyword [,keyword]...)

This parameter starts command logging for the buffers specified in the list of keywords.

The following keywords can be used:

Keyword	Meaning
CB	Enables logging of control block
FB	Enables logging of format buffers
RB	Enables logging of record buffers
SB	Enables logging of search buffer
VB	Enables logging of value buffer
IB	Enables logging of ISN buffer
ABD	Enables logging of Adabas buffer descriptions
IO	Enables I/O list logging
OFF	Stops logging of all buffers, but keeps the command log file open

If the nucleus was started with LOGGING=OFF and buffer logging is requested, then the CLOG file will be created.

LOGIN_ID

```
LOGIN_ID = number
```

This function influences the output of the DISPLAY options CQ, HQ, ICQ, UQ, UQ_FILES, UQ_FULL and UQ_TIME_LIMITS. Only entries with the specified login ID are displayed.

MGC

```
MGC = number
```

This parameter specifies the maximum group-commit count. This defines the maximum limit of ET command grouping before the PLOG buffers are written back to disk. If this limit is reached, and the final IO has been performed, all remaining users will be posted.

If the specified value is less than the corresponding high-water value, a warning is issued.

The minimum value is 1, the maximum value is 500.

NISNHQ

```
NISNHQ = number
```

This parameter specifies the maximum number of records that can be placed into hold at any time by a single user.

If the specified value is less than the corresponding high-water value, a warning is issued.

The minimum value is 0, where 0 means unlimited.

NODE_ID

```
NODE_ID = string
```

This function influences the output of the DISPLAY options CQ, HQ, ICQ, UQ, UQ_FILES, UQ_FULL and UQ_TIME_LIMITS. Only entries for the specified node are displayed.

OPTIONS

OPTIONS = (keyword[, keyword])

The available keywords are:

Keyword	Meaning
[NO]LOCAL_UTILITIES	If LOCAL_UTILITIES is specified, the nucleus rejects all remote utility calls, i.e. the Adabas utilities cannot be run from a remote node across a network.
[NO]UTILITIES_ONLY	If UTILITIES_ONLY is selected, all calls other than for utilities will be rejected. Note, however, that this restriction only applies to new users; users who were already active when OPTIONS=UTILITIES_ONLY was specified can continue processing normally. If you want exclusive utility control over files or the entire database, use the LOCK function of ADAOPR instead.

These options can be disabled using the prefix 'NO', e.g. OPTIONS=NOUTILITIES_ONLY.

READ_PARALLEL_LIMITS

READ_PARALLEL_LIMITS = (records, blocks, total)

This parameter is used to modify the nucleus parameter READ_PARALLEL_LIMITS. Please refer to the description in ADANUC for further information.

RESET

RESET = keyword

RESET=HIGH_WATER resets the high water mark values to the value currently in use.

RESET=COMMANDS resets the command counts displayed by ADAOPR DISPLAY=COMMANDS.

[NO]RESPONSE_ABORT

[NO]RESPONSE_ABORT

If response checking is enabled with the RESPONSE_CHECK parameter of ADAOPR, the RESPONSE_ABORT option determines whether the nucleus aborts when one of the specified responses occurs (RESPONSE_ABORT), or whether the nucleus resumes operation and a database section file is written to disk (NORESPONSE_ABORT).

The setting of the [NO]RESPONSE_ABORT option can only be changed before the RESPONSE_CHECK parameter. The same applies for XA_RESPONSE_CHECK (not on OpenVMS).

The default is NORESPONSE_ABORT.

Refer to the RESPONSE_CHECK parameter for further information.

RESPONSE_CHECK

```
RESPONSE_CHECK = [(number[-number][,number[-number]]...)]
```

This function enables the DBA to gather information if one of a list of Adabas response codes occurs. The information written may be used to analyze possible problems in the database's operation. If a response check for an Adabas response code is enabled, the database section file is written to disk if this response code occurs.

Depending on the setting of the RESPONSE_ABORT option, the nucleus either aborts or continues operation:

- if the RESPONSE_ABORT option is set, the database section file (Adabas.xxx.hh:mm:ss [UNIX], or Adabas.xxx.hh-mm-ss [Windows] or Adabas-xxx-hh-mm-ss [OpenVMS]) is written to the database's default directory. The database section file is also called the CSA dump file. See ADANUC and the environment variable ADA_CSA_DUMP for more information.

When the CSA dump file is written, the SMP dump file is also written (UNIX platforms only); the name of the SMP dump file is SMPPOS.APP:hh:mm:ss.

- if the NORESPONSE_ABORT option is set (default setting), the nucleus continues running and the database section file (Adabas.xxx.RSPyyy.hh:mm:ss [UNIX], or Adabas.xxx.RSPyyy.hh-mm-ss [Windows] or Adabas-xxx-RSPyyy-hh-mm-ss [OpenVMS]) is written to the database's default directory. See ADANUC and the environment variable ADA_CSA_DUMP for more information. Only one dump is generated for one response code; if a response code occurs, the RESPONSE_CHECK option is deactivated for that response code, but if it has been activated for other response codes, it remains active for the other response codes.

Refer to the RESPONSE_ABORT action for further information.

By default, no response is trapped and the nucleus continues operation.

To disable response trapping, use "RESPONSE_CHECK =" without arguments.



Note: Some response codes can be generated outside the nucleus (e.g. by ADALNK and ENTIRE NET-WORK). If this happens, they cannot be trapped by Adabas. The response codes in question for ADALNK are: 9, 17, 22, 40, 146-154, 241, 252, 255.

SET_FILE_STATS

```
SET_FILE_STATS = [(number[-number][,number[-number]]...)]
```

This function enables the file level I/O statistics for the specified files. Only these files will be displayed by DISPLAY = FILE_IO.

SHUTDOWN

```
SHUTDOWN
```

This function terminates the Adabas session normally. No new users are accepted. ET-user updating is continued until the end of the current transaction for each user. When all update activity has ended as described above, the Adabas session is terminated.

The communication link to the database is cut but the shared memory is still held. In this case, display functions are still possible with ADAOPR but parameter modification commands are no longer permitted.

STATUS

```
STATUS = (keyword [,keyword] ,... )
```

This function influences the output of the DISPLAY parameter options HQ, ICQ, UQ, UQ_FILES, UQ_TIME_LIMITS, UQ_FULL. Only entries in the specified state will be displayed.

The valid keywords are:

Keyword	Meaning
[NO]TIMEOUT	User without or with "T" status.
[NO]ET_STATUS	Users at "ET" status with open transactions.
[NO]PENDING_ET	Users without or with "P" status.

STOP

```
STOP = (number[-number][,number[-number]]...)
```

This parameter terminates the user with the specified ID (internal identification). The ID can be retrieved with DISPLAY = UQ.

The message "Stop handling started for n users" is displayed, where "n" is the number of users who will be stopped.



Note: Utilities cannot always be stopped in this way.

The actions that Adabas takes when a user is stopped depend on the user type, and also whether the nucleus requires an explicit OP (open) command at the start of a user session, as shown in the following table.

The abbreviation SUQE used in the table means "Stop user queue element", and consists of the following actions: release all Command IDs, scratch the file list, scratch the user ID, scratch the user type, set response 9 for the next call.

User Type	Adabas Actions without ADANUC OPTIONS=OPEN_REQUIRED	Adabas Actions with ADANUC OPTIONS=OPEN_REQUIRED
ACC	For ID user: SUQE For non-ID user: session closed	session closed
ET, ET Status	For ID user: SUQE For non-ID user: session closed	session closed
ET, no ET Status	Backout transaction, SUQE	Backout transaction, session closed
EX	SUQE, CLSE checkpoint	session closed
EX, ET with ET status	SUQE, CLSE checkpoint	session closed
EX, ET, no ET status	Backout transaction, SUQE, CLSE checkpoint	Backout transaction, session closed
UT	session closed	session closed

If a STOP command is issued for a user while running

```
ADAREC REGENERATE = *
```

it will be rejected (see *ADAREC* in this manual for more information).

TNAA

```
TNAA = number
```

This parameter sets the non-activity time limit (in seconds) for access-only users who have not explicitly specified a TNAA value in the OP command (see *Command Reference, OP command*).

Note that the figure you specify for this parameter is only approximate. In any particular instance, the actual amount of time can vary from this value by up to 10 seconds.

The minimum value is 20, the maximum value is 2592000.

TNAE

TNAE = number

This parameter sets the non-activity time limit (in seconds) for ET logic users who have not explicitly specified a TNAE value in the OP command (see *Command Reference, OP command*).

Note that the figure you specify for this parameter is only approximate. In any particular instance, the actual amount of time can vary from this value by up to 10 seconds.

The minimum value is 20, the maximum value is 2592000.

TNAX

TNAX = number

This parameter sets the non-activity time limit (in seconds) for EXU and EXF users who have not explicitly specified a TNAX value in the OP command (see *Command Reference, OP command*).

Note that the figure you specify for this parameter is only approximate. In any particular instance, the actual amount of time can vary from this value by up to 10 seconds.

The minimum value is 20, the maximum value is 2592000.

TT

TT = number

This parameter sets the transaction time limit for ET logic users who have not explicitly specified a TT value in the OP command (see *Command Reference, OP command*).

If the specified value is less than the corresponding high-water value, a warning is issued.

Note that the figure you specify for this parameter is only approximate. In any particular instance, the actual amount of time can vary from this value by up to 10 seconds.

The minimum value is 20, the maximum value is 2592000.

USER_ID

USER_ID = string

This function influences the output of the DISPLAY parameter options CQ, HQ, ICQ, UQ, UQ_FILES, UQ_TIME_LIMITS, UQ_FULL. Only entries in the specified user ID will be displayed.

WRITE_LIMIT

```
WRITE_LIMIT = [number]
```

This parameter specifies the percentage of modified blocks permitted in the buffer pool before an implicit buffer flush is taken.

Note that "WRITE_LIMIT=" (keeping the equals sign but omitting the number) is equivalent to "WRITE_LIMIT=0".

The minimum value is 0 and the maximum value is 70; 0 means that Adabas will dynamically choose an appropriate value.

XA_RESPONSE_CHECK

```
XA_RESPONSE_CHECK = (keyword [,keyword] ,... )
```

This function enables the DBA to gather information if one of a list of XA response codes occurs (not on OpenVMS). The information written may be used to analyze possible problems in the database's operation. If a response check for an XA response code is enabled, the database section file is written to disk if this response code occurs.

Depending on the setting of the RESPONSE_ABORT option, the nucleus either aborts or continues operation:

- if the RESPONSE_ABORT option is set, the database section file (Adabas.xxx.hh:mm:ss) is written to the database's default directory;
- if the NORESPONSE_ABORT option is set (default setting), the nucleus continues running and the database section file (Adabas.xxx.XAyyyy.hh:mm:ss) is written to disk (refer to the ADAOPR FILE parameter for further information).

By default, no response is trapped and the nucleus continues operation.

Refer to the RESPONSE_ABORT option for further information.

To disable response trapping, use "XA_RESPONSE_CHECK =" without arguments.

The following keywords are supported:

```

XA_RBROLLBACK
XA_RBCOMMFAIL
XA_RBDEADLOCK
XA_RBINTEGRITY
XA_RBOTHER
XA_RBPROTO
XA_RBTIMEOUT
XA_RBTRANSIENT
XA_NOMIGRATE

```

XA_HEURHAZ
XA_HEURCOM
XA_HEURRB
XA_HEURMIX
XA_RETRY
XAER_ASYNC
XAER_RMERR
XAER_NOTA
XAER_INVALID
XAER_PROTO
XAER_RMFAIL
XAER_DUPID
XAER_OUTSIDE
XA_RBROLLBACK

For more information, see *Administration, XA Support*.

20

ADAORD (Reorder Database Or Files, Export/Import Files)

▪ Functional Overview	286
▪ Procedure Flow	287
▪ Checkpoints	289
▪ Control Parameters	289
▪ Restart Considerations	298
▪ Examples	298

This chapter describes the utility "ADAORD".

Functional Overview

The reorder utility ADAORD provides functions to reorganize a whole database (REORDER) and to migrate files between databases (EXPORT/IMPORT).

Depending on the function selected, ADAORD produces or requires a sequential file (ORDEXP).

The main reasons for running ADAORD are:

- To change the layout of a complete database. This includes increasing or decreasing the maximum number of files permitted;
- To change the space allocation or placement of a file, to reduce the number of logical extents assigned to its index, Address Converter or Data Storage and to change or re-establish the padding factors;
- To create one or more test files that all contain the same data. This procedure requires a file to be exported and then imported using a different file number;
- To archive and subsequently reestablish a file, independent of its original placement and the database device types used.

When exporting files from a database, the Adabas nucleus is not required. If a system file is processed, the nucleus must be inactive. For detailed information, please refer to the table of nucleus requirements.

When importing files into a database, the Adabas nucleus is not required to be active. The nucleus may be either started or shut down during this procedure.

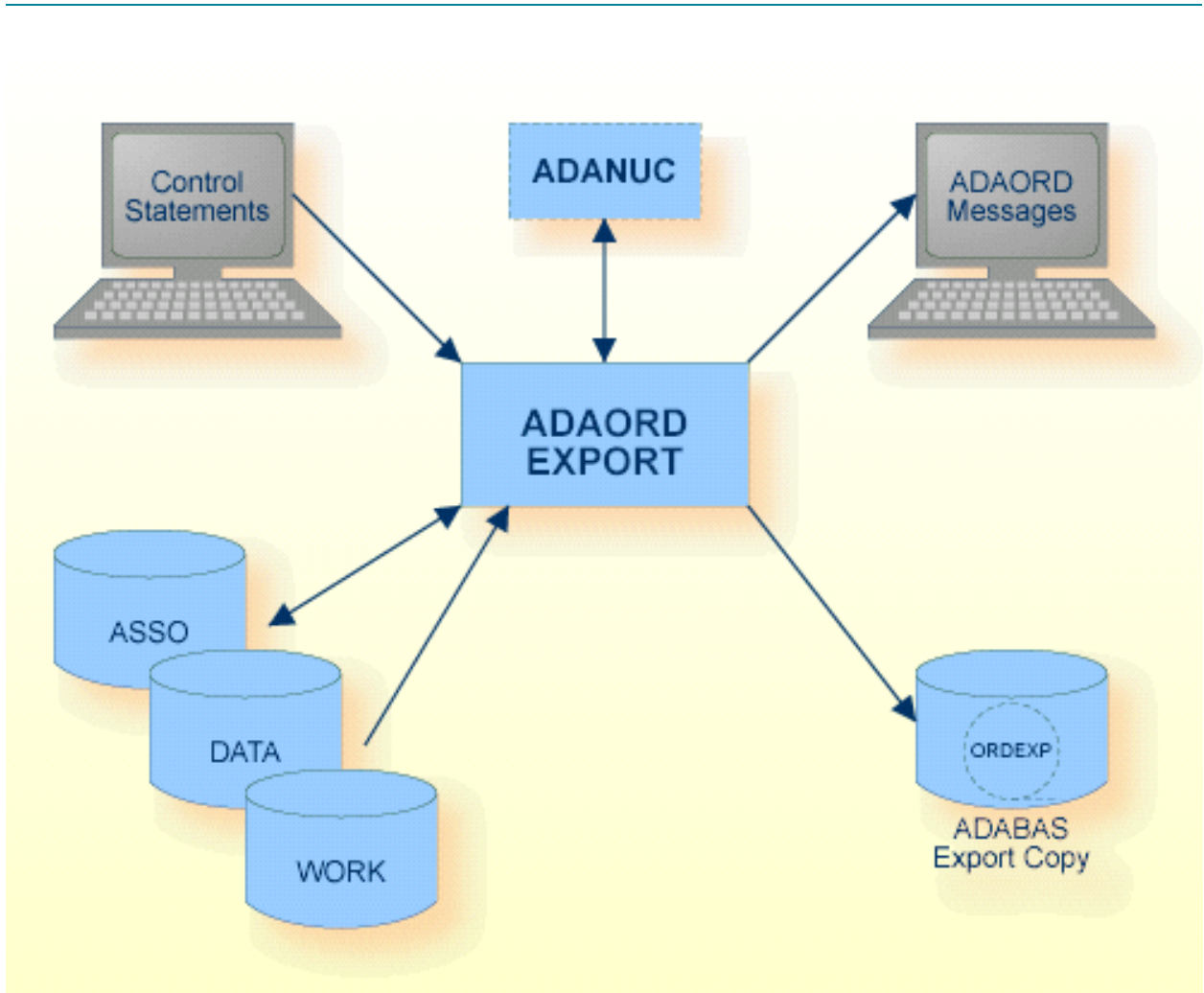
When reordering the database, the nucleus must be inactive.



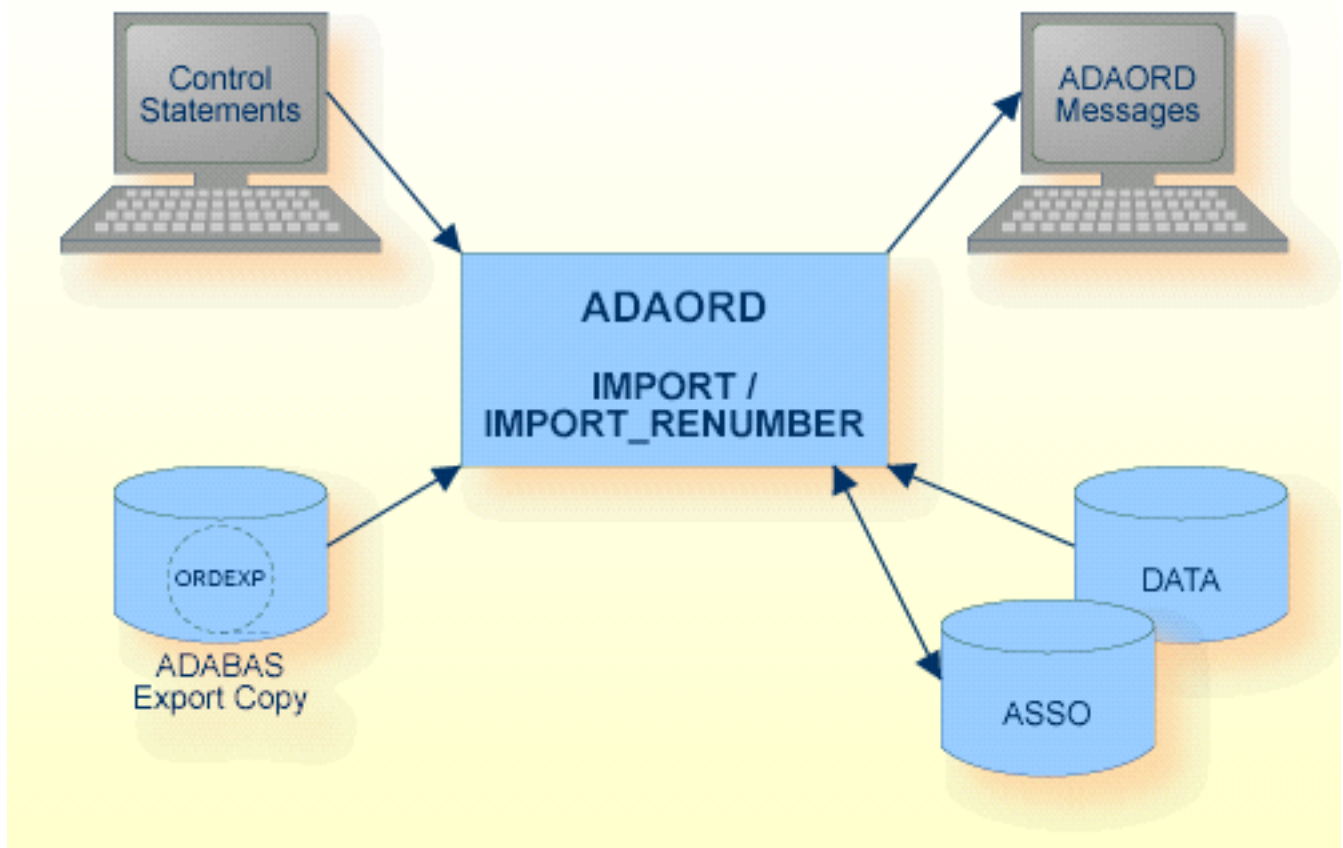
Note: The IMPORT and IMPORT_RENUMBER functions can process export files created with earlier Adabas versions, but not export files created with later Adabas versions.

This utility is a single-function utility.

Procedure Flow



The sequential file ORDEXP can have multiple extents, but only if you are using raw devices. For detailed information about sequential files with multiple extents, see *Administration, Using Utilities*.



The sequential file ORDEXP can have multiple extents, but only if you are using raw devices. For detailed information about sequential files with multiple extents, see *Administration, Using Utilities*.

Data Set	Environment Variable/ Logical Name	Storage Medium	Additional Information
Associator	ASSOx	Disk	
Data storage	DATAx	Disk	
Export copy	ORDEXP	Disk, Tape (* see note)	Export (out), Reorder (in/out), other functions (in)
Control statements	stdin/ SYS\$INPUT		Utilities Manual
ADAORD messages	stdout/ SYS\$OUTPUT		Messages and Codes
Work storage	WORK1	Disk	



Note: (*) A named pipe cannot be used for this sequential file (see *Administration, Using Utilities* for details).

Checkpoints

The following table shows the nucleus requirements for each function and the checkpoints written:

Function	Nucleus must be active	Nucleus must NOT be active	Nucleus is NOT required	Checkpoint written
CONTENTS			X	-
EXPORT		X(* see note)	X	SYNX
IMPORT		X(* see note)	X	SYNP
IMPORT_ RENUMBER		X(* see note)	X	SYNP
REORDER		X	X	SYNP



Note: (*) When processing an Adabas system file

In the case of the EXPORT function, ADAORD writes a single checkpoint and removes the UCB entry when all of the specified files have been exported and the sequential output file (ORDEXP) has been closed.

In the case of the IMPORT function, ADAORD writes a checkpoint and informs the nucleus that the file has been loaded every time a file is successfully imported.

The UCB entry is removed when all of the specified files have been imported. When the utility is executed offline, writing multiple checkpoints increases the probability of a checkpoint block (CPB) overflow. The checkpoint file should, therefore, always be present to allow the Adabas nucleus to be started in order to empty the CPB.

In the case of the REORDER function, ADAORD writes a single checkpoint and removes the UCB entry when the function terminates.

Control Parameters

The following control parameters are available:

CONTENTS

DBID = number

```

EXPORT = (number[-number][,number[-number]]...)
          [,FDT]
          [,SORTSEQ = ({descriptor_name|ISN|PHYSICAL},...)]

FILES = (number[[-number], number[-number]] ...)

IMPORT = (number[-number][,number[-number]]...)
          [,ACRABN = number]
          [,ASSOPFAC = number]
          [,DATAPFAC = number]
          [,DSRABN = number] [,DSSIZE = number[B|M] ]
          [,LOBACRABN = number]
          [,LOBDSRABN = number]
          [,LOBNIRABN = number]
          [,LOBSIZE = numberM]
          [,LOBUIRABN = number]
          [,MAXISN = number]
          [,NIRABN = number|(number,number)]
          [,NISIZE = number[B|M]|(number[B|M],number[B|M])]
          [,UIRABN = number|(number,number)]
          [,UISIZE = number[B|M]|(number[B|M],number[B|M])]

IMPORT_RENUMBER = (number, number[,number])
                  [,ACRABN = number]
                  [,ASSOPFAC = number]
                  [,DATAPFAC = number]
                  [,DSRABN = number] [,DSSIZE = number[B|M] ]
                  [,LOBACRABN = number]
                  [,LOBDSRABN = number]
                  [,LOBNIRABN = number]
                  [,LOBSIZE = numberM]
                  [,LOBUIRABN = number]
                  [,MAXISN = number]
                  [,NIRABN = number|(number,number)]
                  [,NISIZE = number[B|M]|(number[B|M],number[B|M])]
                  [,UIRABN = number|(number,number)]
                  [,UISIZE = number[B|M]|(number[B|M],number[B|M])]

REORDER = *

```

CONTENTS

CONTENTS

This function displays the list of files contained in the sequential output file (ORDEXP) created by a previous run of the EXPORT function.

DBID

DBID = number

This parameter selects the database to be used.

EXPORT

```
EXPORT = (number[-number][,number[-number]]...)
          [,FDT]
          [,SORTSEQ = ({descriptor_name|ISN|PHYSICAL},...)]
```

This function exports (copies) one or more files from the database to a sequential output file (ORDEXP). In order to maintain referential integrity in the export copy, all files that are connected via referential constraints to a specified file are also exported. The file numbers specified are only taken into consideration if they are the file numbers of base files; the corresponding LOB files for the selected files are exported automatically with the base files without having to be specified. An EXPORT consists of copying each file's Data Storage, together with the information that is required to reestablish its index. All of the files to be processed are written to ORDEXP in the sequence in which they are specified. Overlapping ranges and numbers are removed.



Note: If the checkpoint file is included in the file list, it will be processed last.

FDT

This parameter displays the FDT of the file to be processed.

SORTSEQ = ({descriptor_name|ISN|PHYSICAL} ,...)

This parameter controls the sequence in which the Data Storage is processed. It specifies either the field name of a descriptor, subdescriptor or superdescriptor, or the keyword 'ISN' or 'PHYSICAL'.

The default is physical sequence.

The following values can be specified:

Value	Sequence
descriptor_name	<p>If the name of a descriptor, sub- or superdescriptor is specified, the data records are processed in ascending logical sequence of the descriptor values to which the field name refers.</p> <p>A field with the MU, MC or NU option or one that is contained in a periodic group or a sub- or superdescriptor derived from such a field must not be specified.</p> <p>Logical sequence can be used only if a single file has been selected.</p>

Value	Sequence
ISN	If ISN is specified, the data records are processed in ascending ISN sequence.
PHYSICAL	If PHYSICAL is specified or if the SORTSEQ parameter is omitted, the data records are processed in the physical sequence in which they are stored in the Data Storage.

The performance when processing in logical sequence and ISN sequence is better if the database is online (provided that the buffer pool is large enough).

If one value is specified for SORTSEQ, that value is valid for all files. If more than one value is specified, the number of values must be the same as the number of file ranges specified for the EXPORT parameter. In this case, the first file range is exported in the first specified sort sequence, the second file range is exported in the second specified sort sequence, and so on.

Example

```
EXPORT = (1, 20-30, 40)
SORTSEQ = (AA, PHYSICAL, ISN)
```

File 1 is exported in the sequence of descriptor AA, files 20-30 are exported in physical sequence and file 40 is exported in ISN sequence.

FILES

```
FILES = (number[[-number], number[-number]] ...)
```

This parameter is used to display information concerning the status of the specified files contained on the sequential input file (ORDEXP).

IMPORT

```
IMPORT = (number[-number][,number[-number]]...)
    [,ACRABN = number]
    [,ASSOPFAC = number]
    [,DATAPFAC = number]
    [,DSRABN = number] [,DSSIZE = number[B|M] ]
    [,LOBACRABN = number]
    [,LOBDSRABN = number]
    [,LOBNIRABN = number]
    [,LOBSIZE = numberM]
    [,LOBUIRABN = number]
    [,MAXISN = number]
    [,NIRABN = number|(number,number)]
    [,NISIZE = number[B|M]|(number[B|M],number[B|M])]
    [,UIRABN = number|(number,number)]
    [,UISIZE = number[B|M]|(number[B|M],number[B|M])]
```

This function imports one or more files into a database, using the data on the sequential file (ORDEXP) produced by a previous run of ADAORD. In order to maintain referential integrity, all

files connected via referential constraints to a specified file are also imported. The file numbers specified are only taken into consideration if they are the file numbers of base files; the corresponding LOB files for the selected files are imported automatically with the base files without having to be specified. The file numbers specified are sorted into ascending sequence. Overlapping ranges and numbers are removed.

The file numbers specified must not be loaded in the database.

By default, ADAORD controls the file placement and the allocation quantities. The parameters that can be used to overwrite these defaults may be used only if a single file has been selected.

Please refer to the [IMPORT_RENUMBER](#) function for the description of the parameters.

IMPORT_RENUMBER

```
IMPORT_RENUMBER = (number, number[,number])
                  [,ACRABN = number]
                  [,ASSOPFAC = number]
                  [,DATAPFAC = number]
                  [,DSRABN = number] [,DSSIZE = number[B|M] ]
                  [,LOBACRABN = number]
                  [,LOBDSRABN = number]
                  [,LOBNIRABN = number]
                  [,LOBSIZE = numberM]
                  [,LOBUIRABN = number]
                  [,MAXISN = number]
                  [,NIRABN = number|(number,number)]
                  [,NISIZE = number[B|M]|(number[B|M],number[B|M])]
```

This function imports a file into a database, using the data on the sequential file (ORDEXP) produced by a previous run of ADAORD. It is not possible to import and renumber a file that is connected to another file via referential integrity. Constraints must either be dropped before exporting the files, or the files must be imported without renumbering and be renumbered later (ADADBM RENUMBER). The first number given defines the base file to be imported, and the second number is the new file number to be assigned to the file. The third, optional number is the new file number for the LOB file. If the third number is not specified, the LOB file number (if it exists) remains unchanged.

The new file number must not be loaded in the database.

Unless otherwise specified, ADAORD controls the file placement and the allocation quantities.

ACRABN = number

This parameter specifies the RABN at which the space allocation for the Address Converter (AC) is to start.

If this parameter is omitted, ADAORD assigns the starting RABN.

ASSOPFAC = number

This parameter specifies the new padding factor to be used for the file's index. The number specified is the percentage of each index block which is not to be used by ADAORD or a subsequent run of the mass update utility ADAMUP. This padding area is reserved for future use if additional entries have to be added to the block by the Adabas nucleus. This avoids the necessity of having to relocate overflow entries to another block.

A value may be specified in the range of 0 to 95.

A small padding factor (0 to 10) should be specified if little or no descriptor updating is expected. A larger padding factor (10 to 50) should be specified if a large amount of descriptor updating is expected in which new descriptor values are created.

If this parameter is omitted, the current padding factor in effect for the file's index is used.

DATAPFAC = number

This parameter specifies the new padding factor to be used for the file's Data Storage. The number specified is the percentage of each data block which is not to be used by ADAORD. This padding area is reserved for future use if any record in a block requires additional space as result of record updating by the Adabas nucleus. This avoids the necessity of having to relocate overflow entries to another block.

A value may be specified in the range of 0 to 95.

A small padding factor (0 to 10) should be specified if there is little or no record expansion. A larger padding factor (10 to 50) should be specified if there is a large amount of record updating which will cause expansion.

If this parameter is omitted, the current padding factor in effect for the file's Data Storage is used.

DSRABN = number

This parameter specifies the RABN at which the space allocation for the file's Data Storage (DS) is to start.

If this parameter is omitted, ADAORD assigns the start RABN.

DSSIZE = number[B|M]

This parameter specifies the number of blocks (B) or megabytes (M) to be initially assigned to the file's Data Storage (DS). By default, the size is given in megabytes.

If this parameter is omitted, ADAORD calculates the size based on the old number of blocks allocated and the difference between the old and new padding factor.

LOBACRABN=number

This parameter specifies the RABN at which the space allocation for the LOB file's Address Converter (AC) is to start.

If this parameter is omitted, ADAORD assigns the start RABN.

LOBDSRABN=number

This parameter specifies the RABN at which the space allocation for the LOB file's Data Storage (DS) is to start.

If this parameter is omitted, ADAORD assigns the start RABN.

LOBNIRABN=number

This parameter specifies the RABN at which the space allocation for the LOB file's Normal Index (NI) is to start.

If this parameter is omitted, ADAORD assigns the start RABN.

LOBSIZE=numberM

This parameter specifies the number of megabytes to be initially assigned to the LOB file's Data Storage (DS). The AC size, NI size and UI size for the LOB file are derived from this size.

If this parameter is omitted, ADAORD calculates the size based on the old number of blocks allocated and the difference between the old and new padding factor.

LOBUIRABN=number

This parameter specifies the RABN at which the space allocation for the LOB file's Upper Index (UI) is to start.

If this parameter is omitted, ADAORD assigns the start RABN.

MAXISN = number

This parameter specifies the highest permissible ISN for the file. ADAORD uses this parameter to determine the amount of space to be allocated for the file's Address Converter (AC).

Because there is no automatic extension of the initial allocation, a value that is smaller than the file's current first free ISN will cause ADAORD to terminate execution and return an error status if there are ISNs outside the Address Converter.

If this parameter is omitted, the value of MAXISN currently in effect for the file's Address Converter is used.

A contiguous-best-try allocation is used.

NIRABN = number|(number,number)

This parameter specifies the RABN(s) at which the space allocation for the file's Normal Index (NI) is to start. Adabas usually stores small descriptor values (≤ 253 bytes) in small index blocks (block size < 16 KB) and large descriptor values in large index blocks (block size ≥ 16 KB). For this reason, it is possible to specify 2 RABNs - if you specify 2 RABNs, one must have a block size < 16 KB, and the other must have a block size ≥ 16 KB.

If this parameter is omitted, ADAORD assigns the start RABN.

NISIZE = number[B|M]|(number[B|M],number[B|M])

This parameter specifies the number of blocks (B) or megabytes (M) to be initially assigned to the file's Normal Index (NI). By default, the size is given in megabytes. If two values are specified and the NIRABN parameter is also specified, the first value corresponds to the first value of the NIRABN parameter, and the second value corresponds to the second value of the NIRABN parameter. If two values are specified and the NIRABN parameter is not specified, the first value specifies the size of small normal index blocks (< 16 KB), and the second value specifies the size of large NI blocks (≥ 16 KB).

If this parameter is omitted, ADAORD calculates the size based on the old number of blocks allocated and the difference between the old and new padding factor.

UIRABN = number|(number,number)

This parameter specifies the RABN(s) at which the space allocation for the file's Upper Index (UI) is to start. Adabas usually stores small descriptor values (≤ 253 bytes) in small index blocks (block size < 16 KB) and large descriptor values in large index blocks (block size ≥ 16 KB). For this reason, it is possible to specify 2 RABNs - if you specify 2 RABNs, one must have a block size < 16 KB, and the other must have a block size ≥ 16 KB.

If this parameter is omitted, ADAORD assigns the start RABN.

UIsize = number[B|M]|(number[B|M],number[B|M])

This parameter specifies the number of blocks (B) or megabytes (M) to be initially assigned to the file's Upper Index (UI). By default, the size is given in megabytes. If two values are specified and the UIRABN parameter is also specified, the first value corresponds to the first value of the UIRABN parameter, and the second value corresponds to the second value of the UIRABN parameter. If two values are specified and the UIRABN parameter is not specified, the first value specifies the size of small upper index blocks (< 16 KB), and the second value specifies the size of large UI blocks (≥ 16 KB).

If this parameter is omitted, ADAORD calculates the size based on the old number of blocks allocated and the difference between the old and new padding factor.

REORDER

REORDER = *

This function is used to change the layout of a whole database. It rearranges the database's global areas, eliminates fragmentation in the DSST and the files' Address Converter, Data Storage, Normal Index and Upper Index extents by physically changing their placement. It also re-establishes the files' padding factors. Exclusive control of the database container files is required.

A REORDER database implicitly exports the files, deletes them from the database and then re-imports them. The sequential file (ORDEXP) that is created during the REORDER is kept.



Note: ADAORD uses a best-fit algorithm for the allocation of the disk space for the files. Therefore, it may occur that the first container of a given type remains empty if it is followed by another container with adequate block size which is smaller than the first one.

Restart Considerations

ADAORD has no restart capability.

An abnormally terminated EXPORT must be rerun from the beginning.

An abnormally terminated IMPORT of one or more files will result in lost RABNs for the last file being imported. These RABNs can be recovered by executing ADADBm's RECOVER function. The files preceding the one being processed when the interrupt occurred will be available in the database. Therefore, the IMPORT function should be rerun starting with the file number at which the interrupt occurred.

An abnormally terminated IMPORT_RENUMBER will result in lost RABNs for the file being imported. These RABNs can be recovered by executing ADADBm's RECOVER function. The IMPORT_RENUMBER function has to be rerun from the beginning.

An abnormally terminated REORDER at the database level may result in a database that cannot be accessed if the interrupt occurred while reordering the database's global areas (GCB, FST, DSST, etc.). In this case, either a new empty database has to be created using ADAFRM or the old database has to be reestablished from an Adabas backup copy, using ADABCK's RESTORE database function. If the interrupt occurred during the re-import phase, it will result in lost RABNs for the last file being imported. These RABNs can be recovered by executing ADADBm's RECOVER function. The files preceding the one being processed when the interrupt occurred will be available in the database. The remaining files can be obtained from the sequential work file (ORDEXP) by using ADAORD's IMPORT function.

Examples

In the examples below, the files 1, 2, 4, 6, 7, 8, 10, 11, 12 and 25 are loaded in database 1. Database 2 contains files 3, 6 and 11.

Example 1

```
adaord: dbid = 1
adaord: export = (1-4,7,10-25)
```

Files 1, 2, 4, 7, 10, 11, 12 and 25 are exported from database 1.

Example 2

```
adaord: dbid = 2
adaord: import = (1-10,12)
```

Files 1, 2, 4, 7, 10 and 12 are imported into database 2. It is not possible to specify "import=(1-12)" because ADAORD first checks to see if one of the files to be imported is already loaded, and if it is, then the whole import is rejected - in this case file 11 is already loaded.

Example 3

```
adaord: dbid = 2
adaord: import_renumber = (11,19), acrabn = 131, datapfac = 20
```

File 11 is imported into database 2 using a new file number of 19 (because 11 is already in use). The file's Address Converter (AC) is to be allocated at ASSO RABN 131. The new padding factor for the Data Storage (DS) is 20 percent.

Example 4

```
adaord: dbid = 1
adaord: reorder = *
```

The whole database is reordered.

21 ADAPLP (Protection Log Printout)

▪ Functional Overview	302
▪ Procedure Flow	303
▪ Checkpoints	304
▪ Control Parameters	304
▪ ADAPLP Output	313

This chapter describes the utility "ADAPLP".

Functional Overview

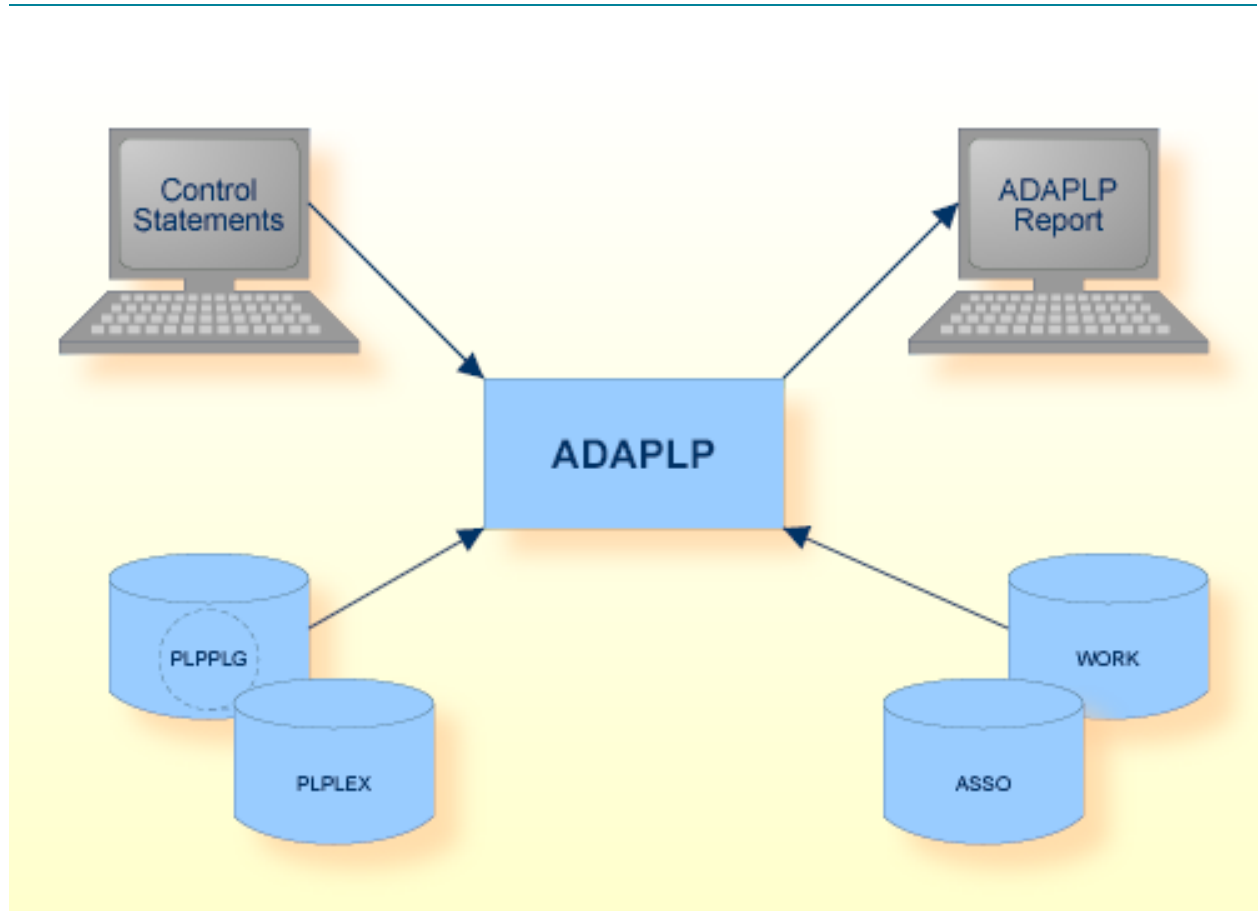
The ADAPLP utility prints the Protection Log or WORK.

This utility is a multi-function utility.



Note: LOB values are split into several records in a LOB file; when LOB values are stored in the database, the Protection Log contains the log records for the modifications of the LOB file. This means that ADAPLP does not display the LOB values as one value, but rather it displays the modifications of the corresponding records in the LOB file instead. It is not possible to decompress the LOB file records because the LOB records are too large to fit into one block, and the continued Protection Log records cannot be decompressed.

Procedure Flow



Data Set	Environment Variable/ Logical Name	Storage Medium	Additional Information
Associator	ASSOx	Disk	
Protection log	PLPPLG	Disk, Tape	Utilities Manual
Protection log (last extent)	PLPLEX	Disk	Only required if you process a PLOG with an extension count > 1 and if you use the DECOMPRESS or DELTA option: you must provide the last PLOG extent before the PLOG extent to be processed.
Control statements	stdin/ SYS\$INPUT		Utilities Manual
ADAPLP report	stdout/ SYS\$OUTPUT		Messages and Codes
Work storage	WORK1	Disk	

The sequential file PLPPLG can have multiple extents. For information about sequential files with multiple extents, see *Administration, Using Utilities*.

Checkpoints

The utility writes no checkpoints.

Control Parameters

The following control parameters are available:

```
M   DATASET = keyword
    DBID = number
D   [NO]DECOMPRESSED
    DELTA
D   [NO]DUMP
    FILES = (number [-number][,number [-number]]...)
D   [NO]HEADER
    INTERNAL_ID = number
    ISN = (number [,number] ... )
    MODIFIED_RABN = number
    NOFILETYPE
    NONULL
    PLOG = (number [,number])
M   RABN = {*|number[-number]}
    RECORD ={*| (number [- number] [, number [- number]]...) }
    SEQ = number
D   [NO]SHORT
```



```

    THREAD = number

    TSN = number
D   TYPE = (keyword [,keyword]...)

    USER_ID = string
D   [NO]WXA

```

DATASET

```
DATASET = keyword
```

This parameter selects the file containing the Protection Log information to be processed. The keyword can take the values PLOG or WORK. The parameter PLOG must be specified first if DATASET=PLOG is to be specified and the protection log is located on a raw device.

DBID

```
DBID = number
```

This parameter selects the database to be used.

This parameter must be used when DATASET=WORK is requested, or if DATASET=PLOG and the protection log is within a raw section. This parameter must be the first parameter to be specified. Otherwise, if this parameter is not specified, the DBID stored in the PLOG is used.

[NO]DECOMPRESSED

```
[NO]DECOMPRESSED
```

This option indicates whether for each selected DATA record from a protection log, one line per field is printed with the field name and its decompressed value in hex (DECOMPRESSED) or not (NODECOMPRESSED).

For an inserted record, an after image containing the field values of the record after the insert is displayed.

For an updated record, a before image containing the field values before the update, and an after image containing the field values of the record after the update are displayed.

For a deleted record, a before image containing the field values before the delete operation is displayed.

If you specify DECOMPRESSED and NONULL, no output is produced for the following:

- Fields with NU or NC option with null-value;


```
Field   : AX: ^3030303030303030
Field   : AY: ^3030303030303030
End of PE-group : AW
MU-field : AZ, count = ^01
        AZ( 1): ^202020
```

Example output for DECOMPRESSED (with NONULL option)

```
>>> After Image <<<
```

```
Length = 15, ISN = 2
```

```
Field   : AA: ^30372E31322E3034
Field   : AF: ^20
Field   : AG: ^20
```

DELTA

```
DELTA
```

This parameter indicates that only changed fields after an update are displayed.

For an inserted record, the same output is produced as with the options DECOMPRESSED, NONULL.

For an updated record, a Delta containing the modified field values is displayed. Note that for MU/PE fields, the value count displayed can be smaller than the displayed MU/PE indices if the MU/PE count has been decreased - this is because all field values have been set to the null value.

If a record is deleted, no output is produced, however, you can see the deletion if you display protection log entries of the type CE.



Note: Decompression (DELTA output) is not possible for CONTINUED records. CONTINUED records are created if a PLOG record plus the block header is larger than 32 KB in the PLOG or larger than the block size used for WORK.

[NO]DUMP

```
[NO]DUMP
```

This option indicates whether the variable part of a Protection Log record is included in the printout (DUMP) or not (NODUMP).

If DUMP is specified, the variable part of each Protection Log record is displayed in both hexadecimal and uninterpreted ASCII format.

DUMP implicitly resets SHORT.

The default is NODUMP.

FILES

```
FILES = (number [-number][,number [-number]]...)
```

The Protection Log records are only displayed if they belong to the file(s) specified by this parameter.

Only records of the types DA, DV, EXT, INDEX and FCB are displayed.

Please refer to the tables at the end of this section for a description of the various types of Protection Log records.

[NO]HEADER

```
[NO]HEADER
```

This option indicates whether for each block of the Protection Log a header is displayed (HEADER) or not (NOHEADER).

The default is HEADER.

INTERNAL_ID

```
INTERNAL_ID = number
```

This option displays only the records with the specified internal ID.

ISN

```
ISN = (number [,number] ... )
```

The Protection Log records are only displayed if they belong to the ISNs specified by this parameter. Only records of the types DA and DV are displayed. Please refer to the tables at the end of this section for a description of the various types of Protection Log records. This parameter can only be used in conjunction with the FILE parameter.

MODIFIED_RABN

```
MODIFIED_RABN = number
```

This option displays only the records in which modifications for the specified RABN are logged.

Please refer to the tables at the end of this section for a description of the various types of Protection Log records.

NOFILETYPE

```
NOFILETYPE
```

This keyword specifies that record types that are independent of file numbers (for example ET and BT records) will be displayed in addition to the record types that are bound to file numbers.

Example

```
adaplp dbid=6 file=25 type=(da,dv,et,bt) nofiletype
```

The DA and DVT records of file 25 together with all ET and BT records will be displayed.

NONULL

```
NONULL
```

The NONULL parameter is only relevant if the DECOMPRESSED parameter is also specified. Please refer to the [DECOMPRESSED](#) parameter for further information.

PLOG

```
PLOG = (number[,number])
```

This parameter is required if DATASET=PLOG is specified and the protection log is within a raw section. It is optional if the protection log is within a file system. The PLOG number and the extension count can be specified. If an extension count is specified, then only the specified extent will be processed. If no extension count is specified, Adabas will open subsequent extents when necessary. The parameter PLOG must be specified before DATASET=PLOG is specified.

Example:

```
Section layout
      .
      .
      .
250000 260000  10001  30  PLG.36 created
377000 378000   1001  30  PLG.36(3) created

adaplp: plog=36
adaplp: dataset=plog
```

PLG.36 will be opened

```
adaplp: plog=(36,3)
adaplp: dataset=plog
```

PLG.36(3) will be opened.

RABN

```
RABN ={*| number [- number] }
```

This parameter selects one block or a range of consecutive blocks on the WORK or Protection Log file. The information contained in the specified blocks is displayed.

If you specify "*", all blocks are displayed.



Note: If you start ADAPLP without specifying RABN, the utility will run, but will not produce any output.

Example

```
adaplp: rabn = 123
adaplp: rabn = 123 - 1246
```

RECORD

```
RECORD ={* | (number [- number] [, number [- number]]...)
```

This parameter selects the records or ranges of records to be printed. All of the records are printed if '*' or nothing is specified.

Example:

```
adaplp: record = (2-5,9,11)
```

The records 2, 3, 4, 5, 9 and 11 are written while printing one or more PLOG blocks.

SEQUENCE

```
SEQUENCE = number
```

This option displays only the records written by the specified sequence number.

[NO]SHORT

```
[NO]SHORT
```

This option indicates whether only Protection Log block headers are printed out (SHORT), or whether all the records in each block are included in the display (NOSHORT).

SHORT implicitly resets DUMP.

By default, the Protection Log block header is displayed followed by all of the records contained in the block.

The default is NOSHORT.

THREAD

```
THREAD = number
```

This option displays only the records with the specified thread.

TSN

TSN = number

This option displays only the records with the specified transaction sequence number (TSN).

Only records of the type BT, C5, CL, DA, DV, ET and XA (not on OpenVMS) are displayed.

Please refer to the tables at the end of this section for a description of the various types of Protection Log records.

TYPE

TYPE = (keyword [,keyword]...)

This option displays only the protection log records specified by the given keyword(s). Each keyword corresponds to one or more protection log record types, as shown in the following table.

Keyword	Protection Log record type
AB	AB
ASSO	the record type AC and all record types that are selected by the keywords EXT, FCB and INDEX
AT	AT
BF	BS, BE, BF
BT	BT
C1	C1
C5	C5
CE	CE
CF	CF
CT	CT
DA	DA
DATA	all record types that are selected by the keywords BT, CE, DA, DV, ET and OP.
DC	DC
DT	DT
ET	ET, CL
EXT	ACEXT, UIEXT, NIEXT, DSEXT
FCB	FCBDS, FCBIX, SPISN
INDEX	FE, INDEX, IB, INSRU, REMRU
OP	OP
XA	all record types that are selected by the keywords YB, YD, YF and YP
YB	YB (not on OpenVMS)
YD	YD (not on OpenVMS)

Keyword	Protection Log record type
YF	YF (not on OpenVMS)
YP	YP (not on OpenVMS)

Please refer to the tables at the end of this section for a description of the various types of Protection Log records.

The default is to display all protection log record types.

USER_ID

```
USER_ID = string
```

This option displays only the records which start with the specified user ID.

Only records of the type BT, C1, C5, CL, DA, DV, ET, FCBDS, FCBIX, INDEX and XA are displayed.

Please refer to the tables at the end of this section for a description of the various types of Protection Log records.

[NO]WXA

```
[NO]WXA
```

This option alternates between the WORK part 1 ring buffer (NOWXA) and WORK part 1 XA area (WXA).

The default is NOWXA.

ADAPLP Output

Each block of the Protection Log or WORK is preceded by a header, which consists of the following:

- the block sequence number;
- the size of the block;
- the number of the session that the block belongs to (identical to the PLOG number);
- the time stamp showing when the block was created (internal time stamp for WORK).

The output for a record consists of the following entries:

- a record sequence number (starting at 1 for each block);
- the internal length of the record;
- the command sequence number (uniquely identifies a command);

- the type of PLOG record (see the following table for more information);
- the number of the thread that executed the command.

In addition, most records also have the following entries:

- the internal user identification (in hexadecimal notation) that is uniquely assigned for each command that opens a transaction.

The table below shows the types of PLOG records:

Type	Description
AB	logs WORK wrap around (WORK only).
AC	logs the relocation of a record during backout transaction (WORK only).
ACEXT	logs the extension of the address converter (WORK only).
AT	logs the adding of a field (ADADBM).
BE	logs the end of a buffer flush (WORK only).
BF	logs the start and end of a buffer flush (WORK only).
BS	logs the start of a buffer flush (WORK only).
BT	logs the start of BT processing.
C1	log record from a C1 command. Contains the checkpoint name (PLOG only).
C5	log record from a C5 command (PLOG only).
CE	indicates the last entry of a command (last entry with this sequence number). If the command was a delete operation, the file number and the ISN of the deleted record is displayed. Example >>> DELETE FILE 10 ISN 2 <<<
CF	logs the creation of an FDT (ADAFDU).
CL	logs the CLOSE of a user.
CT	logs the creation of a file (ADAFDU).
DA	logs a data record change. The file, RABN, and ISN of the data record are displayed. The record is either an after image (AI), a before image (BI), or a delta image (DI) and is displayed when DUMP is enabled. `TSN' is an internal transaction sequence number. All entries that originate from one transaction have the same TSN (see also the description of the ET command in the Command Reference Manual). The output of `WB' is only displayed if DATASET=WORK has been specified. It shows the WORK block where the previous PLOG record of the same TSN can be found. A `clu' value that is not zero indicates an exclusive or privileged user.
DC	logs the dropping of a field (ADADBM).
DSEXT	logs the extension of data storage (WORK only).
DT	logs the deletion of a file (ADADBM).

Type	Description
DV	logs a descriptor update (should always be preceded by a DA record). The entries for the file, ISN, TSN, clu, and WB are the same as for the DA record type.
ET	log entry from an ET command. The ET TSN gives the TSN of the last user data written by an ET command.
FCBDS	logs an FCB change for data storage (WORK only).
FCBIX	logs an FCB change for the normal index (WORK only).
FE	logs a change of an index block's first entry (WORK only).
IB	logs an index block that is modified (WORK only).
INDEX	logs an index block that is split (WORK only).
INSRU	logs the insertion of an index block into a reuse chain (WORK only).
NIEXT	logs the extension of the normal index (WORK only).
OP	logs the OPEN of a user.
REMRU	logs the deletion of an index block from a reuse chain (WORK only).
SPISN	logs changes in ISN reuse or space reuse.
UIEXT	logs the extension of the upper index (WORK only).
YB	logs the backout of a transaction within the XA protocol (not on OpenVMS).
YD	logs the discarding of a heuristically terminated transaction within the XA protocol (not on OpenVMS).
YF	logs the final commit of a transaction within the XA protocol (not on OpenVMS).
YP	logs the preliminary commit of a transaction within the XA protocol (not on OpenVMS).

There are also several flags that may be displayed with DA or DV records:

Flag	Description
AI	the data of this PLOG record contain an after image of the data record (record type DA).
BACKOUT	indicates that the record was written during a backout within a single command.
BI	the data of this PLOG record contain a before image of the data record (record type DA).
BT	indicates that the record was written during the backout of a transaction.
DI	the data of this PLOG record contain a delta image of the data record (record type DA).
FDATA	indicates that this is the first DA record of this command.
FIRST_ENTRY	indicates that this is the first record with a given sequence number.
HIMERGE	merge of the highest index level.
HISPLIT	split of the highest index level.
USERD	transaction carries user data.

22 ADAPRI (Print Adabas Blocks)

- Functional Overview 318
- Procedure Flow 319
- Checkpoints 320
- Control Parameters 320

This chapter describes the utility "ADAPRI".

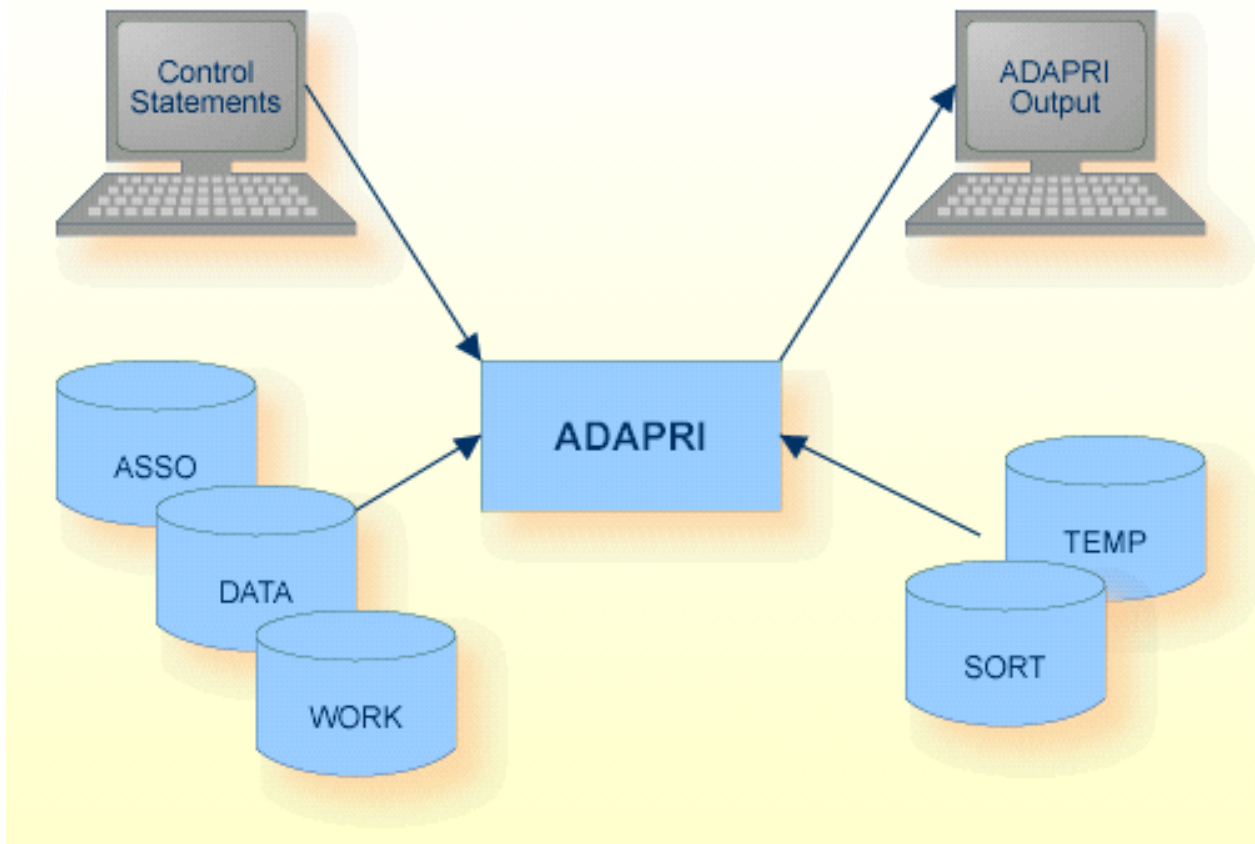
Functional Overview

The ADAPRI utility prints the contents of a block (or range of blocks) in the Associator, Data Storage, WORK, TEMP, or SORT for maintenance or auditing purposes.

The output is in hexadecimal and ASCII format. Subsequent identical lines and blocks are suppressed.

This utility is a multi-function utility.

Procedure Flow



Data Set	Environment Variable/ Logical Name	Storage Medium	Additional Information
Associator	ASSOx	Disk	
Data storage	DATAx	Disk	
Sort storage	SORTx	Disk	
Control statements	stdin/ SYS\$INPUT		Utilities Manual
ADAPRI output	stdout/ SYS\$OUTPUT		
Temporary storage	TEMPx	Disk	
Work storage	WORK1	Disk	

Assignments to the ASSO container files are required in order to be able to process the DATA or WORK container files.

Checkpoints

The utility writes no checkpoints.

Control Parameters

The following control parameters are available:

```
DATASET = keyword
```

```
DBID = number
```

```
RABN = number [- number]
```

DATASET

```
DATASET = keyword
```

This parameter specifies the part of the database to be dumped. Valid keywords are:

Keyword	Meaning
ASSO	Associator
DATA	Data Storage
SORT	Sort Area
TEMP	Temporary Area
WORK	Work Area

Example

```
adapri: dataset = asso, rabn = 123 - 321
```

The Associator is dumped from RABN 123 to RABN 321

DBID

```
DBID = number
```

This parameter selects the database to be used.

This parameter is not required if DATASET = TEMP or SORT.

RABN

```
RABN = number [- number]
```

This parameter specifies one RABN or a range of RABNs to be dumped.

Examples

```
adapri: dbid = 1, dataset = data, rabn = 123
```

DATA RABN 123 of database 1 is to be dumped.

```
adapri: dataset = sort, rabn = 123 - 129
```

The RABNs from 123 to 129 on the data set SORT are to be dumped.

23

ADAREC (Recovery Of Database Or Files)

▪ Functional Overview	324
▪ Procedure Flow	325
▪ Checkpoints	327
▪ ADAREC Input Data	327
▪ Control Parameters	327
▪ Examples	333
▪ ADAREC Restart Considerations	340

This chapter describes the utility "ADAREC".

Functional Overview

The ADAREC utility consists of the following database recovery functions:

- The CLOSE function writes a clean end-of-file to an abnormally-terminated Protection Log file within a disk section (UNIX platforms only).
- The LIST function lists information about a Protection Log.
- The REGENERATE function re-applies all of the updates made between two specified checkpoints. The checkpoints used are normally the result of a checkpoint command (C1) but may also be internal checkpoints taken by OP commands from EXU users or utility actions. If the whole database is to be regenerated, certain files may be excluded by using the EXCLUDE_FILES option. The files specified with this option are not regenerated, and the updates that are excluded are reported.

If REGENERATE terminates at a SYNPN checkpoint, ADAREC "looks ahead" on the current PLOG to find an alternative restart point for the next run of this PLOG. The utility then displays a list of other utility functions that have to be executed before ADAREC can be restarted. If one or more SYNPN checkpoints were found, ADAREC terminates with status 14 (setting the status to 14 was introduced with version 6.1.7.1 - previous Adabas versions terminated with status 0). The calculated restart point can be reset or overridden by entering BLOCK = or CHECKPOINT =. Refer to the database report utility ADAREP in this manual for a description of the possible system checkpoint types.

Normally, REGENERATE completes all fully-logged and confirmed transactions. This function is most frequently used when the database (or one or more files) has been restored to a previous status with the RESTORE function of the ADABCK utility.

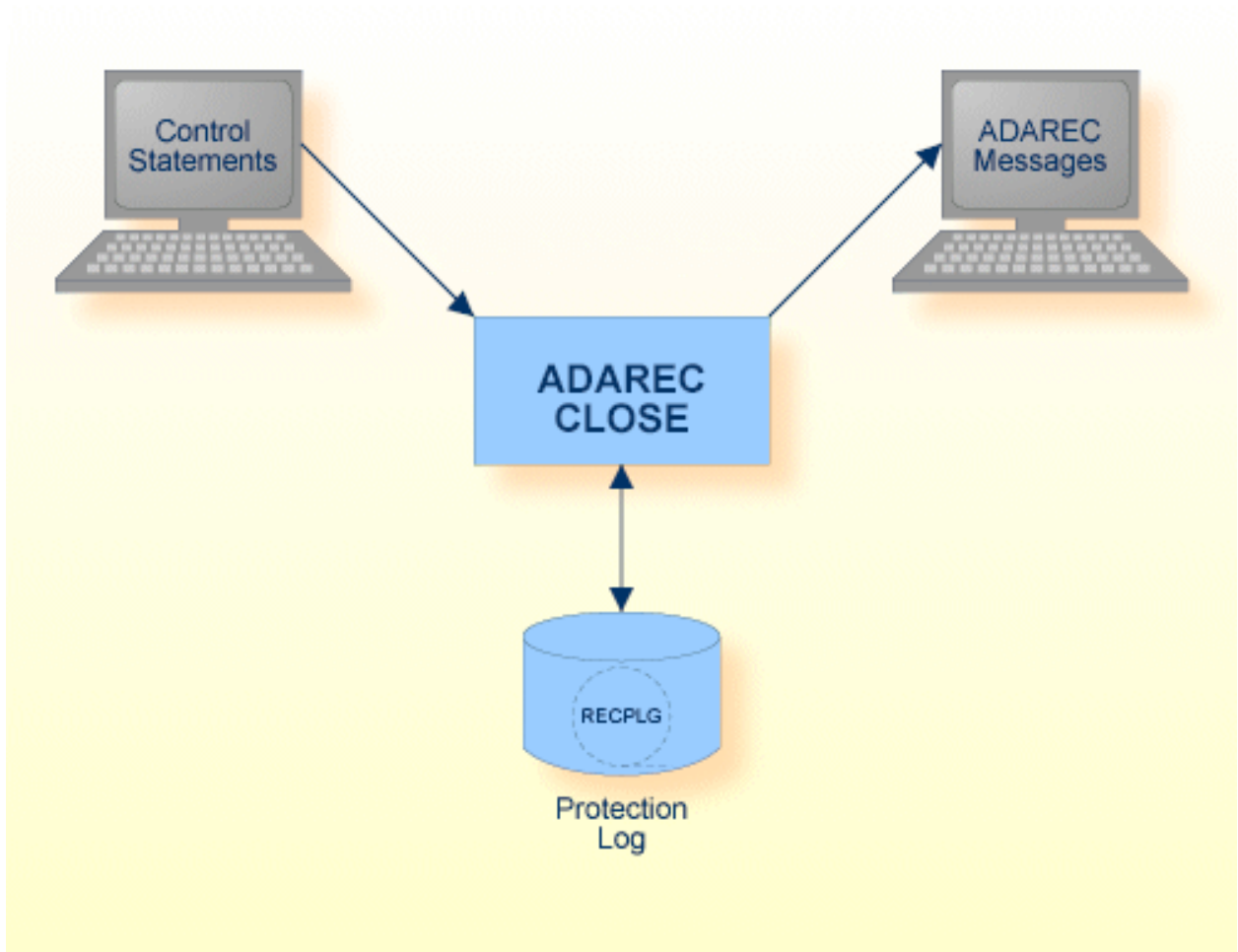
If the utility writes records to the error file, it will exit with a non-zero status.



Note: If ADAREC is used more than once at the same time to regenerate files, you should first increase the value of the nucleus parameter LBP - this is because ADAREC performs a large number of database updates, and failure to provide a large enough value of LBP may lead to an Adabas response code 162 being returned.


This utility is a single-function utility.

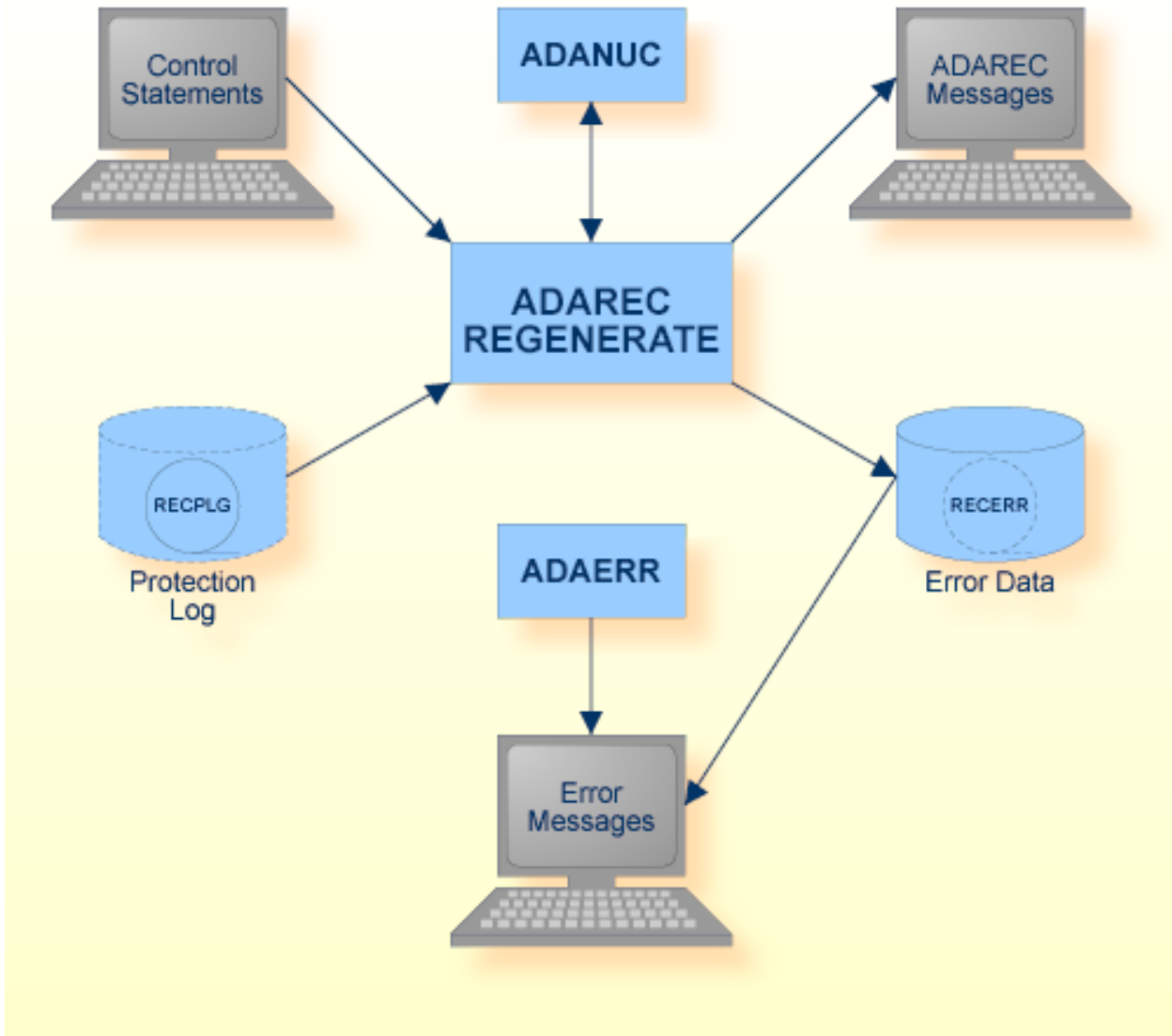
Procedure Flow



(UNIX platforms only)

Data Set	Environment Variable	Storage Medium	Additional Information
Protection log	RECPLG	Disk (* see note)	

 **Note:** (*) The CLOSE function works only on protection log files in raw disk sections.



REGENERATE Function

Data Set	Environment Variable/ Logical Name	Storage Medium	Additional Information
Control statements	stdin/ SYS\$INPUT		Utilities Manual
ADAREC messages	stdout/ SYS\$OUTPUT		Messages and Codes
Rejected data	RECERR	Disk, Tape (* see note)	Output of ADAREC
Protection log	RECPLG	Disk, Tape	



Note: (*) A named pipe can be used for this sequential file (not on OpenVMS, see *Administration, Using Utilities* for details).

The sequential file RECPLG can have multiple extents. For detailed information about sequential files with multiple extents, see *Administration, Using Utilities*.

Checkpoints

The following table shows the nucleus requirements for each function and the checkpoints written:

Function	Nucleus must be active	Nucleus must NOT be active	Nucleus is NOT required	Checkpoint written
CLOSE			X	-
LIST			X	-
REGENERATE	X			SYNX

ADAREC Input Data

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 regenerate the updates.

Control Parameters

The following control parameters are available:

```

CLOSE = PLOG-number[(extent-number)]

M   DBID = number

    LIST = keyword

    REGENERATE = {* [,EXCLUDE_FILES =
                  (number[-number] [,number[-number] ] ... ) } |
                  (number[-number] [,number[-number] ] ... )
    PLOG = number
D   [, [NO]BI_CHECK]
    [,BLOCK = ([number][,number])
      ,CHECKPOINT = ([string][,string])
D   [, [NO]ERROR_LOG]
D   [,ON_ERROR = keyword

```

CLOSE

```
CLOSE = PLOG-number[(extent-number)]
```

The CLOSE function writes a clean end-of-file to an abnormally-terminated Protection Log file within a disk section. This function must be executed before such a Protection Log file can be used as input for the REGENERATE function.

The CLOSE function may be run when an AUTORESTART is pending or after the AUTORESTART has been performed.

This function can still be used even if subsequent Adabas sessions have created other Protection Log data files.

PLOG-number and extent-number specify the Adabas Protection Log number and the extent number of the Protection Log file to be closed. These numbers are displayed by the LAYOUT function of ADADEV.



Note: This function only applies to UNIX platforms.

Example:

```
adarec: db=1
%ADAREC-I-DBON, database 1 accessed online
adarec: close=93
adarec:

Protection log 93 - 20-JUL-2005 13:12:54 closed successfully
```

The CLOSE function closes Protection Log 93 of database 1.

DBID

```
DBID = number
```

This parameter selects the database to be used.



Note: Program functions which do not require the nucleus to be running need the environment variables/logical names set for the container files.

LIST

```
[PLOG=number,] LIST = keyword
```

Valid keywords are BRIEF, FULL and RESTART. BRIEF lists the Protection Log number and its creation date. FULL lists additional information about the records on the Protection Log, e.g. the checkpoints, the number of modifications for each file, etc. RESTART displays the restart points that ADAREC writes when it encounters checkpoints while processing.

The LIST=FULL function also checks the structure of the Protection Log to ensure that it is internally consistent. If a structural error is detected, a message is output indicating the error type as well as the record and block numbers.

If the Protection Log is within a disk section, the PLOG parameter must be set before LIST can be specified.

Examples

```
adarec: list=brief
```

```
Protection log 1 - 26-OCT-2006 11:39:03
```

The creation date of PLOG 1 is displayed.

REGENERATE

This function is used to regenerate a whole database or files within a database.

Database Regeneration

```
REGENERATE = *, PLOG = number
    [,EXCLUDE_FILES = (number[-number][,number[-number]]...)]
    [, [NO]BI_CHECK]
    [,BLOCK = ([number][,number]),
        CHECKPOINT = ([string][,string])]
    [, [NO]ERROR_LOG]
    [,ON_ERROR = keyword]
```

This option of the REGENERATE function regenerates a database. A file exclusion list can be used to exclude certain files from the regenerate. ET logic is supported.

During REGENERATE processing, ADAREC sets the database to utility-only mode. Processing terminates if a SYN checkpoint is encountered. In this case, ADAREC inspects the Protection Log in order to calculate an alternative restart point. This restart point is then displayed together with a list of utility functions that must be executed before processing can be continued. The next call to REGENERATE automatically sets up at this point. The use of the calculated restart point

can be overridden by specifying "BLOCK=" or "CHECKPOINT=" (that is, supplying empty values for these keywords). This procedure is repeated until the end of the PLOG is reached. After ADAREC has terminated, the database remains in utility-only mode, because more calls to REGENERATE may follow. After the database regeneration has finished, you can enable the database for normal processing with the ADAOPR command `OPTIONS=NOUTILITIES_ONLY`.

[NO]BI_CHECK

If this option is set to `BI_CHECK`, ADAREC checks the consistency of the before images in the Protection Log against the data in the database (is the ISN in use; does the record exist; is there a before image mismatch?). If a mismatch is encountered, ADAREC issues messages containing the relevant information and does not perform the update.

If this option is set to `NOBI_CHECK`, the consistency check is still made and the `ERROR_LOG` is implicitly enabled; however, on finding a BI inconsistency, the update is made and the mismatch is reported to the `ERROR_LOG` (see below). If errors are encountered, only the first error for each file will be displayed, all subsequent errors are logged to the `ERROR_LOG`. Note that the index might become inconsistent in this case.

However, if the PLOG was written with the `NOBI` option of the nucleus, it will not contain any before images and the `BI_CHECK` option cannot be set.

The default is `BI_CHECK`.

BLOCK = ([number],[number])

This parameter specifies the numbers of the blocks in the Protection Log files that contain the corresponding checkpoint names. The block numbers can be taken from `ADAREC LIST=FULL`.

CHECKPOINT = ([string],[string])

This parameter specifies the starting and ending checkpoint names. The checkpoint names can be taken from the ADAREP database status report or `ADAREC LIST=FULL`.

If processing is to start at the beginning of the Protection Log file, the first parameter must be omitted.

[NO]ERROR_LOG

Setting this option to `ERROR_LOG` enables the automatic logging of any BI inconsistencies that may be detected when using the `NOBI_CHECK` option. The contents of the error file produced can be examined using the `ADAERR` utility. Do not print this error file using the standard operating system print utilities, since the records contain nonprintable characters. See *ADAERR* for further information.

The default is `NOERROR_LOG`.

EXCLUDE_FILES = (number[-number][,number[-number]]...)

This parameter specifies the files to be excluded when regenerating a complete database. The updates that are excluded are written to a report.

ON_ERROR = keyword

Valid keywords are ABORT and EXCLUDE. The keyword used determines what action to take if ADAREC detects non-fatal errors during processing (e.g. response code 17, file not loaded). ABORT abnormally terminates regenerate processing, and EXCLUDE excludes the file in question from the regenerate if Data Storage errors occur (nucleus response codes 17, 49, 75, 77 and 113).

If, however, an error occurs while updating a file's index (nucleus response codes 75, 76, 77, 98, 165, 166, 167 and 176), only the regeneration of the Data Storage for this file will continue. When the regeneration process is complete, the index of this file is marked as invalid. The ADAINV REINVERT function with the ALL_FIELDS option then has to be run for this file (please refer to the ADAINV utility in this manual for more detailed information). If index errors occur and if the regenerate includes several Protection Logs, all of the Protection Logs should be processed before reinverting the index. Reinverting the index each time a Protection Log results in index errors would waste considerable amounts of time and computer resources.

The default is ON_ERROR=EXCLUDE.

PLOG = number

This parameter specifies the log number of the Adabas Protection Log to be used as input for the REGENERATE function. This number can be found with ADAREC using the LIST = BRIEF function.

File Regeneration

```
REGENERATE = (number[-number][,number[-number]]...), PLOG = number
             [, [NO]BI_CHECK]
             [, BLOCK = ([number][,number]),
                       CHECKPOINT = ([string][,string])]
             [, [NO]ERROR_LOG]
             [, ON_ERROR = keyword]
```

This option of the REGENERATE function re-applies all updates in a Protection Log for the specified files or ranges of files. LOB files specified are ignored, but the LOB files assigned to all base files specified are dumped too.

During regenerate processing, ADAREC locks the files for exclusive use. The regenerate terminates if a SYNCP checkpoint is found while processing a protection log. In this case, ADAREC inspects the Protection Log in order to calculate an alternative restart point. This restart point is then displayed with a list of utility functions that must be executed before processing can be continued. The next call to REGENERATE automatically sets up at this point. The use of the calculated restart point can be overridden by specifying "BLOCK=" or "CHECKPOINT=" (that is, supplying empty

values for these keywords). This procedure is repeated until the end of the Protection Log is reached.

The files remain locked, because more calls to REGENERATE may follow. After the files regeneration is finished, you must unlock the files with the ADAOPR command UNLOCK.

The following functions are not allowed while ADAREC is active:

- ADAOPR ET_SYNC FEOF = PLOG
- ADABCK DUMP
- ADAOPR STOP to a sub-user while the associated ADAREC user exists

[NO]BI_CHECK

If this option is set to BI_CHECK, ADAREC checks the consistency of the before images in the Protection Log against the data in the database (is the ISN in use; does the record exist; is there a before image mismatch?). If a mismatch is encountered, ADAREC issues messages containing the relevant information and does not perform the update.

If this option is set to NOBI_CHECK, the consistency check is still made and the ERROR_LOG is implicitly enabled; however, on finding a BI inconsistency, the update is made and the mismatch is reported to the ERROR_LOG (see below). If errors are encountered, only the first error for each file will be displayed, all subsequent errors are logged to the ERROR_LOG. Note that the index might become inconsistent in this case.

NOBI_CHECK improves performance at the expense of possible loss of data consistency. We advise you therefore not to use NOBI_CHECK for mission critical databases.

The default is BI_CHECK.

BLOCK = ([number] [,number])

This parameter specifies the blocks in the Protection Log files that contain the corresponding checkpoint names. The block numbers can be taken from ADAREC LIST=FULL.

CHECKPOINT = ([string] [,string])

This parameter specifies the starting and ending checkpoint names. The checkpoint names can be taken from the ADAREP database status report.

If processing is to start at the beginning of the Protection Log file, the first parameter must be omitted. However, if the first checkpoint name is supplied, it must be found in the first Protection Log file.

If processing is to stop at the end of the last Protection Log file, the second checkpoint name must be omitted.

[NO]ERROR_LOG

Setting this option to ERROR_LOG enables the automatic logging of any BI inconsistencies that may be detected when using the NOBI_CHECK option. The contents of the error file produced can be examined using the ADAERR utility. Do not print this error file using the standard OpenVMS print utilities, since the records contain non-printable characters. Please refer to the ADAERR utility in this manual for more detailed information.

The default is NOERROR_LOG.

ON_ERROR = keyword

Valid keywords are ABORT and EXCLUDE. The keyword used determines what action to take if ADAREC detects non-fatal errors during processing (e.g. response code 17, file not loaded). ABORT abnormally terminates regenerate processing, and EXCLUDE excludes the file in question from the regenerate if Data Storage errors occur (nucleus response codes 17, 49, 75, 77 and 113).

If, however, an error occurs while updating a file's index (nucleus response codes 75, 76, 77, 98, 165, 166, 167 and 176), only the regeneration of the Data Storage for this file will continue. When the regeneration process is complete, the index of this file is marked as invalid. The ADAINV REINVERT function with the ALL_FIELDS option then has to be run for this file (please refer to the ADAINV utility in this manual for more detailed information). If index errors occur and if the regenerate includes several Protection Logs, all of the Protection Logs should be processed before reinverting the index. Reinverting the index each time a Protection Log results in index errors would waste considerable amounts of time and computer resources.

The default is ON_ERROR=EXCLUDE.

PLOG = number

This parameter specifies the log number of the Adabas Protection Log to be used as input for the REGENERATE function. This number can be found with ADAREC using the LIST = BRIEF function.

Examples

Example 1

In this example, database 2 is to be regenerated using the Protection Log 2. File 12 is to be excluded from the regenerate.

```
adarec: regenerate=*,plog=2
adarec: exclude_files=12
adarec:

Protection log 2 - 26-OCT-2006 11:48:59

Block      3 - checkpoint SYNC - 11:49:00 - USERID ADANUC <version>
%ADAREC-I-CHKIGN, Checkpoint ignored

The following utility functions were executed in the original session:

Block      4 - checkpoint SYNP - 11:50:02 - USERID ADADBAM REFRESH=13

Block      5 - checkpoint SYNX - 11:50:03 - USERID ADADBAM RESET=UCB,IDENT=7

Block      6 - checkpoint SYNP - 11:50:03 - USERID ADADBAM RECOVER

Re-execute all SYNP utility functions starting from block 4.

REGENERATE summary

Calculated RESTART point - BLOCK=6,CHECKPOINT=SYNP
```

Processing of the Protection Log terminated at the SYNP checkpoint in block 4. However, no updates were found on looking ahead and processing can be continued from the calculated restart point in block 6. ADAREC displays a list of the utility functions that must be executed before processing continues. The next call to REGENERATE=* will automatically continue at this calculated restart point.

```
adarec: regenerate=*,plog=2
%adarec-I-restartp, calculated restart point - block=6,checkpoint=synp
adarec: exclude_files=12
adarec:

Protection log 2 - 26-OCT-2006 11:48:59.86

Block      6 - checkpoint SYNP - 11:50:03.86 - USERID ADADBAM RECOVER
%ADAREC-I-CHKSTP, starting checkpoint

    1 modifications in file 11
    1 modifications EXCLUDED from file 12

    4 ET commands issued

Block      7 - checkpoint SYNC - 11:52:38.98 - USERID ADANUC SHUTDOWN
%ADAREC-I-CHKIGN, Checkpoint ignored
```

```
REGENERATE summary
```

```
Protection log 2 processed
```

Processing of the Protection Log continues at the calculated restart point. The regenerate terminates successfully.

Example 2

In this example, database 2 is to be regenerated using the Protection Log 2. Processing is to start at the checkpoint SYNPN in block 6 of the Protection Log. If Data Storage errors occur, the file in question will be excluded from the regenerate. If index errors occur, the file's index will be excluded from the regenerate and marked as invalid.

```
adarec: regenerate=*,plog=2,block=6,checkpoint=synp
adarec: on_error=exclude
adarec:

Protection log 2 - 26-OCT-2006 11:48:59.86

%ADAREC-W-UTIENA, OPTIONS=UTILITIES enabled in nucleus by ADAREC
%ADAREC-W-RECUPD, Updates performed between Nucleus and REGENERATE'S startup
%ADAREC-W-RECCMD, 1 N1  command(s)

Block      6 - checkpoint SYNPN - 11:50:03.86 - USERID ADADBM RECOVER
%ADAREC-I-CHKSTP, starting checkpoint

      1 modifications in file 11

      3 ET commands issued

%ADAREC-E-ISNINUSE, ISN 774 in use in file 12
%ADAREC-I-PLOGRB, from record 14 in block 7 in PLOG 2
%ADAREC-I-UPDEXC, ALL following updates in file 12 will be EXCLUDED

      1 modifications EXCLUDED from file 12

      1 ET command issued

Block      7 - checkpoint SYNC - 11:52:38.98 - USERID ADANUC SHUTDOWN
%ADAREC-I-CHKIGN, Checkpoint ignored

REGENERATE summary

Protection log 2 processed
```

An ISN conflict occurred in file 12 and all subsequent updates to this file were excluded. The cause of the error has to be investigated. However, the nucleus was started without 'OPTIONS=UTILITIES_ONLY' and an N1 command was issued before the regenerate was started.

The Protection Log was processed to its end, the abort system message is used only to indicate a fatal error.

Example 3

This example is similar to the previous one, with the exception that processing will abort if Data Storage or index errors are encountered.

```
adarec: regenerate=*,plog=2,block=6,checkpoint=synp
adarec: on_error=abort
adarec:

Protection log 2 - 26-OCT-2006 11:48:59.86

%ADAREC-W-UTIENA, OPTIONS=UTILITIES enabled in nucleus by ADAREC
%ADAREC-W-RECUPD, Updates performed between Nucleus and REGENERATE'S startup
%ADAREC-W-RECCMD, 1 N1  command(s)

Block      6 - checkpoint SYNCP - 11:50:03.86 - USERID ADADBM RECOVER
%ADAREC-I-CHKSTP, starting checkpoint

      1 modifications in file  11

      3 ET commands issued

%ADAREC-E-ISNINUSE, ISN 774 in use in file  12
%ADAREC-I-PLOGRB, from record 14 in block 7 in PLOG 2
```

An ISN conflict occurred in file 12 and further processing was aborted.

Example 4

In this example, database 2 is to be regenerated using the Protection Log 3. The before images in the Protection Log will be checked against the data in the database and mismatches will be displayed on the terminal.

```
adarec: regenerate=*,plog=3
adarec:

Protection log 3 - 26-OCT-2006 12:10:25.12

%ADAREC-W-UTIENA, OPTIONS=UTILITIES enabled in nucleus by ADAREC

Block      1 - checkpoint SYNC - 12:10:25.12 - USERID ADANUC 3.2/0 PL 0
```



```

%ADAREC-I-CHKIGN, Checkpoint ignored

    1 ET command issued

%ADAREC-E-RECMIS, Before image mismatch for ISN 3 in file 11
%ADAREC-I-PLOGRB, from record 7 in block 2 in PLOG 3
%ADAREC-I-UPDEXC, ALL following updates in file 11 will be EXCLUDED

    1 modifications EXCLUDED from file 11
    1 modifications in file 12

    3 ET commands issued

Block      2 - checkpoint SYNC - 12:11:44.30 - USERID ADANUC SHUTDOWN
%ADAREC-I-CHKIGN, Checkpoint ignored

REGENERATE summary

Protection log 3 processed

```

One before image mismatch occurred during processing. As a result, one update was excluded from file 11.

Having restored the files, the same example can be rerun with no consistency check of the before images and with BI error logging enabled.

The Protection Log was processed to its end, the abort system message is used only to indicate a fatal error.

```

adarec: regenerate=*,plog=3,nobi_check
adarec:

Protection log 3 - 26-OCT-2006 12:10:25.12

%ADAREC-W-UTIENA, OPTIONS=UTILITIES enabled in nucleus by ADAREC

Block      1 - checkpoint SYNC - 12:10:25.12 - USERID ADANUC 3.2/0 PL 0
%ADAREC-I-CHKIGN, Checkpoint ignored

%ADAREC-W-RECMIS, Before image mismatch for ISN 3 in file 11
%ADAREC-I-PLOGRB, from record 7 in block 2 in PLOG 3

    1 modifications in file 11
    1 modifications in file 12

    4 ET commands issued

    1 BI_CHECK error in file 11

```

```
Block      2 - checkpoint SYNC - 12:11:44.30 - USERID ADANUC SHUTDOWN
%ADAREC-I-CHKIGN, Checkpoint ignored
```

```
REGENERATE summary
```

```
    1 BI_CHECK error in file  11
```

```
Protection log 3 processed
```

One BI_CHECK error occurred during processing.

The Protection Log was processed to its end, the abort system message is used only to indicate a fatal error.

The source of the errors is written to an error file which can be displayed using the ADAERR utility. The first error is logged and also written to the error file. All subsequent errors are written to ERROR_LOG.

The following error file was produced:

```
%ADAERR-E-RECMIS, Before image mismatch for ISN 3 in file  11
%ADAERR-I-PLOGRB, from record 7 in block 2 in PLOG 3
```

Example 5

In this example, database 2 is to be regenerated using the Protection Log 3.

```
adarec: regenerate=*,plog=3
adarec:
```

```
Protection log 3 - 26-OCT-2006 12:10:25.12
```

```
%ADAREC-W-UTIENA, OPTIONS=UTILITIES enabled in nucleus by ADAREC
```

```
Block      1 - checkpoint SYNC - 12:10:25.12 - USERID ADANUC 3.2/0 PL 0
%ADAREC-I-CHKIGN, Checkpoint ignored
```

```
%ADAREC-E-ERRIUP, Error response 165 during index update
%ADAREC-E-Adabas_165, * Invalid descriptor name in DVT
%ADAREC-I-DESNAM, Descriptor name XA
%ADAREC-I-ISNFILE, from ISN 3 in file  11
%ADAREC-I-PLOGRB, from record 7 in block 2 in PLOG 3
%ADAREC-I-REINVERT, REINVERT all descriptors to re-establish INDEX
%ADAREC-I-REGDAT, Regenerating ONLY data-storage for file  11
```

```
    1 modifications in file  11
```

```

1 modifications in file 12

4 ET commands issued

Block      2 - checkpoint SYNC - 12:11:44.30 - USERID ADANUC SHUTDOWN
%ADAREC-I-CHKIGN, Checkpoint ignored

REGENERATE summary

Protection log 3 processed

```

An invalid descriptor name was encountered during processing. As a result, only the data storage of file 11 was regenerated. All of the descriptors will have to be reinverted in order to reestablish the index.

The Protection Log was processed to its end, the abort system message is used only to indicate a fatal error.

If index errors occur and if the regenerate includes several Protection Logs, all of the Protection Logs should be processed before reinverting the index. Reinverting the index each time a Protection Log results in index errors would waste considerable amounts of time and computer resources.

Example 6

In this example, database 2 is to be regenerated using the Protection Log 4 after the regenerate processing of Protection Log 3 resulted in an index error.

```

adarec: regenerate=*,plog=4
adarec:

Protection log 4 - 26-OCT-2006 12:12:00.15

Block      1 - checkpoint SYNC - 12:12:00.15 - USERID ADANUC <version>
%ADAREC-I-CHKIGN, Checkpoint ignored

%ADAREC-E-FCBNAC, file 11's index not accessible
%ADAREC-I-REGDAT, Regenerating ONLY data-storage for file 11

1 modifications in file 11
1 modifications in file 12

4 ET commands issued

Block      2 - checkpoint SYNC - 12:12:19.35 - USERID ADANUC SHUTDOWN
%ADAREC-I-CHKIGN, Checkpoint ignored

```

```
REGENERATE summary
```

```
Protection log 4 processed
```

The index error that occurred while processing Protection Log 3 (see example 5) means that file 11's index is no longer accessible. Only the Data Storage of file 11 is regenerated, whereas both the Data Storage and the index of file 12 are regenerated.

The Protection Log was processed to its end, the abort system message is used only to indicate a fatal error.

ADAREC Restart Considerations

An interrupted ADAREC run which leaves a UCB entry has to be re-started from the beginning. Because modifications have already been made, a RESTORE database or RESTORE file has to be executed before re-starting ADAREC. However, if there is no UCB entry, the database has not been modified and ADAREC can be re-started.

An abnormally terminated ADAREC (RESTORE/RECOVER) leaves the database in a consistent state, although it is not possible to tell exactly in which state. ADAREC cannot determine which transactions have already been recovered, so it is necessary to repeat the RESTORE operation and restart the ADAREC from the beginning in order to ensure that everything is recovered.

Having performed the first update, ADAREC writes a 'started' checkpoint to the checkpoint file, e.g.

```
SYNX    22-MAR-2007 16:49:46    192    ADAREC REG STARTED
```

24 ADAREP (Database Report)

- Functional Overview 342
- Procedure Flow 343
- Checkpoints 343
- Control Parameters 344

This chapter describes the utility "ADAREP".

Functional Overview

The ADAREP utility generates the database status report. This contains information about the current physical layout and logical contents of the database. Unless otherwise stated, the functions can be executed when the nucleus is active or inactive.

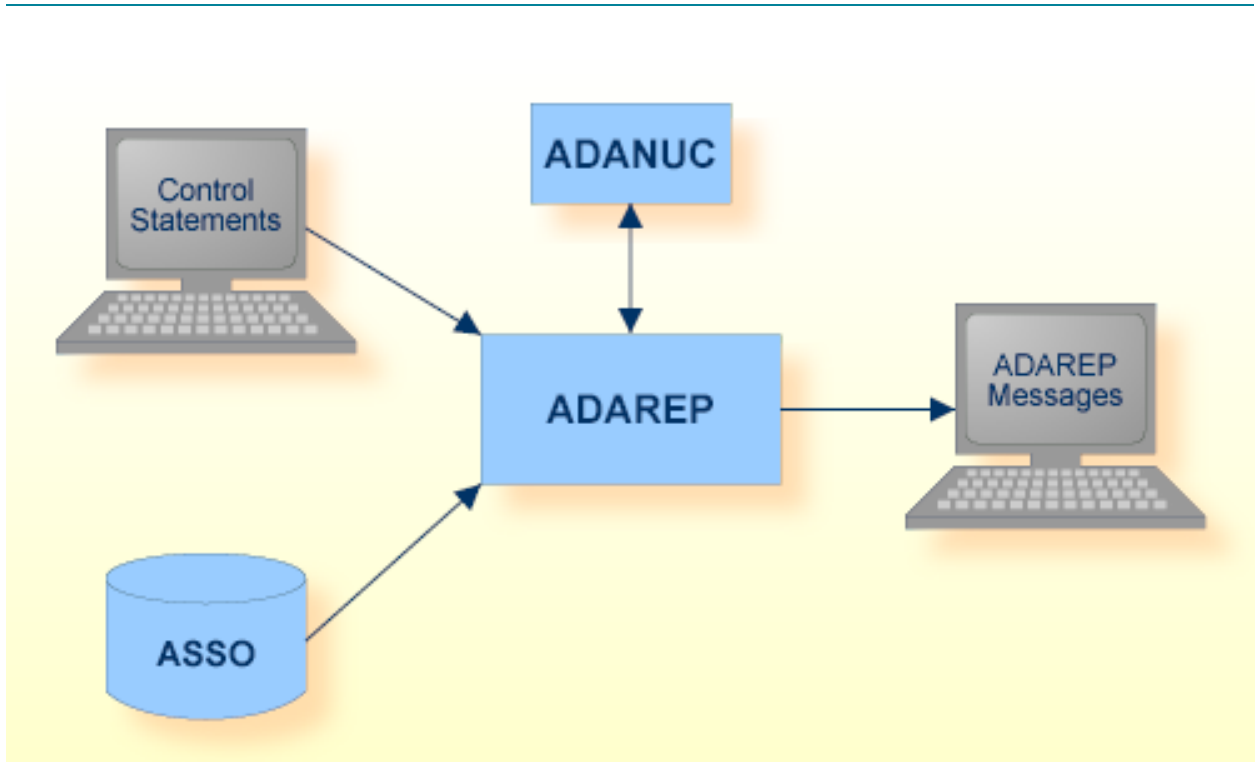
The information contained in this report includes:

- The amount and location of space currently allocated for the Associator and Data Storage;
- The amount and location of unused space available for the Associator and Data Storage;
- Database file summary;
- Checkpoint information;
- Information about each file in the database (space allocation, space available, number of records loaded, MAXISN setting, field definitions, etc.);

Only the CHECKPOINTS control parameter (see description below) requires the nucleus to be active.

This utility is a multi-function utility.

Procedure Flow



Data Set	Environment Variable/ Logical Name	Storage Medium	Additional Information
Associator	ASSOx	Disk	
Control statements	stdin/ SYS\$INPUT		Utilities Manual
ADAREP report	stdout/ SYS\$OUTPUT		Messages and Codes, Utilities Manual

Checkpoints

The utility writes no checkpoints.

Control Parameters

The following control parameters are available:

```
CHECKPOINTS = { * | ( [absolute-date] [, [absolute-date] ] ) }  
CONSTRAINTS  
CONTENTS  
M DBID = number  
D [NO]FDT  
FILES = { * | ( number [-number] [, number [-number]]... ) }  
D [NO]FULL  
FREE_SPACE  
LAYOUT  
SUMMARY
```

CHECKPOINTS

```
CHECKPOINTS = { * | ( [absolute-date] [, [absolute-date]] ) }
```

This function displays selected information from the checkpoint list and requires the nucleus to be active.

Five types of system checkpoints (SYNP, SYNC, SYNX, OPEN and CLSE) are written to the checkpoint file and to the protection log, together with the user checkpoints written by C1 commands.

SYNC indicates a checkpoint made during nucleus initialization, shutdown or cancel processing; during the ADAOPR function FEOF = PLOG; or due to ADABCK NEW_PLOG processing.

SYNP indicates a checkpoint made by an Adabas utility that requires privileged control, i.e. the module can make updates without using the nucleus. A SYNP checkpoint is, for example, written at the end of an ADAMUP UPDATE run.

SYNX indicates a checkpoint made by a utility that requires exclusive control of one or more files. A SYNX checkpoint is, for example, written by ADAULD.

An OPEN checkpoint is written by the OP command of EXU/EXF users.

A CLSE checkpoint is written by the CL command of EXU/EXF users.



Note: If the ADAREC `REGENERATE' function is executed using the Protection Log, this utility stops at each SYN* checkpoint since DBA intervention is required.

If an asterisk `*' is entered, all checkpoints are displayed.

The date strings must correspond to the following absolute data and time format:

```
dd-mmm-yyyy[:hh[:mm[:ss]]]
```

Leading zeroes in the date and time specification may be omitted. Any numbers not specified are set to 0, for example 28-jul-2006 is equivalent to 28-jul-2006:00:00:00.

The following table shows the possible values for parameter CHECKPOINTS, and the corresponding checkpoints displayed by this value:

Value specified for parameter CHECKPOINTS	Checkpoints displayed for this specification
* or (,)	All checkpoints
absolute-date	Only the checkpoints written exactly at the date and time specified
(absolute-date,)	Checkpoints written from date and time specified onwards
(,absolute-date)	Checkpoints written up to the date and time specified
(absolute-date,absolute-date)	Checkpoints written from first date and time value specified onwards up to the second date and time value specified

Example

```
adarep: checkpoints=*
```

Name	Date/Time	Session	User Id / Function
----	-----	-----	-----
SYNP	28-JUL-2006 12:50:34	8	ADADBM DELCP
SYNX	28-JUL-2006 12:50:36	8	ADABCK DUMP=* STARTED
SYNX	28-JUL-2006 12:50:37	8	ADABCK DUMP=*
OPEN	28-JUL-2006 17:23:53	8	otto
OPEN	28-JUL-2006 17:24:15	8	otto
CLSE	28-JUL-2006 17:24:24	8	otto

All checkpoints are displayed.

The column "User ID / Function" contains

- for user checkpoints created via OP/CL commands for EXU/EXF users or via C1 command: the user specified in the Additions 1 field of the relevant OP command;
- for utility checkpoints: the utility function executed.

Taking the output of the example above (checkpoints=*), the selection criteria can be used to filter the checkpoints selected as shown below.

Specifying

```
checkpoints=28-jul-2006:12:50:36
```

will produce the following output:

Name	Date/Time	Session	User Id / Function
SYNX	28-JUL-2006 12:50:36	8	ADABCK DUMP=* STARTED

Specifying

```
checkpoints=(28-jul-2006:12:50:36,)
```

will produce the following output:

Name	Date/Time	Session	User Id / Function
SYNX	28-JUL-2006 12:50:36	8	ADABCK DUMP=* STARTED
SYNX	28-JUL-2006 12:50:37	8	ADABCK DUMP=*
OPEN	28-JUL-2006 17:23:53	8	otto
OPEN	28-JUL-2006 17:24:15	8	otto
CLSE	28-JUL-2006 17:24:24	8	otto

Specifying

```
checkpoints=(,28-jul-2006:12:50:36)
```

will produce the following output:

Name	Date/Time	Session	User Id / Function
----	-----	-----	-----
SYNP	28-JUL-2006 12:50:34	8	ADADBM DELCP
SYNX	28-JUL-2006 12:50:36	8	ADABCK DUMP=* STARTED

Specifying

```
checkpoints=(28-jul-2006:17, 28-jul-2006:17:24)
```

will produce the following output:

Name	Date/Time	Session	User Id / Function
----	-----	-----	-----
OPEN	28-JUL-2006 17:23:53	8	otto

CONSTRAINTS

CONSTRAINTS

This function displays information about all referential constraints in the database that you specify with the DBID parameter.

Example

```
adarep: constraints
```

Primary file		Foreign file	Name	Action
-----		-----	-----	-----
9 (AA)	<---	12 (AC)	HO	DX UX

The referential constraint HO links the primary key field AA in the primary file 9 with the foreign key field AC in the foreign file 12. The associated actions are delete no action (DX) and update no action (UX).

CONTENTS

CONTENTS

This function displays information about the files in the database that you specify with the DBID parameter.

Example

```
adarep: contents
```

Content of Database 76 25-AUG-2006 12:51:06

File	Filename	Loaded on	Top ISN	Index level	Extents				Pad %		flags
					N	U	A	D	A	D	
1	CHECKPOINT-FILE	25-AUG-2006	4	3	1	1	1	1	5	5	S
2	USER-DATA-FILE	25-AUG-2006	0	3	1	1	1	1	5	5	IS
3	SECURITY-FILE	25-AUG-2006	0	3	1	1	1	1	5	5	IS
11	EMPLOYEE-FILE	12-AUG-2006	72	3	20	12	4	9	5	5	I
12	VEHICLES	12-AUG-2006	171	3	1	1	5	9	5	5	IS
13	MISCELLANEOUS	12-AUG-2006	179	3	2	1	1	1	3	5	IS

File	Filename	Allocated blocks				Unused blocks		
		NI	UI	AC	DS	NI	UI	DS
1	CHECKPOINT-FILE	1	1	5	32	1	0	31
2	USER-DATA-FILE	24	2	5	57	24	1	56
3	SECURITY-FILE	2	2	1	5	2	1	4
11	EMPLOYEE-FILE	109	26	4	68	97	13	64
12	VEHICLES	40	20	6	48	30	12	44
13	MISCELLANEOUS	113	100	3	50	33	32	1
Total		289	151	24	260	187	59	200

The column 'Extents' shows the number of logical extents currently assigned to the Normal Index (N), the Main/Upper Indices (U), the Address Converter (A) and Data Storage (D).

The column 'Pad' shows the block padding factors in percent defined for the Associator (A) and Data Storage (D) (please refer to the ASSOPFAC and DATAPFAC parameters of ADAFDU, ADAMUP or ADAORD for more detailed information).

The column 'Flags' contains the following information:

Subcolumn	Flag	Meaning
A	A	Indicates an Adam file
L	L	File is a LOB file
	B	File is a base file with a corresponding LOB file
R	R	ISN and space reusage enabled for the file
	I	ISN reusage, but no space reusage enabled for the file

Subcolumn	Flag	Meaning
	S	Space reusage, but no ISN reusage enabled for the file
C	C	Ciphering enabled for the file
P	P	Program Refresh enabled for the file

If ISNs are to be reused, the ISNs of deleted records can be reassigned to new records. If space is to be reused, the space released within a block as a result of deleting a record can be reused for a new record (please refer to the REUSE parameter of ADADBAM or ADAFDDU for more detailed information).

The second table shows the number of blocks allocated for Normal Index (NI) Main/Upper Indices (UI), Address Converter (AC) and Data Storage (DS) for each file. The remaining columns show the number of unused blocks in the Main/Upper Indices (UI) and Data Storage (DS).

DBID

```
DBID = number
```

This parameter selects the database to be used. Multiple DBIDs are supported within one session.

The DBID parameter must be the first ADAREP parameter specified.

Example

```
adarep: dbid = 1, contents
      .
      .
adarep: dbid = 2, contents
      .
      .
adarep: dbid = 3, contents
```

[NO]FDT

```
[NO]FDT
```

If this parameter is set to FDT, the Field Definition Tables (FDTs) will be included in the status information subsequently displayed by the FILES function.

The default is NOFDT.

FILES

FILES = { * | (number [-number][,number [-number]]...) }

This function displays status information for the files selected.

Example

```

adarep: fdt
adarep: file=9
Database 1, File      9 (EMPLOYEES      )          28-JAN-2010 14:15:54

Highest Index Level:      3      Padding Factors:      ASSO  5%, DATA  5%
Top ISN:                  1,272  Maximum ISN expected:      8,191
Records loaded:          1,272  Corresponding LOB file:      14
Last FDT Modification:    11-NOV-2009 13:53:32.447000

ISN reusage:      Enabled, inactive  Space reusage:      Enabled
Program refresh:  Disabled           Ciphering:          Disabled

Container  Block  Extent  Extents in Blocks  Allocated  Unused
File       Size  Type   from      to        Blocks  MB   Blocks  MB
-----
ASSO:
  2 32KB AC      2,562    2,562      1  0      0  0
  1 4KB  NI        63      152      90  0      35  0
  1 4KB  UI       153      167      15  0       0  0

DATA:
  1 32KB DS        14      45      32  1      23  0
-----

Field Definition Table:

Level | I Name | I Length | I Format | I Options | I Flags | I Encoding
-----|-----|-----|-----|-----|-----|-----
1     | I A0  | I       | I       | I         | I       | I
2     | I AA  | I 8     | I A     | I DE,UQ,NC,NN | I RP   | I
2     | I AB  | I       | I       | I         | I       | I
3     | I AC  | I 4     | I F     | I DE     | I       | I
3     | I AD  | I 8     | I B     | I NU,HF  | I       | I
3     | I AE  | I 0     | I A     | I NU,NV,NB,LA | I       | I
1     | I B0  | I       | I       | I         | I       | I
2     | I BA  | I 40    | I W     | I NU     | I       | I
2     | I BB  | I 40    | I W     | I NU     | I       | I
2     | I BC  | I 50    | I W     | I DE,NU  | I SP   | I
1     | I CA  | I 1     | I A     | I FI     | I       | I
    
```

1	I	DA	I	1	I	A	I	FI	I	I
1	I	EA	I	4	I	P	I	DE,NC	I	I
1	I	FO	I		I		I	PE	I	I
2	I	FA	I	60	I	W	I	NU,MU	I	I
2	I	FB	I	40	I	W	I	DE,NU	I	I
2	I	FC	I	10	I	A	I	NU	I	I
2	I	FD	I	3	I	A	I	NU	I	I
2	I	F1	I		I		I		I	I
3	I	FE	I	6	I	A	I	NU	I	I
3	I	FF	I	15	I	A	I	NU	I	I
3	I	FG	I	15	I	A	I	NU	I	I
3	I	FH	I	15	I	A	I	NU	I	I
3	I	FI	I	80	I	A	I	DE,NU,MU	I	I
1	I	IO	I		I		I	PE	I	I
2	I	IA	I	40	I	W	I	NU,MU	I	I
2	I	IB	I	40	I	W	I	DE,NU	I	I
2	I	IC	I	10	I	A	I	NU	I	I
2	I	ID	I	3	I	A	I	NU	I	I
2	I	IE	I	5	I	A	I	NU	I	I
2	I	I1	I		I		I		I	I
3	I	IF	I	6	I	A	I	NU	I	I
3	I	IG	I	15	I	A	I	NU	I	I
3	I	IH	I	15	I	A	I	NU	I	I
3	I	II	I	15	I	A	I	NU	I	I
3	I	IJ	I	80	I	A	I	DE,NU,MU	I	I
1	I	JA	I	6	I	A	I	DE	I	SB,SP
1	I	KA	I	66	I	W	I	DE,NU	I	I
1	I	LO	I		I		I	PE	I	I
2	I	LA	I	3	I	A	I	NU	I	SP
2	I	LB	I	6	I	P	I	NU	I	SP
2	I	LC	I	6	I	P	I	DE,NU,MU	I	I
1	I	MA	I	4	I	G	I	NU	I	I
1	I	NO	I		I		I		I	I
2	I	NA	I	2	I	U	I		I	SP
2	I	NB	I	3	I	U	I	NU	I	SP
1	I	OO	I		I		I	PE	I	I
2	I	OA	I	8	I	U	I	NU	I	I
								DT=E(
2	I	OB	I	8	I	U	I	NU	I	I
								DT=E(
1	I	PA	I	3	I	A	I	DE,NU,MU	I	I
1	I	QA	I	7	I	P	I		I	I
1	I	RA	I	0	I	A	I	NU,NV,NB,LB	I	I
1	I	SO	I		I		I	PE	I	I
2	I	SA	I	80	I	W	I	NU	I	I
2	I	SB	I	3	I	A	I	NU	I	I
2	I	SC	I	0	I	A	I	NU,MU,NV,NB,LB	I	I

Type	I	Name	I	Length	I	Format	I	Options	I	Parent field(s)	Fmt
COLL	I	CN	I	11,144	I		I	NU,HE	I	BC	de@collation=phonebook

	I	I	I	I	I	I	I	PRIMARY
SUPER	I H1	I 5	I B	I NU	I NA (1 - 2)	U	I NB (1 - 3)	U
SUB	I S1	I 2	I A	I	I JA (1 - 2)	A		
SUPER	I S2	I 46	I A	I NU	I JA (1 - 6)	A	I BC (1 - 40)	W
SUPER	I S3	I 9	I A	I NU,PE	I LA (1 - 3)	A	I LB (1 - 6)	P
Referential Integrity								
Type	I Name	I Refer.	I Primary	I Foreign	I Rules			
	I	I file	I field	I field	I			
PRIMARY	I H0	I 12	I AA	I AC	I DELETE_NOACTION		UPDATE_NOACTION	

The FILES parameter displays the file number and file name, the highest index level, the padding factors for ASSO and DATA, the highest and maximum ISNs, the number of records loaded, the corresponding base file number or LOB file number if it exists, as well as the switches for ISN re-usage, space reusage, program refresh and ciphering. The time and date of the last FDT modification are also displayed.

The layout of the ASSO and DATA elements of a file are displayed: the block size on which the various list elements are stored, the location of these extents and the number of corresponding blocks/megabytes allocated or unused.

In addition, the FDT function displays the Field Definition Table of the file.

The flags which can be displayed in the Field Definition Table are as follows:

Flag	Meaning
HY	the field is part of a hyperdescriptor
P	the field is phoneticized
SB	part of this field is subdescriptor
SP	part of superdescriptor

FREE_SPACE

FREE_SPACE

This function displays a summary of free blocks in ASSO and DATA. This is a subset of the information that is displayed by the LAYOUT function.

Example

```
adarep: free_space
```

```
Free space of Database 76      28-NOV-2006 12:51:24
```

Container File	Extents from	in Blocks to	Number of Blocks	Block Size

ASSO:				
1-2	611	1,546	936	2,048
DATA:				
1	245	768	524	4,096
2	769	868	100	3,072
3-4	869	888	20	6,144

[NO]FULL

[NO]FULL

If FULL is specified together with FDT, dropped fields will be included in the display of the FDT information (but without the field names).

The default is NOFULL.

LAYOUT

LAYOUT

This function displays a summary of the blocks in ASSO and DATA and reports lost blocks. Lost blocks are blocks that are not listed in the Free Space Table (FST) and are not allocated to a file, the DSST or the database's global area. This function also reports double-allocated blocks.

Example

adarep: layout

Layout of Database 76

28-NOV-2006 12:51:24

Container File	Extents in Blocks from	to	Number of Blocks	Block Size	Extent Type	File Number
-------------------	---------------------------	----	---------------------	---------------	----------------	----------------

ASSO:

1	1	30	30	4,096	CB	
1	31	31	1	4,096	FCB	1
1	32	32	1	4,096	FDT	1
1	33	35	3	4,096	AC	1
1	36	36	1	4,096	UI	1
1	37	37	1	4,096	NI	1
1	38	38	1	4,096	FCB	2
1	39	39	1	4,096	FDT	2
1	40	40	1	4,096	AC	2
1	41	42	2	4,096	UI	2
1	43	43	1	4,096	NI	2
1	44	44	1	4,096	FCB	3
1	45	45	1	4,096	FDT	3
1	46	48	3	4,096	AC	3
1	49	50	2	4,096	UI	3
1	51	62	12	4,096	NI	3
1	63	152	90	4,096	NI	9
1	153	167	15	4,096	UI	9
1	168	168	1	4,096	FCB	9
1	169	169	1	4,096	FDT	9
1	170	170	1	4,096	NI	14
1	171	172	2	4,096	UI	14
1	173	173	1	4,096	FCB	14
1	174	174	1	4,096	FDT	14
1	175	264	90	4,096	NI	11
1	265	279	15	4,096	UI	11
1	280	280	1	4,096	FCB	11
1	281	281	1	4,096	FDT	11
1	282	321	40	4,096	NI	12
1	322	341	20	4,096	UI	12
1	342	342	1	4,096	FCB	12
1	343	343	1	4,096	FDT	12
1	344	393	50	4,096	NI	13
1	394	403	10	4,096	UI	13
1	404	404	1	4,096	FCB	13
1	405	406	2	4,096	FDT	13
1	407	2,560	2,154	4,096	FREE	
2	2,561	2,561	1	32,768	DSST	

2	2,562	2,562	1	32,768	AC	9
2	2,563	2,563	1	32,768	AC	14
2	2,564	2,572	9	32,768	AC	11
2	2,573	2,581	9	32,768	AC	12
2	2,582	2,582	1	32,768	AC	13
2	2,583	2,880	298	32,768	FREE	
DATA:						
1	1	4	4	32,768	DS	1
1	5	5	1	32,768	DS	2
1	6	13	8	32,768	DS	3
1	14	45	32	32,768	DS	9
1	46	170	125	32,768	DS	14
1	171	202	32	32,768	DS	11
1	203	212	10	32,768	DS	12
1	213	222	10	32,768	DS	13
1	223	640	418	32,768	FREE	

LAYOUT provides a summary of all blocks in ASSO and DATA. The locations and lengths of sections of contiguous blocks, the block size, the type of usage and the numbers of the corresponding files are displayed. These blocks may be free (FREE) or used for the Global Blocks (CB), the File Control Block (FCB), the FCB extension (FCBE), the FCB Root Block (FCBR), the Field Definition Table (FDT), the Free Space Table (FST), the Data Space Storage Table (DSST), the Normal Index (NI), the Upper/Main Index (UI), the Address Converter (AC) or the Data Storage (DS).



Note: The first FCBR block and the first FST block are part of the global blocks. For this reason, the layout only displays FCBR and FST blocks if the database contains more than one of these blocks.

SUMMARY

SUMMARY

SUMMARY provides general information about the database and the physical layout of ASSO, DATA and WORK.

Example

```
adarep: summary
```

```
Summary of Database 76          28-NOV-2006 12:51:24
```

```

DATABASE NAME          DOKU-DATABASE
DATABASE ID            76
MAXIMUM NUMBER OF FILES 30
SYSTEM FILES          1 (CHK),    2 (SEC),    3 (USR)
ACTUAL FILES LOADED    6

```

ADAREP (Database Report)

AC SIZE						3	
CURRENT PLOG NUMBER						8	
CURRENT CLOG NUMBER						0	
Container File	Device Type	Extents from	in Blocks to	Number of Blocks	Block Size	Total Size (Megabytes)	
ASS01	file	1	1,536	1,536	2,048	3.00	
ASS02	raw	1,537	1,546	10	2,048	0.02	
DATA1	file	1	768	768	4,096	3.00	
DATA2	raw	769	868	100	3,072	0.29	
DATA3	file	869	878	10	6,144	0.06	
DATA4	file	879	888	10	6,144	0.06	
WORK1	file	1	1,365	1,365	3,072	4.00	
						10.43	=====

The device type can be "raw" (raw section), "file" (file system) or "worm" (write once, read many device. e.g. optical disk).

25 ADASCR (Security Functions)

- Functional Overview 358
- Procedure Flow 359
- Checkpoints 360
- Control Parameters 360

This chapter describes the utility "ADASCR".

Functional Overview

The security utility ADASCR creates, modifies and deletes file protection levels and user passwords, and enables the security capabilities of individual passwords. Additionally, the utility is used to display file and password security information. The output of the export functionality of ADASCR can be used to apply some of all of the security definitions of a database to another database.

Access to this utility should be strictly limited to the person or persons responsible for database security (DBA).

Multiple functions may be specified within a single run of ADASCR. There is no restriction on the number of functions which may be specified.

The affected database(s) must be online.

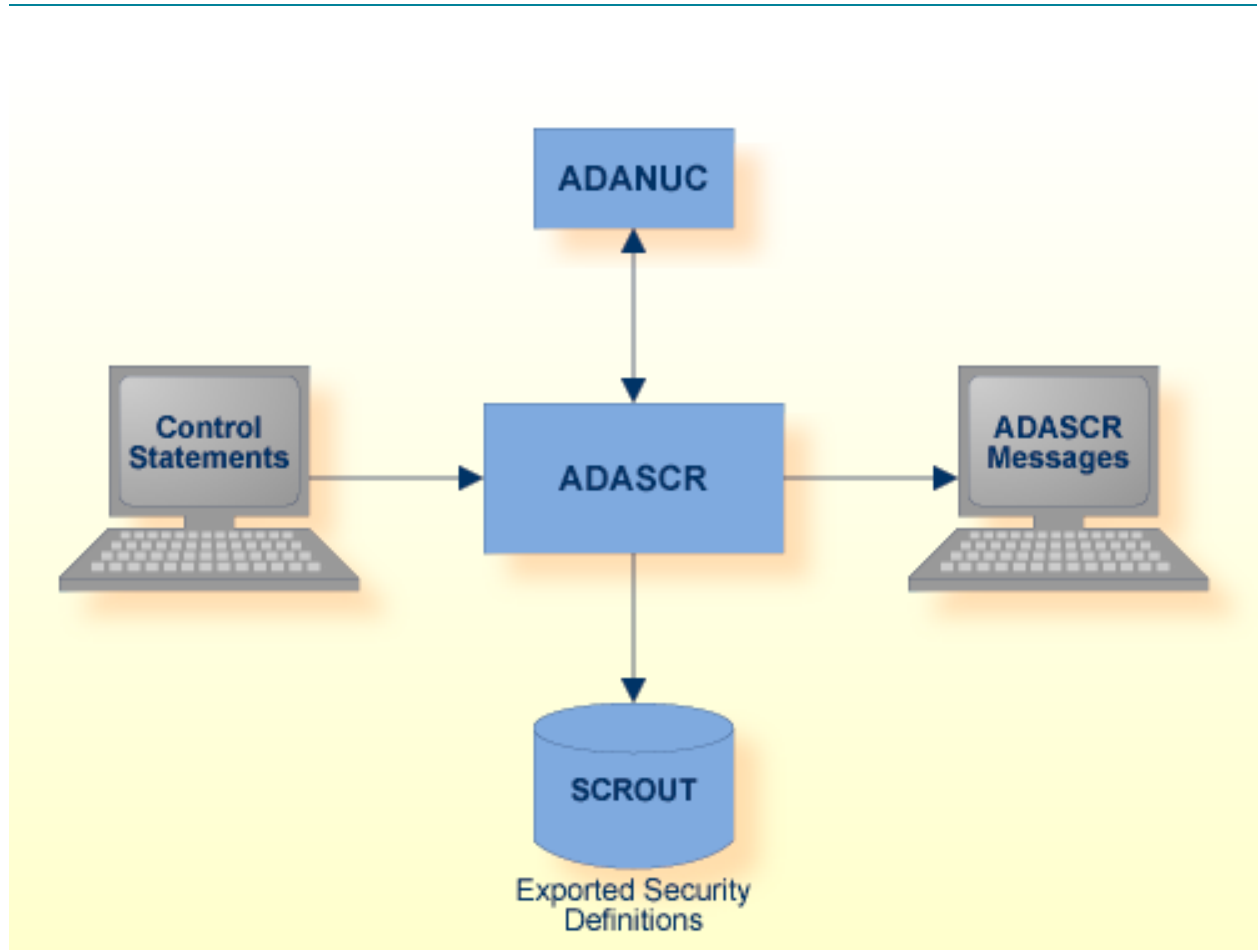
All updates resulting from ADASCR take effect immediately.

This utility is a multi-function utility.



Note: To copy existing security definitions from one hardware architecture or operating system to another you must use the EXPORT control parameter. You cannot copy the security definitions from one hardware architecture to another by using ADAULD/ADADCU and ADACMP/ADAMUP - this is because the data stored in the security file are stored in a platform-dependent internal format.

Procedure Flow



Data Set	Logical Name/ Environment variable	Storage Medium	Additional Information
Control Statements	stdin/ SYS\$INPUT		Utilities Manual
ADASCR Messages	stdout/ SYS\$OUTPUT		Messages and Codes
Exported Security Definitions	SCROUT	Disk	Utilities Manual

Checkpoints

The utility writes no checkpoints.

Control Parameters

The following control parameters are available:

```

CHANGE {=:} (string, string)

M DBID = number

DELETE = string

DISPLAY = [PASSWORDS,
           PERMISSIONS, PASSWORD = {* | string} |
           FILE = {* | (number[-number][,number[-number]]...)} |
           VALUE_CRITERIA, PASSWORD = {* | string}]

D EXPORT = {PASSWORDS,
           [TARGET_ARCHITECTURE = KEYWORD,]
           PASSWORD = {* | string} |
           PROTECTIONS,
D [TARGET_ARCHITECTURE = KEYWORD,]
           FILE = {* | (number[-number][,number[-number]]...)} |
           VALUE_CRITERIA,
D [TARGET_ARCHITECTURE = KEYWORD,]
           FILE = {* | (number[-number][,number[-number]]...)}
           PASSWORD = {* | string}}

INSERT {=:} string, FILE = (number[-number][,number[-number]]...)
        ,ACCESS = (number[,number]...)
        ,UPDATE = (number[,number]...)

PROTECT = (number[-number][,number[-number]]...)
        ,ACCESS = (number[,number]...)
        ,UPDATE = (number[,number]...)

SECURITY_BY_VALUE {=:} string, FILE = number
        ,ACCESS_CRITERION
        ,SEARCH_BUFFER = string *
        ,VALUE_BUFFER = string *
        ,UPDATE_CRITERION
        ,SEARCH_BUFFER = string *
        ,VALUE_BUFFER = string *

```


*The search/access buffer string parameters must be followed by <Newline> with no preceding comma.

CHANGE

```
CHANGE {=|:} (string, string)
```

This function changes an existing password.

The password specified by the first string must be an existing password.

The value specified by the second string must not be the same as an existing password. A password may be between 1 and 8 characters long. If less than 8 characters are specified, trailing blanks are added. The password may not contain any special characters or embedded blanks.

If you specify an equals sign, the value given for 'string' will be converted to upper case; if you specify a colon, no upper-case conversion is performed.

All entries in effect for the password specified by the first string remain in effect for the new password.

Example

```
adascr: change = (oldpw1,newpw1)
```

The password OLDPW1 is changed to NEWPW1.

DBID

```
DBID = number
```

This parameter selects the current database.



Note: The nucleus must be running.

Example

```
adascr: dbid = 155
```

The database currently being used is database 155.

DELETE

```
DELETE = string
```

This function deletes the existing password specified by the string, together with its associated permission levels and Security by Value criteria.

Example

```
adascr: delete = userpw1
```

The password USERPW1 is deleted.

DISPLAY

```
DISPLAY = [PASSWORDS |  
          PERMISSIONS, PASSWORD = {* | string} |  
          PROTECTIONS,  
          FILE = {* | (number[-number][,number[-number]]...)} |  
          VALUE_CRITERIA, PASSWORD = {* | string}]
```

This function shows current security information for files and passwords, as defined by the ADASCR utility.

Details of file protection levels, passwords, password permission levels and Security by Value criteria may be displayed.

FILE = {* | number[-number][,number[-number]]...}

The FILE parameter provides the list, range or ranges of files for which the preceding DISPLAY function is to be applied.

PASSWORDS

The PASSWORDS parameter prints a list of the passwords currently contained in the security file in ascending, alphanumeric sequence.

Example

```

adascr: display = passwords

List of defined passwords for Database 155 ("ALPHA-TS")

FORTYTWO
G6MON
USERPW1
VOYAGER

Total of 4 defined passwords

```

PERMISSIONS

The PERMISSIONS parameter prints a list of the access and update capabilities of the specified password for each currently loaded file, by comparing the password permission information against the current file protection levels.

Where respective access or update capability is granted, this is shown by the letter Y; conversely, where the capability is not granted, this is shown by the letter N.

Where access or update permissions for a given file are granted and additional Security by Value restrictions apply to that file, this is indicated by enclosing brackets, i.e. (Y).

Example

```

adascr: display = permissions, password = fortytwo

Password : "FORTYTWO"

FILE | ACCESS | UPDATE
-----|-----|-----
  1 |      N |      N
  2 |      Y |      N
  3 |      N |      N
  4 |      N |     (Y)
  6 |      Y |      Y
 10 |     (Y) |     (Y)
-----|-----|-----

( ) Further value restrictions apply where brackets shown.

```

PROTECTIONS

The PROTECTIONS parameter prints file protection information for the specified range of files, irrespective of whether or not the files specified are currently loaded.

Example:

```
display = protections, file = (4-10)
```

FILE	ACCESS	UPDATE
4	1	3
6	0	0
10	3	6

VALUE_CRITERIA

The VALUE_CRITERIA parameter prints all Security by Value criteria currently defined for the specified password.

Example:

```
display = value_criteria, password = *
```

Password : "FORTYTWO"

File	Security by Value criterion
4	ACCESS_CRITERION SEARCH_BUFFER=. VALUE_BUFFER= UPDATE_CRITERION SEARCH_BUFFER=AA,4,A,GE. VALUE_BUFFER=LENA
10	ACCESS_CRITERION SEARCH_BUFFER=CA,4,U,LT. VALUE_BUFFER=1707 UPDATE_CRITERION SEARCH_BUFFER=CA,4,S,CA,4,D,AA,0,AA. VALUE_BUFFER=01001599MS

Password : "G6MON"

File	Security by Value criterion
------	-----------------------------

```

-----
4 | ACCESS_CRITERION
   |   SEARCH_BUFFER=.
   |   VALUE_BUFFER=
   | UPDATE_CRITERION
   |   SEARCH_BUFFER=AA,3.
   |   VALUE_BUFFER=HAM
-----

```

No value criteria defined for password "USERPWD1".

No value criteria defined for password "VOYAGER".

EXPORT

```

EXPORT = {PASSWORDS,
          [TARGET_ARCHITECTURE = keyword,]
          PASSWORD = {* | string} |
          PROTECTIONS,
          [TARGET_ARCHITECTURE = keyword,]
          FILE = {* | (number[-number][,number[-number]]...)} |
          VALUE_CRITERIA,
          [TARGET_ARCHITECTURE = keyword,]
          FILE = {* | (number[-number][,number[-number]]...)},
          PASSWORD = {* | string}}

```

This function exports the current security settings (password definitions, file protection levels and security by value criteria) to the sequential file SCROUT. The output in the file SCROUT can be used as ADASCR input in order to import the security definitions into another database.

The security definitions may be exported either in mainframe syntax to import them on to mainframe platforms or in open systems syntax to import them on to open systems platforms.

The following examples for the EXPORT parameter assume that the following security definitions have already been made:

```

INSERT=MYSECRET,FILE=(9,12,13),ACCESS=(7,2,4),UPDATE=(11,2,4)

PROTECT=9,ACCESS=7,UPDATE=11
PROTECT=12,ACCESS=2,UPDATE=2
PROTECT=13,ACCESS=4,UPDATE=4

SECURITY_BY_VALUE=MYSECRET,FILE=13,ACCESS_CRITERION,SEARCH_BUFFER=B0,GE.
VALUE_BUFFER=0x01000000
UPDATE_CRITERION,SEARCH_BUFFER=B0,3,GE.
VALUE_BUFFER=0xFF0201

SECURITY_BY_VALUE=MYSECRET,FILE=12,ACCESS_CRITERION,SEARCH_BUFFER=AM,3,GE.

```

```

VALUE_BUFFER=0x33000C
UPDATE_CRITERION,SEARCH_BUFFER=AM,3,GE.
VALUE_BUFFER=0x50000C

SECURITY_BY_VALUE=MYSECRET,FILE=9,ACCESS_CRITERION,SEARCH_BUFFER=AZ,2,GE.
VALUE_BUFFER=FR
UPDATE_CRITERION,SEARCH_BUFFER=AH,8,GE,D,AH,8,LE,D,AA,8,LE. ←
VALUE_BUFFER=195201011952020160000000
    
```

PASSWORDS

```

PASSWORDS,
TARGET_ARCHITECTURE = keyword,
PASSWORD = {*|string}
    
```

The PASSWORDS parameter exports password permission levels (access and update) and the associated file or file list for the given password/passwords.

The TARGET_ARCHITECTURE parameter defines the syntax of the target platform. The following keywords can be used:

Keyword	Meaning
MAINFRAME	Export the defined password/passwords in mainframe syntax.
OPEN_SYSTEMS	Export the defined password/passwords in open systems syntax.

The default TARGET_ARCHITECTURE is OPEN_SYSTEMS.

The PASSWORD parameter specifies the password for the security settings have to be exported. It is also possible to export all defined passwords in the database - this can be done by specifying an asterisk for this parameter.

Example (export for open systems):

```

adascr: export = passwords, target_architecture = open_systems
adascr: password = mysecret
    
```

This results in the following output for SCROUT:

```

INSERT=MYSECRET,FILE=(9,12,13),ACCESS=(7,2,4),UPDATE=(11,2,4)
    
```

Example (export for mainframe):

```
adascr: export = passwords, target_architecture = mainframe, password = mysecret
```

This results in the following output for SCROUT:

```
ADASCR INSERT PW=MYSECRET,FILE=9,12,13,ACC=7,2,4,UPD=11,2,4
```

PROTECTIONS

```
PROTECTIONS,  
TARGET_ARCHITECTURE = keyword,  
FILE = {*|(number[-number][,number[-number]]...)}  
}
```

The PROTECTIONS parameter exports the protection levels of the given file, file range or ranges.

The TARGET_ARCHITECTURE parameter defines the syntax of the target platform. The following keywords can be used:

Keyword	Meaning
MAINFRAME	Export the defined password/passwords in mainframe syntax.
OPEN_SYSTEMS	Export the defined password/passwords in open systems syntax.

The default TARGET_ARCHITECTURE is OPEN_SYSTEMS.

The FILE parameter specifies the files for which protection levels are to be exported.

Example (export for open systems):

```
adascr: export = protections, target_architecture = open_systems, file = (9,12,13)
```

This results in the following output for SCROUT:

```
PROTECT=(9,12,13),ACCESS=(7,2,4),UPDATE=(11,2,4)
```

Example (export for mainframe):

```
adascr: export = protections, target_architecture = mainframe, file = (9,12,13)
```

This results in the following output for SCROUT:

ADASCR PROTECT FILE=9,12,13,ACC=7,2,4,UPD=11,2,4

VALUE_CRITERIA

```
VALUE_CRITERIA,
TARGET_ARCHITECTURE = (keyword [,keyword]),
FILE = {*|(number[-number][,number[-number]]...)},
PASSWORD = {*|string}
```

The VALUE_CRITERIA parameter exports defined security-by-value settings for a specific file, for the password specified by 'string'.

The TARGET_ARCHITECTURE parameter defines the syntax and also the byte order of the target platform. The following keywords can be used:

Keyword Group	Valid Keywords
Syntax	MAINFRAME OPEN_SYSTEMS
Byte Order	HIGH_ORDER_BYTE_FIRST LOW_ORDER_BYTE_FIRST

The default TARGET_ARCHITECTURE is OPEN_SYSTEMS.

The default byte order corresponds to the architecture of the machine on which ADASCR is running.



Note: If you export security-by-value definitions from open systems to mainframe platforms, a warning will be issued if the search buffer contains W-formatted fields. This is because W-formatted fields are not supported on mainframe platforms in security-by-value definitions. Any search buffer that contains W-formatted fields will not be exported.

The FILE parameter specifies the file list or range(s) of files for which security-by-value definitions are to be exported. Multiple specifications of the same file number are not permitted.

The PASSWORD parameter specifies the password for which security-by-value definitions are to be exported. If you specify an asterisk, the export will be for all defined passwords.

Example (export for open systems):

```
adascr: export = value_criteria
adascr: target_architecture = (open_systems, low_order_byte_first)
adascr: file = 9
adascr: password = mysecret
```

This results in the following output for SCROUT:


```
SECURITY_BY_VALUE=MYSECRET,FILE=9,ACCESS_CRITERION,SEARCH_BUFFER=AH,8,GE.
VALUE_BUFFER=19520130
UPDATE_CRITERION,SEARCH_BUFFER=AH,8,GE,D,AH,8,LE,D,AA,8,LE.
VALUE_BUFFER=195201011952020160000000
```

Example (export for mainframe):

```
adascr: export = value_criteria
adascr: target_architecture = mainframe
adascr: file = 9
adascr: password = mysecret
```

This results in the following output for SCROUT:

```
ADASCR SBYVALUE PW=MYSECRET,FILE=9,
CRITACC='A=AH,GE.',
VALACC='19520130',
CRITUPD='A=AH,GE,D,AH,LE,D,AA,LE.',
VALUPD='19520101,19520201,A(60000000)'
```

Example

This example shows how to export of existing security settings of a Windows Adabas database 33 in order to subsequently import them into a Unix Adabas database 34.

The export function of ADASCR is used as follows:

```
set SCROUT=scrout.txt
adascr
adascr: db=33
adascr: export=passwords,password=MYSECRET
adascr: export=protections,file=(9,12,13)
adascr: export=VALUE_CRITERIA,target_architecture=(open_systems,
high_order_byte_first),file=(9,12,13),password=MYSECRET
```

The file “scrout.txt” can now be edited as required, for example, to secure more files other than the ones exported with the given password.

Now copy the exported text file “scrout.txt” to the desired target platform, in this case to a Unix platform. Import the exported security settings using ADASCR with the following statement:

```
adascr db=34 + < scrout.txt

%ADASCR-I-DBON, database 34 accessed online
%ADASCR-I-PWINS, password "MYSECRET" inserted
%ADASCR-I-FILPRO, protections (access 7, update 11) set for file 9
%ADASCR-I-FILPRO, protections (access 2, update 2) set for file 12
%ADASCR-I-FILPRO, protections (access 4, update 4) set for file 13
%ADASCR-I-SEVINS, Value criteria for file 13 added to password "MYSECRET"
```

```
%ADASCR-I-SEVINS, Value criteria for file 12 added to password "MYSECRET"  
%ADASCR-I-SEVINS, Value criteria for file 9 added to password "MYSECRET"
```

You can check the imported security settings with the DISPLAY control parameter of ADASCR:

```
adascr  
adascr: dbid=34 display=passwords  
  
List of defined passwords for Database 34 ("GENERAL_DATABASE")  
  
MYSECRET  
  
Total of 1 defined password  
  
adascr: dbid=34 display=permissions,password=MYSECRET  
  
password : "MYSECRET"  
  
FILE | ACCESS | UPDATE  
-----  
  1 |      Y |      Y  
  2 |      Y |      Y  
  3 |      Y |      Y  
  9 |     (Y) |    (Y)  
 11 |      N |      N  
 12 |     (Y) |    (Y)  
 13 |     (Y) |    (Y)  
 14 |      N |      N  
-----  
  
( ) Further value restrictions apply where brackets shown.  
  
Adascr: db=34 display=protections,file=(9,12,13)  
  
FILE | ACCESS | UPDATE  
-----  
  9 |      7 |     11  
 12 |      2 |      2  
 13 |      4 |      4  
-----  
  
adascr: db=34 display=value_criteria,password=MYSECRET  
  
password : "MYSECRET"  
  
| File | Security by Value criterion  
|-----+-----  
|  9 | ACCESS_CRITERION
```

	SEARCH_BUFFER: "AZ,2,GE."
	VALUE_BUFFER: "FR"
	UPDATE_CRITERION
	SEARCH_BUFFER: "AH,8,GE,D,AH,8,LE,D,AA,8,LE. "
	VALUE_BUFFER: "195201011952020160000000"
12	ACCESS_CRITERION
	SEARCH_BUFFER: "AM,3,GE."
	VALUE_BUFFER: 0x33000C
	UPDATE_CRITERION
	SEARCH_BUFFER: "AM,3,GE."
	VALUE_BUFFER: 0x50000C
13	ACCESS_CRITERION
	SEARCH_BUFFER: "BO,GE."
	VALUE_BUFFER: 0x01000000
	UPDATE_CRITERION
	SEARCH_BUFFER: "BO,3,GE."
	VALUE_BUFFER: 0xFF0201

INSERT

```
INSERT {=|:} string, FILE = (number[-number][,number[-number]]...)
      ,ACCESS = (number[,number]...)
      ,UPDATE = (number[,number]...)
```

This function inserts a password specified by 'string' into the password table.

If you specify an equals sign, the value given for 'string' will be converted to upper case; if you specify a colon, no upper-case conversion is performed.

The password may be between 1 and 8 characters long. If less than 8 characters are specified, trailing blanks are added. The password may not contain any special characters or embedded blanks.

ACCESS = (number[,number]...)

The ACCESS parameter specifies the access protection levels to be associated with the files or file ranges specified in the FILE parameter. Each protection level corresponds to one file or one file range. A value may be specified in the range 0 - 14. The protection levels must be specified in the same order as the corresponding files or file ranges in the FILE parameter.

FILE = (number[-number[,number[-number]]...)

The FILE parameter specifies the file list or range(s) of files for which permission levels are being provided. Multiple specifications of the same file number are not permitted.

UPDATE = (number[,number]...)

The UPDATE parameter specifies the update protection levels to be associated with the files or file ranges specified in the FILE parameter. Each protection level corresponds to one file or one file range. A value may be specified in the range 0 - 14. The protection levels must be specified in the same order as the corresponding files or file ranges in the FILE parameter.

Example

```
adascr: insert = userpwx, file = (1,2,3),
          access = (7,7,7), update = (0,8,8)

adascr: insert = userpwy, file = (1,2,3),
          access = (7,2,2), update = (8,0,0)

adascr: insert = userpwz, file = (1-3,5),
          access = (2,3), update = (4,6)
```

PROTECT

```
PROTECT = (number[-number][,number[-number]]...)
          ,ACCESS = (number[,number]...)
          ,UPDATE = (number[,number]...)
```

This function inserts (or updates) the access and/or update protection levels for the file, list or range(s) specified. Multiple specifications of the same file number are not permitted.

ACCESS = (number[,number]...)

The ACCESS parameter specifies the access protection levels to be associated with the files or file ranges specified in the PROTECT parameter. Each protection level corresponds to one file or one file range. A value may be specified in the range 0 - 15. The protection levels must be specified in the same order as the corresponding files or file ranges in the PROTECT parameter.

Note that the maximum protection level for access for the PROTECT function is 15, whereas the INSERT function allows a maximum of only 14. Therefore, with ACCESS=15, a file can be protected to prevent any user from accessing it.

UPDATE = (number[,number]...)

The UPDATE parameter specifies the update protection levels to be associated with the files or file ranges specified in the PROTECT parameter. Each protection level corresponds to one file or one file range. A value may be specified in the range 0 - 15. The protection levels must be specified in the same order as the corresponding files or file ranges in the PROTECT parameter.

Note that the maximum protection level for update for the INSERT function is 14, whereas the PROTECT function allows a maximum of 15. Therefore, with UPDATE=15, a file can be protected to prevent any user from updating it.

Example

```
adascr: protect = 25, access = 7, update = 11
adascr: protect = (1,2,3), access = (7,7,7), update = (8,8,8)
adascr: protect = (4-6), access = 10, update = 12
```

If the file number of a security-protected file is subsequently changed as a result of running the RENUMBER function of the ADADBM utility, the PROTECT function has to be reexecuted in order to reestablish the security protection levels for the file. The passwords also have to be reestablished, since they reflect the old file number.

SECURITY_BY_VALUE

```
SECURITY_BY_VALUE {=|:} string, FILE = number
                    ,ACCESS_CRITERION
                    ,SEARCH_BUFFER = string *
                    ,VALUE_BUFFER = string *
                    ,UPDATE_CRITERION
                    ,SEARCH_BUFFER = string *
                    ,VALUE_BUFFER = string *
```

* The search/access buffer string parameters must be followed by <Newline> with no preceding comma.

This function inserts (or updates) the Security by Value criteria for a specific file, for the given password specified by 'string'. The password must already have been inserted into the security file, using the INSERT function. Each password can have Security by Value criteria defined for a maximum of 99 files.

If you specify an equals sign, the value given for 'string' will be converted to upper case; if you specify a colon, no upper-case conversion is performed.

ACCESS_CRITERION

The ACCESS_CRITERION keyword must precede the search buffer and value buffer which will define the criterion for restricting access to data using the provided password.

In order for the access value criterion to be specified, the ACCESS password permission level for the file must also have previously been set (i.e. non-zero); in the case where the level is not set, the ACCESS_CRITERION keyword cannot be specified.

FILE = number

The FILE parameter specifies the file for which the value criteria are to be defined. Exactly one file must be specified and the file must be currently loaded in the database.

UPDATE_CRITERION

The UPDATE_CRITERION keyword must precede the search buffer and value buffer which will define the criterion for restricting the update of data using the provided password.

In order for the update value criterion to be specified, the UPDATE password permission level for the file must also have previously been set (i.e. non-zero); in the case where the level is not set, the UPDATE_CRITERION keyword cannot be specified.

SEARCH_BUFFER = string

The SEARCH_BUFFER parameter is used to provide the search expressions for the access/update criterion. Syntax and examples of search buffer construction are provided in *Command Reference, Calling Adabas, Search and Value Buffers*.

Certain restrictions apply to the search buffer when used for defining Security by Value criteria; soft coupling and sub-, super-, hyper- and phonetic descriptors are not supported.

If the required criterion is that no restrictions should apply, then the associated search buffer should be specified containing only the terminator, i.e.:

```
SEARCH_BUFFER = .
```

In this case, the VALUE_BUFFER parameter is not required and does not need to be supplied.

VALUE_BUFFER = string

The VALUE_BUFFER parameter is used to provide the corresponding values for the search expressions for the access/update value criterion, specified in the preceding search buffer. The string may be specified either directly, as an alphanumeric string or as a string using hexadecimal notation.

Example

```
adascr: security_by_value = fortytwo, file = 10,  
        access_criterion, search_buffer = CA,4,U,LT.  
adascr: value_buffer = 1707  
adascr: update_criterion, search_buffer = A,4,S,CA,4,D,AA,0,AA.  
adascr: value_buffer = 01001599MS
```

```
adascr: security_by_value = g6mon, file = 3  
adascr: access_criterion, search_buffer = .  
adascr: update_criterion, search_buffer = AC,3.  
adascr: value_buffer = HAM
```

If either the access or update protection level for the specified file is zero, the associated value criterion will not be tested when the password is used for accessing/updating records for that file.

26 ADATST (Issuing Adabas Commands)

- Functional Overview 378
- Procedure Flow 379
- Checkpoints 379
- Control Parameters 380

This chapter describes the utility "ADATST".

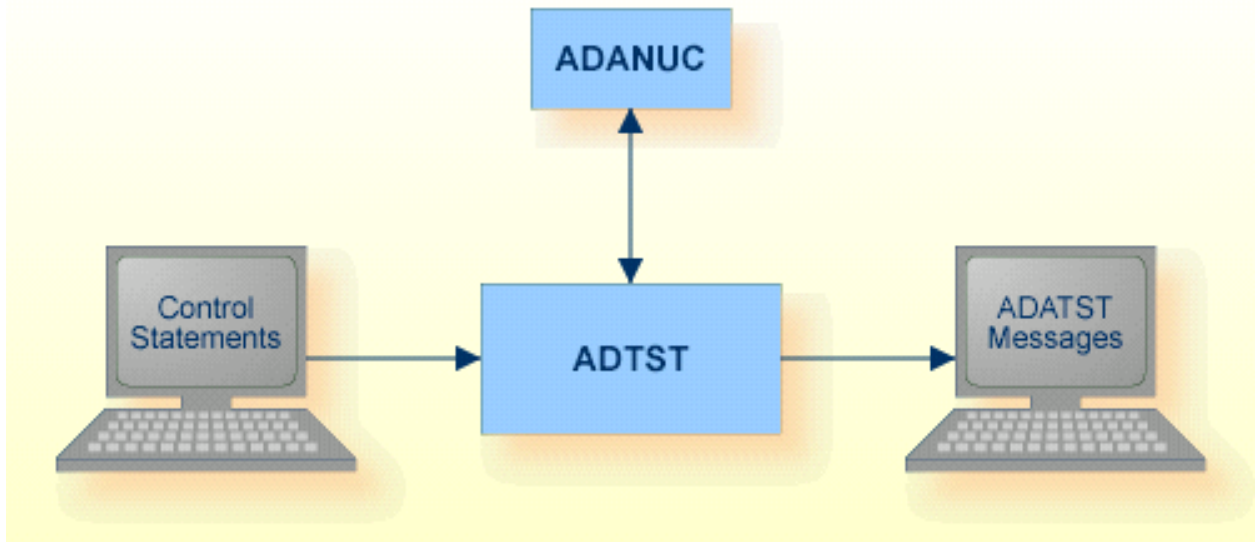
Functional Overview

The ADATST utility is used to fill the control block and the necessary buffers in order to issue Adabas commands.

Both the old ACB Adabas command interface and the new ACBX command interface are supported. For more information on the command interfaces please refer to the *Command Reference* documentation. Note that the ACB interface can be considered as a subset of the ACBX interface: Fields in the old ACB Adabas control block are also contained in the new ACBX control block (or the ABDs – the Adabas buffer descriptions) with the difference that some ACB fields are smaller than the corresponding fields in the new interface. In particular the buffer lengths have been increased that now Adabas buffers greater than 64 KB are possible. Switching between the interfaces is possible, but when you switch back to the old ACB interface you must be aware of the restrictions of the old ACB interface.

This utility is a multi-function utility.

Procedure Flow



Data Set	Environment Variable/ Logical Name	Storage Medium	Additional Information
Control statements	stdin/ SYS\$INPUT		Utilities Manual
ADATST messages	stdout/ SYS\$OUTPUT		Messages and Codes

Checkpoints

The utility writes no checkpoints.

Control Parameters



Note: In the following, "string" is an ASCII string or 0x followed by hexadecimal data.

The following control parameters are available:

```
A1 {=|:} string
A2 {=|:} string
A3 {=|:} string
A4 {=|:} string
A5 {=|:} string
A6 {=|:} string
ABD
ALOOP [= number]
CB
CBDUMP
CC = string
CID = string
C01 = string
C02 = string
C03 = string
C04 = string
C05 = string
C06 = string
C07 = string
C08 = string
M DBID = number
```

```
DLOOP  
  
ELOOP  
  
ERRORS = number  
  
EXECUTE = { number | ISNQ }  
  
FB [={|:} string]  
  
FB2 [={|:} string]  
  
FB3 [={|:} string]  
  
FBL = number  
  
FB2L [= number]  
  
FB3L [= number]  
  
FILE = number  
  
GO [= { number | ISNQ } ]  
  
IB [= (number [,number]...)]  
  
IBL = number  
  
D INTERFACE = keyword  
  
ISN = number  
  
ISND = number  
  
ISNI = number  
  
ISNL = { number | ISN }  
  
ISNQ = number  
  
LOOP  
  
MB = (number_buffers, number_isns)  
  
D [NO]OUTPUT  
  
OVERWRITE_RB = string  
  
OVERWRITE_RB2 = string  
  
OVERWRITE_RB3 = string
```

```
RB [={|:} string]
RB2 [={|:} string]
RB3 [={|:} string]
RBL = number
RB2L [= number]
RB3L [= number]
READ_RB = string
READ_RB2 = string
READ_RB3 = string
RESPONSE = number
SB [={|:} string]
SBL = number
TIME
D [NO]TRACE
  VB [={|:} string]
  VBL = number
D WAIT [= [time]]
  WRITE_RB = string
  WRITE_RB2 = string
  WRITE_RB3 = string
```

A1

```
A1 {=|:} string
```

This parameter sets the Additions 1 field.

If you specify an equals sign, the value given for 'string' will be converted to upper case; if you specify a colon, no upper-case conversion is performed.

A2

```
A2 {=|:} string
```

This parameter sets the Additions 2 field.

If you specify an equals sign, the value given for 'string' will be converted to upper case; if you specify a colon, no upper-case conversion is performed.

A3

```
A3 {=|:} string
```

This parameter sets the Additions 3 field.

If you specify an equals sign, the value given for 'string' will be converted to upper case; if you specify a colon, no upper-case conversion is performed.

A4

```
A4 {=|:} string
```

This parameter sets the Additions 4 field.

If you specify an equals sign, the value given for 'string' will be converted to upper case; if you specify a colon, no upper-case conversion is performed.

A5

```
A5 {=|:} string
```

This parameter sets the Additions 5 field.

If you specify an equals sign, the value given for 'string' will be converted to upper case; if you specify a colon, no upper-case conversion is performed.

A6

```
A6 {=|:} string
```

This parameter sets the Additions 6 field.

If you specify an equals sign, the value given for 'string' will be converted to upper case; if you specify a colon, no upper-case conversion is performed.

ABD

ABD

This parameter is available only after specifying INTERFACE=ACBX. It displays the Adabas buffer definitions for the Adabas buffers that are currently defined.

ALOOP

ALOOP [= number]

This parameter opens a loop to add more lines. 'number' is a line number. If a line number is specified, new lines are added from that point, overwriting existing lines; if no number is specified, new lines are added at the end. Close the loop with ELOOP.

CB

CB

This function displays the contents of the control block.

If the parameter INTERFACE=ACB is specified, only the fields contained in the old ACB Adabas control block are displayed.

If the parameter INTERFACE=ACBX is specified, the fields contained in the new ACBX Adabas control block but not in the old ACB Adabas control block are also displayed.

CBDUMP

CBDUMP

This function dumps the control block in hex.

CC

CC = string

This parameter specifies the command code.

CID

```
CID= string
```

This parameter specifies the command ID.

C01

```
C01 = string
```

This parameter sets the command option 1.

C02

```
C02 = string
```

This parameter sets the command option 2.

C03

```
C03 = string
```

This parameter sets the command option 3.

C04

```
C04 = string
```

This parameter sets the command option 4.

C05

```
C05 = string
```

This parameter sets the command option 5.

C06

```
C06 = string
```

This parameter sets the command option 6.

C07

```
C07 = string
```

This parameter sets the command option 7.

C08

```
C08 = string
```

This parameter sets the command option 8.

DBID

```
DBID = number
```

This parameter specifies the database to be used.

DLOOP

```
DLOOP
```

This function displays a saved command loop.

ELOOP

```
ELOOP
```

This function terminates a loop.

ERRORS

```
ERRORS = number
```

This parameter specifies the number of errors permitted before termination occurs.

EXECUTE

```
EXECUTE = { number | ISNQ }
```

This parameter executes a loop `n` times, where `n` is specified by `number` or by ISNQ. Enter CTRL/C to terminate a loop.

FB

```
FB [{=|:} string]
```

This parameter is used to display the first format buffer or enter data into the format buffer. The length is set implicitly.

FB2

```
FB2 [{=|:} string]
```

This parameter is used to display the second format buffer or enter data into the format buffer. The length is set implicitly.

FB3

```
FB3 [{=|:} string]
```

This parameter is used to display the third format buffer or enter data into the format buffer. The length is set implicitly.

FBL

```
FBL = number
```

This parameter defines the first format buffer length in the control block.

FB2L

```
FB2L = number
```

This parameter defines the second format buffer length in the control block.

FB3L

```
FBL3 = number
```

This parameter defines the third format buffer length in the control block.

FILE

```
FILE = number
```

This parameter specifies the file number.

GO

```
GO [= { number | ISNQ } ]
```

This function calls Adabas once or `n' times, where `n' is specified by `number' or ISNQ. Enter CTRL/C to terminate a loop.

IB

```
IB [= (number [,number]...)]
```

This parameter is used to display the ISN buffer or enter ISNs. The length is set implicitly.

IBL

```
IBL = number
```

This parameter specifies the ISN buffer length in the control block.

INTERFACE

```
INTERFACE = keyword
```

This parameter is used to switch between the old and new Adabas command interface. Valid keywords are ACB and ACBX. The default is ACB. After INTERFACE = ACB has been specified, Adabas calls are performed with the old ACB Adabas interface - any fields that are only contained in the new ACBX Adabas control block and additional format buffers and record buffers are ignored. You can switch between the old and the new Adabas interface in one Adabas session.

ISN

```
ISN = number
```

This parameter sets ISN with the number supplied.

ISND

```
ISND = number
```

This parameter subtracts `number' from ISN.

ISNI

```
ISNI = number
```

This parameter adds `number' to ISN.

ISNL

```
ISNL = { number | ISN }
```

This parameter is used to set the ISN lower limit with the number supplied, or to move the ISN from the control block into the ISN lower limit.

ISNQ

```
ISNQ = number
```

This parameter specifies the ISN quantity with the number supplied.

LOOP

```
LOOP
```

This function defines the start of a loop. All commands that follow will be saved until an ELOOP is entered.

MB

```
MB = (number_buffers, number_isns)
```

This parameter defines the number of multifetch buffers and the number of IS entries that can be stored in a multifetch buffer. `number_buffers` may be a number between 0 and 3. `number_isns` must be a number >0.

The MB parameter can only be specified after `INTERFACE = ACBX` has been specified.. Since multifetch buffers are pure output buffers, it is not possible to enter content into the multifetch buffers.

Once multifetch buffers have been specified, you can display their contents with the RB, RB 2 and RB3 parameters.

[NO]OUTPUT

```
[NO]OUTPUT
```

If this option is set to NOOUTPUT, no messages are output when calling Adabas `n' times. Only error messages will be printed.

The default is OUTPUT.

OVERWRITE_RB

```
OVERWRITE_RB = string
```

This parameter specifies the name of an existing file to which the contents of the first record buffer are written. The current contents of the file will be overwritten.

OVERWRITE_RB2

```
OVERWRITE_RB2 = string
```

This parameter specifies the name of an existing file to which the contents of the second record buffer are written. The current contents of the file will be overwritten.

OVERWRITE_RB3

```
OVERWRITE_RB3 = string
```

This parameter specifies the name of an existing file to which the contents of the third record buffer are written. The current contents of the file will be overwritten.

RB

```
RB [{=|:} string]
```

This parameter is used to display the first record buffer or enter data into the first record buffer. For input to a file, the length is set implicitly.

If you specify an equals sign, the value given for 'string' will be converted to upper case; if you specify a colon, no upper-case conversion is performed.

If you display the first record buffer, and at least one multifetch buffer has been defined with the MB parameter, the first multifetch buffer is also displayed.

RB2

```
RB2 [{=|:} string]
```

This parameter is used to display the second record buffer or enter data into the second record buffer. For input to a file, the length is set implicitly.

If you specify an equals sign, the value given for 'string' will be converted to upper case; if you specify a colon, no upper-case conversion is performed.

If you display the second record buffer, and at least two multifetch buffers have been defined with the MB parameter, the second multifetch buffer is also displayed.

RB3

```
RB3 [{=|:} string]
```

This parameter is used to display the third record buffer or enter data into the third record buffer. For input to a file, the length is set implicitly.

If you specify an equals sign, the value given for 'string' will be converted to upper case; if you specify a colon, no upper-case conversion is performed.

If you display the third record buffer, and at three multifetch buffer have been defined with the MB parameter, the third multifetch buffer is also displayed.

RBL

```
RBL = number
```

This parameter specifies the first record buffer length in the control block.

RB2L

```
RB2L = number
```

This parameter specifies the second record buffer length in the control block.

RB3L

```
RB3L = number
```

This parameter specifies the third record buffer length in the control block.

READ_RB

```
READ_RB = string
```

This parameter specifies the name of a file that is read into the first record buffer.

READ_RB2

```
READ_RB2 = string
```

This parameter specifies the name of a file that is read into the second record buffer.

READ_RB3

```
READ_RB3 = string
```

This parameter specifies the name of a file that is read into the third record buffer.

RESPONSE

```
RESPONSE = number
```

This parameter displays the error text for the given nucleus response code.

SB

```
SB [=|:] string
```

This parameter is used to display the search buffer or enter data into the search buffer. The length is set implicitly.

If you specify an equals sign, the value given for 'string' will be converted to upper case; if you specify a colon, no upper-case conversion is performed.

SBL

```
SBL = number
```

This parameter specifies the search buffer length in the control block.

TIME

```
TIME
```

This function marks the current time, and displays the difference between this and the last time mark.

[NO]TRACE

```
[NO]TRACE
```

This option traces execute loops.

The default is NOTRACE.

VB

```
VB [{=|:} string]
```

This parameter is used to display the value buffer or enter data into the value buffer. The length is set implicitly. If 'string' equals RB, the record buffer is moved into the value buffer.

If you specify an equals sign, the value given for 'string' will be converted to upper case; if you specify a colon, no upper-case conversion is performed.

VBL

```
VBL = number
```

This parameter specifies the value buffer length in the control block.

WAIT

```
WAIT [= seconds]
```

This parameter causes ADATST to wait for a given period. The waiting time is entered in seconds. Once the time is set, you can wait for the same period by entering 'WAIT' without any additions.

The default time is 10 seconds.

Example

```
adatst: wait = 15
```

ADATST waits for fifteen seconds.

WRITE_RB

```
WRITE_RB = string
```

This parameter specifies the name of a file to which the contents of the first record buffer are written. The record buffer is only written if a file with the specified name does not already exist.

WRITE_RB2

```
WRITE_RB2 = string
```

This parameter specifies the name of a file to which the contents of the second record buffer are written. The record buffer is only written if a file with the specified name does not already exist.

WRITE_RB3

```
WRITE_RB3 = string
```

This parameter specifies the name of a file to which the contents of the third record buffer are written. The record buffer is only written if a file with the specified name does not already exist.

27 ADAULD (File Unloading)

- Functional Overview 396
- Procedure Flow 398
- Checkpoints 400
- Control Parameters 401
- Examples 407
- TEMP Data Set Space Estimation 408
- Restart Considerations 408

This chapter describes the utility "ADAULD".

Functional Overview

The utility ADAULD unloads an Adabas file, i.e. records are retrieved from a database or an Adabas backup copy, and written to a sequential file.

The main reasons for unloading a file are:

- To change the space allocation, to reduce the number of logical extents assigned to the index, Address Converter or Data Storage, and/or to change the padding factor. In this case, the file has to be unloaded, deleted and reloaded. These features are also available with ADAORD;
- To create one or more test files, all containing the same data. This procedure requires a file to be unloaded and then reloaded using a different file number. This feature is also available with ADAORD;
- To extract data from a file for subsequent input to ADAMUP. This is useful for moving records from a production database to an archive database;
- To re-establish a file that has been archived on an Adabas backup copy.

When unloading a file from a database, the records may be unloaded in:

Logical sequence

The records are unloaded in an ascending sequence based on the values of a user-specified descriptor;

ISN sequence

The records are unloaded in ascending ISN sequence;

Physical sequence

The records are unloaded in the order in which they are physically located in Data Storage.

Unloading in logical or ISN sequence requires the nucleus to be active. The nucleus is not required when unloading in physical sequence, provided ADAULD has access to the database container files.

When unloading from an Adabas backup copy, the records are unloaded in the sequence in which they were stored by ADABCK. This is generally in ascending data RABN sequence. However, this sequence cannot be guaranteed when the DRIVES option was used or when the dump was made online (please refer to the DRIVES option of the utility ADABCK for more detailed information).

The unloaded records are output in compressed format and are identical to the records produced by the compression utility ADACMP. Since each data record is preceded by its ISN, these ISNs can be used as user ISNs when reloading the file (please refer to the USERISN option of the utility ADAMUP for more detailed information).

The user can specify that the descriptor values required to recreate the index for the file are omitted during the UNLOAD process (SHORT option). This reduces the unload processing time. This option must not be used if the output is intended as direct input for ADAMUP.

After completion, ADAULD returns one of the following exit status values:

0

Records have been successfully unloaded, and no database corruption was detected.

12

The unload was successful, but corrupted data records were detected, which were not unloaded. It is recommended that you run ADAVFY in order to obtain more information about the database corruptions.

15

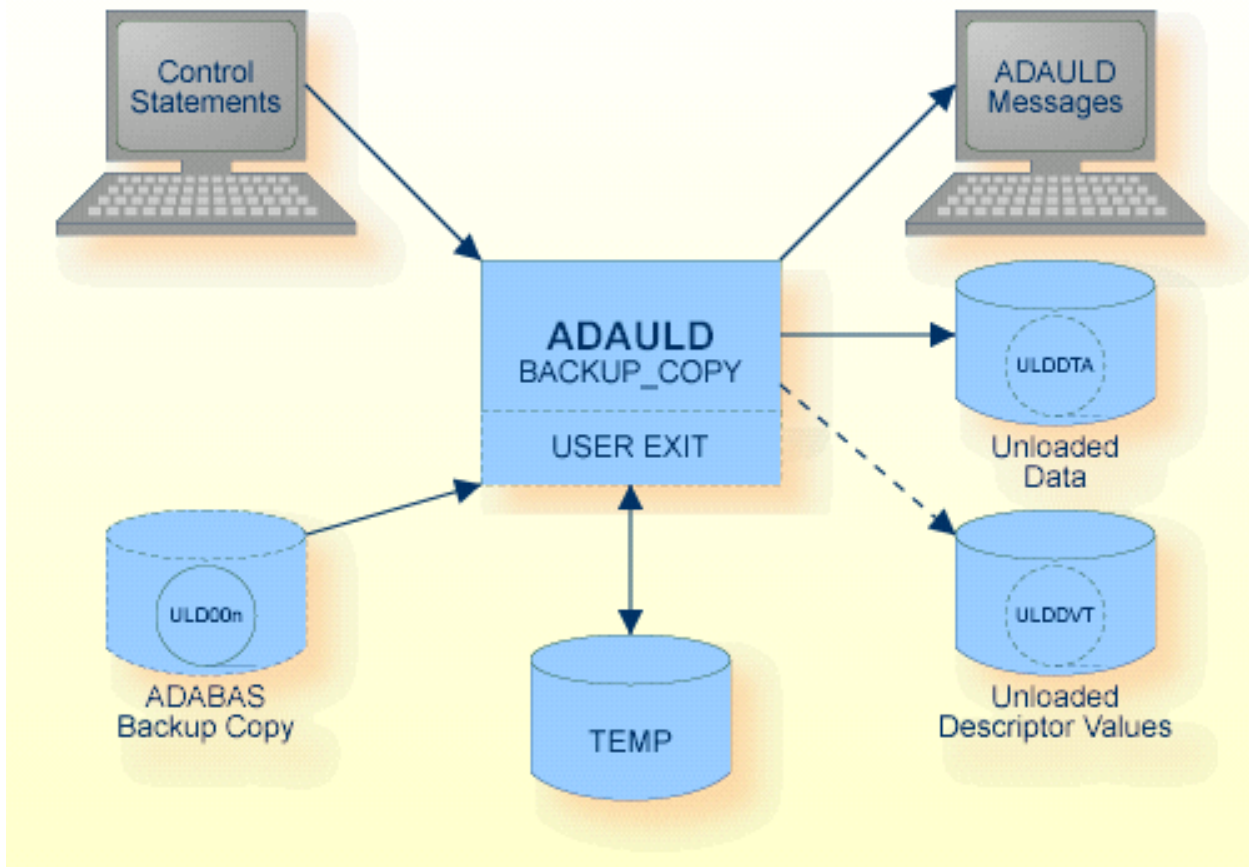
The unload was successful, but no records were unloaded. In scripts, you can check for this status value if further activities are required only after unloading at least one record.

255

Unload was not successful.

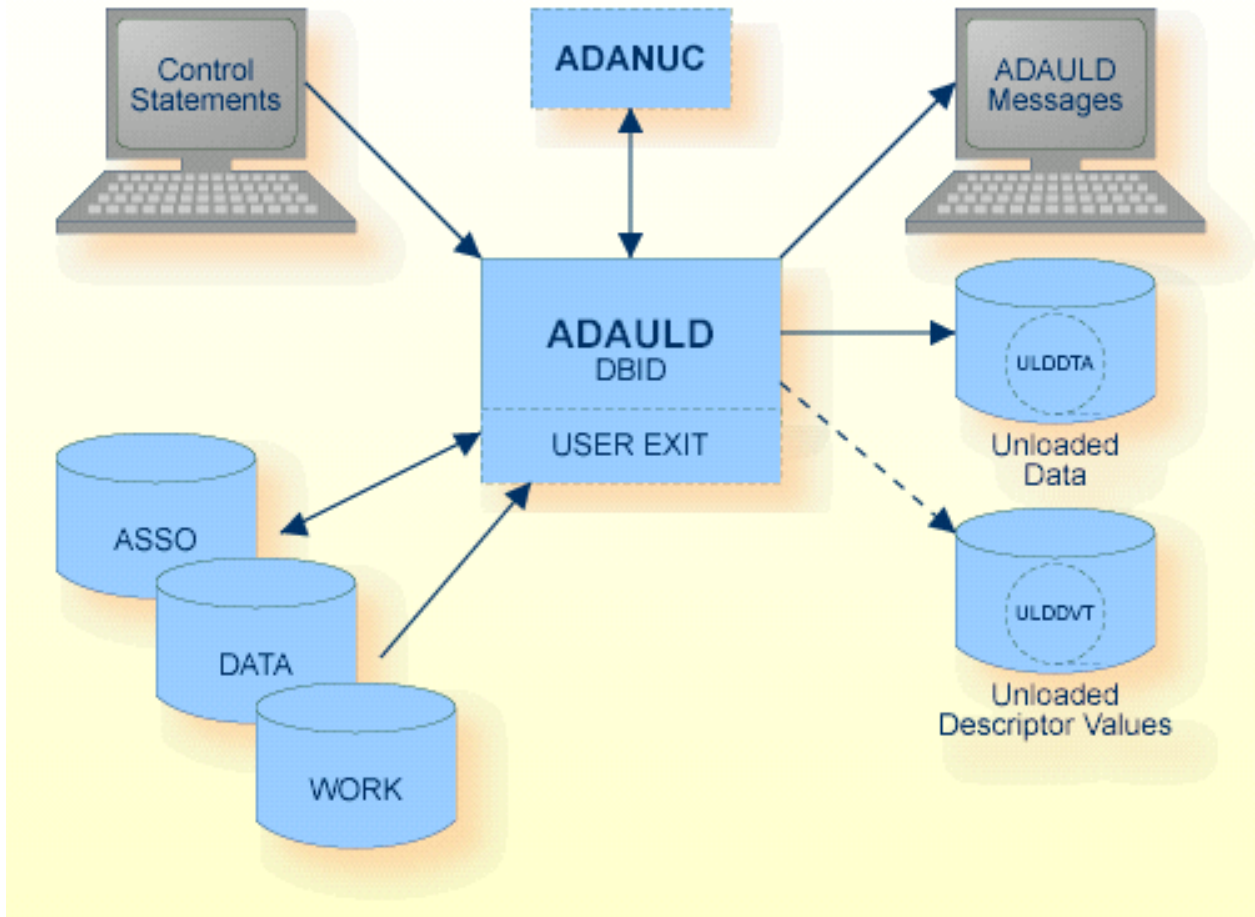
This utility is a single-function utility.

Procedure Flow



BACKUP_COPY Function

The sequential files ULD00n, ULDDTA, ULDDVT can have multiple extents. For detailed information about files with multiple extents, see *Administration, Using Utilities*.



DBID Function

The sequential files ULDDTA, ULDDVT can have multiple extents. For detailed information about files with multiple extents, see Administration, Using Utilities.

Data Set	Environment Variable/ Logical Name	Storage Medium	Additional Information
Associator	ASSOx	Disk, Tape	see note 2
Data storage	DATAx	Disk, Tape	see note 2
Backup copy	ULD00n	Disk, Tape	Output of ADABCK's DUMP function, input for ADAULD
Unloaded data	ULDDTA	Disk, Tape (see note 1)	
Unloaded descriptor values	ULDDVT	Disk, Tape (see note 1)	
Control statements	stdin/ SYS\$INPUT		Utilities Manual

Data Set	Environment Variable/ Logical Name	Storage Medium	Additional Information
ADAULD messages	stdout/ SYS\$OUTPUT		Messages and Codes
Temporary storage	TEMPx	Disk, Tape	see note 3 and 4
Work storage	WORK1	Disk, Tape	see note 2



Notes:

1. A named pipe can be used for this sequential file (not on OpenVMS) see *Administration, Using Utilities* for details).
2. Required by offline unload. Will also increase the speed of online unload using physical sequence.
3. Only required if unloading from a backup copy with the online option being used. If the utility is executed offline, WORK may be used as TEMP if there is no Autorestart pending, by setting the environment variable TEMP1 to the same value as WORK1.
4. The ADAULD BACKUP_COPY function does not read the DBxxx.INI file to find TEMP, therefore you must specify TEMP via environment variables.

Checkpoints

The following table shows the nucleus requirements for each function and the checkpoint written:

Function	Nucleus must be active	Nucleus must NOT be active	Nucleus is NOT required	Checkpoint written
BACKUP_COPY			X	-
DBID	X(see note 1)	X(see note 3)	X(see note2)	SYNX



Notes:

1. When unloading in logical sequence or ISN sequence or when the database container file cannot be accessed by ADAULD (e.g. when unloading from a remote node). Also applies if a file contains LOB data, because LOB data must be unloaded in logical sequence. Also applies if the search buffer and value buffer are provided.
2. When unloading in physical sequence and ADAULD has access to the database container files.
3. When unloading an Adabas system file.

Control Parameters

The following control parameters are available:

```

    BACKUP_COPY = number, FILE = number
                  [,FDT]
                  [,NUMREC = number]
D                [, [NO]ONLINE]
D                [, [NO]SHORT | [NO]SINGLE_FILE ]
                  [,SKIPREC = number]
D                [, [NO]USEREXIT]

    DBID = number , FILE = number
          [,FDT]
D        [, [NO]LITERAL]
          [,NUMREC = number]
          [,SEARCH_BUFFER = string, VALUE_BUFFER = string]
D        [, [NO]SHORT | [NO]SINGLE_FILE ]
          [,SKIPREC = number]
          [,SORTSEQ = { string | ISN } ]
          [,STARTISN = number]
D        [, [NO]USEREXIT]

```

BACKUP_COPY

```

BACKUP_COPY = number
              ,FILE = number
              [,FDT]
              [,NUMREC = number]
              [, [NO]ONLINE]
              [, [NO]SHORT | [NO]SINGLE_FILE ]
              [,SKIPREC = number]
              [, [NO]USEREXIT]

```

This function unloads records from an Adabas backup copy. You are not allowed to specify a LOB file. "BACKUP_COPY=number" specifies the ID of the database from which the backup copy was derived, and "FILE=number" specifies the file number. Both offline and online backup copies can be used. If a LOB file is assigned to the file specified, a partial reload using the ADAMUP parameters NUMREC, SKIPREC is not possible.

FDT

This parameter displays the FDT of the file to be unloaded.

FILE = number

This parameter specifies the file to be unloaded.

NUMREC = number

This parameter limits the number of data records retrieved from the file when unloading. All records are unloaded if NUMREC is omitted and SKIPREC is not specified. You cannot use NUMREC if a LOB file is assigned to the file to be reloaded.

[NO]ONLINE

This option indicates whether the backup copy might contain online data storage blocks for the file to be unloaded.

If the backup copy is expected to contain online data storage blocks, two passes are made when processing the backup copy. This is because the most recent version of each data storage block has to be found. Setting this option to NOONLINE unloads in one pass and saves a considerable amount of processing time, at the risk of ADAULD terminating with an error message if an online data storage block is detected.

The default used depends on whether or not the Adabas nucleus was active when the backup was made.

[NO]SHORT

This option indicates whether the descriptor values used to build up the index should be included in the output or omitted.

If SHORT is specified, no descriptor values are unloaded.

If the output is intended as direct input for the mass update utility, the file must be unloaded in NOSHORT mode.

SHORT and SINGLE_FILE are mutually exclusive.

NOSHORT is the default.

[NO]SINGLE_FILE

If this option is set to SINGLE_FILE, ADAULD writes the DVT and DATA information to a single data set (ULDDTA).

SINGLE_FILE and SHORT are mutually exclusive.

The default is NOSINGLE_FILE.

SKIPREC = number

This parameter specifies the number of records to be skipped before unloading is started. You cannot use SKIPREC if a LOB file is assigned to the file to be reloaded.

[NO]USEREXIT

A user-written routine is dynamically loaded. A pointer to an input parameter block and a pointer to an output parameter are passed with each call (please see the include file adauex.h for more information). For each record retrieved from the database, the decision can be made whether to unload the record (write it to the unload file), skip it or terminate execution immediately.

The environment variable/logical name ADAUEX_7 must point to a user-written routine.

See *Administration, User Exits and Hyperexits* for more details.

NOUSEREXIT is the default.

DBID

```
DBID = number
      ,FILE = number
      [,FDT]
      [,[NO]LITERAL
      [,NUMREC = number]
      [,SEARCH_BUFFER = string]
      [,[NO]SHORT | [NO]SINGLE_FILE ]
      [,SKIPREC = number]
      [,SORTSEQ = { string | ISN }]
      [,STARTISN = number]
      [,[NO]USEREXIT]
      [,VALUE_BUFFER = string]
```

This function unloads records from the specified database.

FDT

This parameter displays the FDT of the file to be unloaded.

FILE = number

This parameter specifies the file to be unloaded. You are not allowed to specify a LOB file.

[NO]LITERAL

If this option is set to LITERAL, leading blanks and lower case characters can be specified in the value buffer and remain relevant in the string, i.e. they are not removed or converted to upper case. If NOLITERAL is set, lower case characters will be transformed to upper case, and leading blanks will be suppressed except when the value is specified as a hexadecimal value.

NOLITERAL is the default.

NUMREC = number

This parameter limits the number of data records retrieved from the file when unloading. All records of the file are unloaded if NUMREC is omitted and SKIPREC or STARTISN are not specified.

SEARCH_BUFFER = string

This parameter is used to restrict the unloaded records to those which meet the selection criterion provided. The selection criterion must be provided according to the syntax for search buffer entries as described in the Command Reference Manual.

The maximum length of this parameter is 200 bytes. For complex entries, use the following method:

```
adauld: search_buffer=aa,20,a,d,\
> ab,10,a.
```

ADAULD will concatenate this to:

```
aa,20,a,d,ab,10,a.
```

The values which correspond to the selection criterion are provided by the VALUE_BUFFER parameter.

[NO]SHORT

This option indicates whether the descriptor values used to build up the index should be included in the output or omitted.

If SHORT is specified, no descriptor values are unloaded.

If the output is intended as direct input for the mass update utility, the file must be unloaded in NOSHORT mode.

SHORT and SINGLE_FILE are mutually exclusive.

NOSHORT is the default.

[NO]SINGLE_FILE

If this option is set to SINGLE_FILE, ADAULD writes the DVT and DTA information to a single data set (ULDDTA).

SINGLE_FILE and SHORT are mutually exclusive.

The default is NOSINGLE_FILE.

SKIPREC = number

This parameter specifies the number of data records to be skipped before unloading is started.

When used together with the STARTISN parameter, positioning is carried out before skipping.

SORTSEQ = string

This parameter controls the sequence in which the file is unloaded. If specified, it may either contain the field name of a descriptor, sub- or superdescriptor (1) or the keyword 'ISN' (2). The default is physical sequence (3).

1. Logical sequence

If a string specifies a field name of a descriptor or sub/superdescriptor, the records are unloaded in ascending logical sequence of the descriptor values to which the field name refers. The field name must not refer to a descriptor contained within a periodic group.

If the field name refers to a descriptor which is a multiple-value field, the same record may be unloaded more than once (once for each different descriptor value in the record). Therefore, it is not recommended to use this type of descriptor to control the unload sequence.

If the field name refers to a descriptor defined with the NU or NC option, the records with a null value for the descriptor are not unloaded.

2. ISN sequence

If 'ISN' is specified, the records are unloaded in ascending ISN sequence.

3. Physical sequence

If the SORTSEQ parameter is omitted, the records are unloaded in the physical sequence in which they are stored in the Data Storage.

If a search buffer has been specified and the SORTSEQ parameter has been omitted, the records are unloaded in ascending ISN sequence.

STARTISN = number

If the SORTSEQ = ISN option is used or a search buffer is provided, the STARTISN parameter may be specified to start unloading at a given ISN rather than from the lowest ISN in the file. If the specified ISN does not exist, unloading starts at the next highest ISN found.

[NO]USEREXIT

A user-written routine is dynamically loaded. A pointer to an input parameter block and a pointer to an output parameter are passed with each call (please see the include file `adauex.h` for more information). For each record retrieved from the database, the decision can be made whether to unload the record (write it to the unload file), skip it or terminate execution immediately.

The environment variable/logical name `ADAUEX_7` must point to a user-written routine.

See *Administration, User Exits and Hyperexits* for more details.

`NOUSEREXIT` is the default.

VALUE_BUFFER = string

If a selection criterion is specified with the `SEARCH_BUFFER` parameter, this parameter is used to supply the values which correspond to the selection criterion. The maximum length of this parameter is 2000 bytes.



Note: See also `[NO]LITERAL`, which controls the conversion of the value buffer to upper case.

Examples

Example 1

```
adauld: backup_copy = 3, file = 6
```

File 6 on the backup copy of database 3 is unloaded. A TEMP data set and two passes through the backup copy may be required, depending on the default setting of the [NO]ONLINE option.

Example 2

```
adauld: backup_copy = 3, file = 6  
adauld: single, noonline
```

The same file is unloaded. Both data records and descriptor value table entries are written to the same output file. The backup copy is processed in one pass as no online blocks are expected. No TEMP data set is required.

Example 3

```
adauld: dbid = 3, file = 6, skiprec = 100
```

File 6 in database 3 is unloaded. The records are unloaded in the physical sequence in which they are stored in the Data Storage. The first 100 records found are not written to the output files.

Example 4

```
adauld: dbid = 3, file = 6  
adauld: numrec = 10  
adauld: sortseq = ab  
adauld: short
```

Ten records from file 6 in database 3 are unloaded. The values of the descriptor AB are used to control the sequence in which the records are retrieved. The values required to re-create the inverted list when reloading are omitted.

Example 5

```
adauld: dbid = 3, file = 6, sortseq = isn, startisn = 123
```

File 6 in database 3 is unloaded. The records are unloaded in ascending ISN sequence starting at ISN 123.

TEMP Data Set Space Estimation

When unloading from an Adabas backup copy without the NOONLINE option set, the TEMP data set is required to accumulate information about online block occurrences.

The formula $TRH=DRH/1000$ can be used as a rough estimate with the default TEMP block size (4 kilobytes).

The following formula may be used to calculate the exact requirements:

```
X = ENTIRE ((DRH / BSTD) * 4)
```

```
TRH = X + ENTIRE (X / BSTD / 8) + 1
```

where:

ENTIRE

the next highest integer

BSTD

TEMP block size in bytes.

DRH

highest Data Storage RABN in the database on the backup copy. The SUMMARY function of the ADABCK utility can be used to obtain this number.

TRH

highest RABN required on TEMP.

Restart Considerations

ADAULD has no restart capability. An interrupted ADAULD run must be re-executed from the beginning.

28

ADAVFY (Database Consistency Check)

▪ Functional Overview	410
▪ Procedure Flow	411
▪ Checkpoints	412
▪ Control Parameters	412
▪ Examples	417

The following topics are covered:

Functional Overview

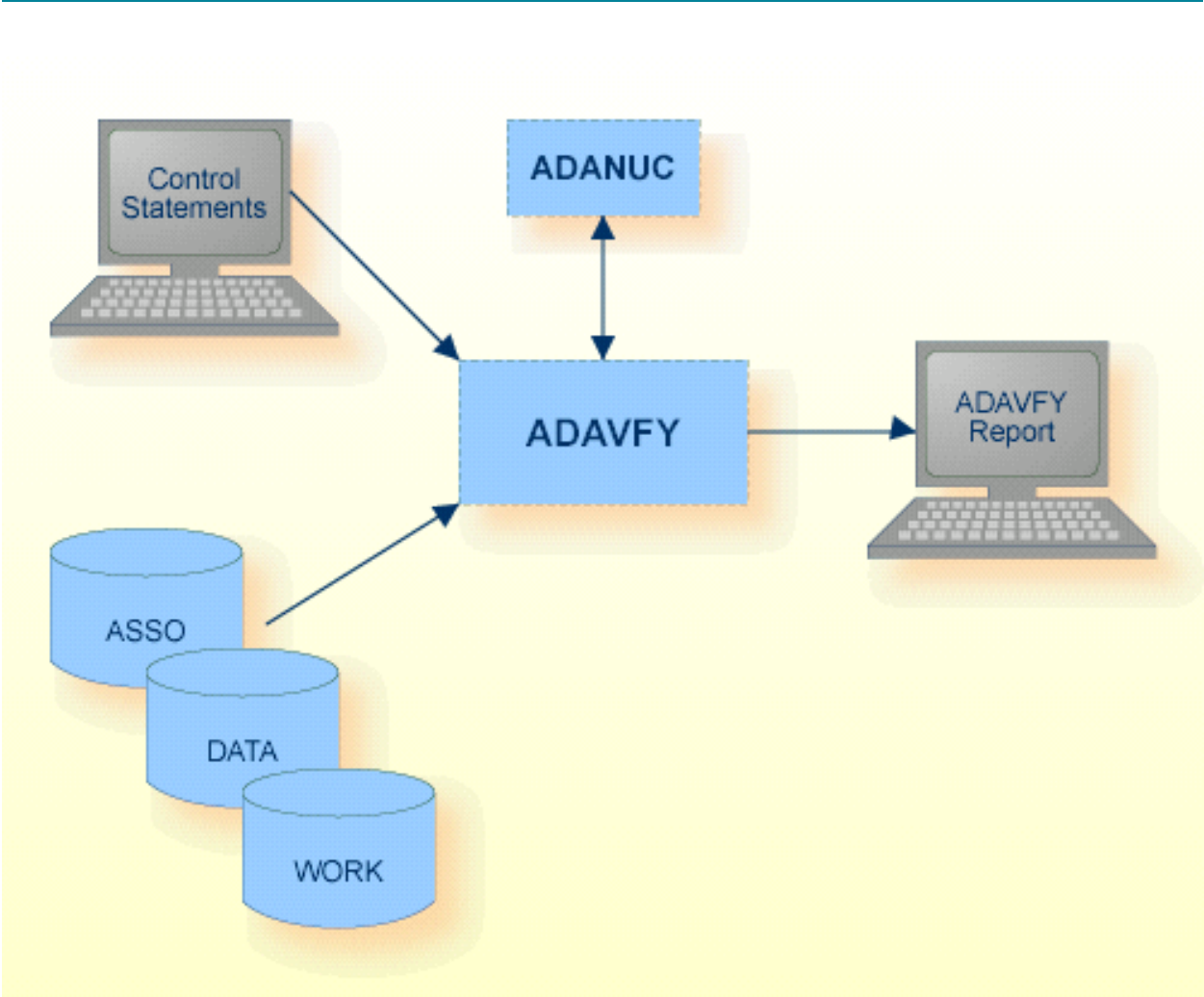
The ADAVFY utility checks the consistency of the database. The General Control Block (GCB) is validated together with each File Control Block (FCB) and each Field Definition Table (FDT) of the loaded files. The index structure and Data Storage are validated. ADAVFY can also search for lost RABNs.

Running ADAVFY against an active nucleus, or running in parallel with utilities that perform database updates, may result in errors being reported. This is because further updates can be made before the utility terminates and some of these updates are only reflected in the nucleus buffer pool. ADAVFY does not require the Adabas nucleus to be active; it processes the database offline.

In general, ADAVFY only displays consistency errors that it detects and it does not modify the database. However, the following error will be corrected if it is detected: the FCB contains a record counter for the number of records in the file, and if this counter has an incorrect value, it will be corrected.

This utility is a multi-function utility.

Procedure Flow



Data Set	Environment Variable/ Logical Name	Storage Medium	Additional Information
Associator	ASSOx	Disk, Tape	
Data storage	DATAx	Disk, Tape	
Control statements	stdin/ SYS\$INPUT		Utilities Manual
ADAVFY messages	stdout/ SYS\$OUTPUT		Messages and Codes
Work	WORK1	Disk, Tape	

Checkpoints

The utility writes no checkpoints.

Control Parameters

The following control parameters are available:

```
AC
CONSTRAINTS
DATA
M  DBID = number
D  ERRORS = number
FCB
FIELD
D  FILES = { * | (number [-number][,number[-number]]...) }
FROM = number - number
INDEX
D  LEVEL = number
LOST
RECORD
```

AC

```
AC
```

This function validates from the Address Converter to the Data Storage and checks that records can be found in the specified Data Storage for the files specified with the FILES parameter (see also DATA).

CONSTRAINTS

CONSTRAINTS

This function verifies the referential integrity constraints for the files specified with the FILES parameter.

DATA

DATA

This function verifies Data Storage for the specified file number(s). This function validates from the Address Converter to the Data Storage and from the Data Storage to the Address Converter for the files specified with the FILES parameter.

DBID

DBID = number

This parameter selects the database to be verified.

ERRORS

ERRORS = number

This parameter specifies the number of errors to be reported before the verification of a single file terminates. The minimum number allowed is 1. The default value is 20.

FCB

FCB

This function validates the file control block together with the Field Definition Table for the files specified with the FILES parameter (see also INDEX).

FIELD

FIELD

This function validates the Data Storage. It checks the record structure and validates the contents of unpacked, packed and floating point values for the specified files.

FILES

```
FILES = { * | (number[-number][,number[-number]]...) }
```

This parameter specifies the files to be verified. If an asterisk `*` is entered, all files will be verified. The FILES parameter is required for all functions except the LOST function.

The default is no files.

FROM

```
FROM = number - number
```

The values specified are used in conjunction with the LEVEL option to print various structures. Please refer to the LEVEL parameter in this section for more detailed information.

INDEX

```
INDEX
```

This function verifies the complete index to level 1 (Normal Index). This includes verification of the FCB and FDT.

ADAVFY also counts the number of used, free, reusable and lost NI (Normal Index, index level 1), MI (Main Index, index level 2) and UI (Upper Index, index level 3 or greater) blocks.

Example:

```
%ADAVFY-I-INDSTR, Index verification
%ADAVFY-I-INDCNT, NI: used: 210, free: 1773, reusage: 17, lost: 0
%ADAVFY-I-INDCNT, UI: used: 1, free: 87, reusage: 2, lost: 1
%ADAVFY-I-INDCNT, MI: used: 9, free: 87, reusage: 2, lost: 1
%ADAVFY-I-INDEND, Index verification completed ↵
```



Notes:

1. Used index blocks are index blocks that are currently in use.
2. Free index blocks are index blocks that have not yet been used.
3. Reusable index blocks are index blocks that already have been used, but that have become empty again and were included in the reusage queue. These blocks can be used again.
4. Lost index blocks are index blocks that are not currently used and that are missing in the reusage queue, and therefore cannot be used again. A value of 1 lost block is normal - this can happen after running ADAINV REINVERT.
5. The number of free, reusable and lost MI and UI blocks is the same, because these blocks are taken from the same logical extent. Please note that the numbers displayed are the numbers for MI and UI together - if you use additional space for MI blocks, this also reduces the number of space available for UI blocks.

LEVEL

LEVEL = number

This parameter specifies how much information ADAVFY should output concerning the internal structures. Specifying this parameter does not affect the degree of verification performed. If this parameter is used, it must be specified before the function in question.

The default value is the highest possible index level plus 1.

with INDEX function

Level n	prints information about index level n and higher
Level 0	prints more detailed structure of the index blocks

The FROM option is used to specify an index RABN range. Only the RABNs specified will be dumped.

with AC/DATA/RECORD/FIELD functions

Level 2	prints which RABNs processed
Level 1	prints record structure (when RECORD or FIELD is used) or where each ISN points (when DATA or AC is used)
Level 0	dumps fields within records

with LOST function

Level 0	dumps the physical structure of the database
---------	--

LOST

LOST

If this option is specified, ADAVFY searches for lost RABNs in the database. If any lost RABNs are found, the space can be recovered by using the RECOVER function of ADADBM.

RECORD

RECORD

This function validates the Data Storage and checks the structure of each record for the specified files (see also FIELD).

Examples

Example 1

```
adavfy: dbid=3,file=*,data,field,index
```

All files of database 3 are validated using the functions DATA, FIELD and INDEX. This combination of functions gives the maximum degree of validation.

Example 2

```
adavfy: dbid=3, file=7, level=1, field
```

File 7 of database 3 is validated. The record structure in Data Storage is validated, as well as the contents of unpacked, packed and floating point fields. ADAVFY prints a list of the RABNs which have been processed and, for each record processed, its offset in the corresponding RABN, its length and its ISN.

A

Appendix A - Example Utility Input Files

The Adabas kit contains example utility input data - this allows you to try out some of the Adabas utilities, and to load example data into the database so that you can gain experience of using Adabas.

The following Adabas demo files are provided with the Adabas kit:

File Number	Adabas File Name	Description
9	EMPLOYEES	File used for the C example program (see Command Reference)
11	EMPLOYEES-NAT	File used as example file by Natural containing employees data
12	VEHICLES	File used as example file by Natural containing vehicles data
13	MISCELLANEOUS	Example for a file containing a large number of fields
14	LOBFILE of 9	Lob file of the Adabas file EMPLOYEES



Notes:

1. On OpenVMS, file 9 is still the old Employees file without the LOB file 14. Please note that the C example delivered on OpenVMS is still the old version of the example program that corresponds to this old example file, which is different to the new version delivered for Windows and Unix and which is listed in the Command Reference document.
2. For creating the Adabas demo database (an Adabas database containing the Adabas demo files) on UNIX the command `crdemodb <dbid>` is available, on Windows there is an icon "Create Demo Database". Also the DBA workbench allows you to specify that the demo files are to be loaded, when you create a database.

The Adabas kit contains the following utility input files in the directory `$ADADIR/$ADAVERS/demodb` (UNIX) or `%ADADIR%\%ADAVERS%\demodb` (Windows):

File Name	Description
LoadDemo.bsh (UNIX) loadall.bat (Windows)	Script to load all demo files provided via ADAFDU, ADACMP and ADAMUP
loadfile.bat (Only Windows)	Script to load one of the demo files via ADACMP and ADAMUP – called by loadall.bat
emp.cmp	ADACMP parameters for the EMPLOYEES file
emp.cmpin	Decompressed demo data to be loaded via ADACMP and ADAMUP in the EMPLOYEES file
emp.fdt	FDUFDT file containing the FDT of the EMPLOYEES file
emp.fdu	ADAFDU parameters for the EMPLOYEES file
emp_nat.cmp	ADACMP parameters for the EMPLOYEES_NAT file
emp_nat.cmpin	Decompressed demo data to be loaded via ADACMP and ADAMUP in the EMPLOYEES_NAT file
emp_nat.fdt	FDUFDT file containing the FDT of the EMPLOYEES_NAT file
emp_nat.fdu	ADAFDU parameters for the EMPLOYEES_NAT file
mis.cmp	ADACMP parameters for the MISCELLANEOUS file
mis.cmpin	Decompressed demo data to be loaded via ADACMP and ADAMUP in the MISCELLANEOUS file
mis.fdt	FDUFDT file containing the FDT of the MISCELLANEOUS file
mis.fdu	ADAFDU parameters for the MISCELLANEOUS file
napp_backup.csh (Only UNIX)	Example script for External Backup on Network Appliance filers
napp_conf (Only UNIX)	Example configuration file for External Backup on Network Appliance filers
napp_restore.csh (Only UNIX)	Example script for External Restore on Network Appliance filers
ordexp.demo	ORDEXP file containing all Adabas demo files. This file is used if you click on Load Demo Files when creating a database with ADADBA
veh.cmp	ADACMP parameters for the VEHICLES file
veh.cmpin	Decompressed demo data to be loaded via ADACMP and ADAMUP in the VEHICLES file
veh.fdt	FDUFDT file containing the FDT of the VEHICLES file
veh.fdu	ADAFDU parameters for the VEHICLES file

For OpenVMS, the directory ADABAS\$EXAMPLES contains the following:

File Name	Description
demo.exp	Demo data to be loaded into the files EMPLOYEES-NAT, VEHICLES and MISCELLANEOUS via @ADABAS\$VERSION:DBGEN.COM

B Appendix B - prilogc

`prilogc` is an example program for printing a command log that is created with the nucleus parameter `CLOGLAYOUT` set to 6.

The Adabas kit does not contain an official utility for creating printable output from a command log created with the `ADANUC` parameter `CLOGLAYOUT=6`, but there is an example C program `prilogc`, which can be modified by the user to adapt the output. This program is not officially supported by Software AG - and it is not guaranteed that it will also be provided with future versions of Adabas.

The source file `prilogc.c` is stored in `$ADADIR/$ADAVERS/examples/server` on UNIX, and in `%ADADIR%\%ADAVERS%\..\examples\server` on Windows. This directory also contains a make file `makefile` to build the executable; the usage is described in the make file. The required header files can be found in `$ADADIR/$ADAVERS/inc` on UNIX, and in `%ADADIR%\%ADAVERS%\..\inc` on Windows.

The executable of `prilogc` is provided in `$ADATOOLS` on UNIX, and in `%ADATOOLS%` on Windows, which is included in the `PATH` setting provided by the Adabas installation.

`prilogc` expects that the environment variable `PRICLG` is set to the command log to be evaluated; parameters that can be specified for `prilogc` can be displayed by entering the following:

```
prilogc -h
```

