Version 2.32

Treehouse **Software** Inc.

ADAREORG User Guide

Trademark Acknowledgments

ADABAS and NATURAL are trademarks of SOFTWARE AG
of Germany and North America

MVS, MVS/XA, MVS/ESA and JES are products of IBM Corporation, USA.
MSP, MSP/AE and MSP/EX are products of Fujitsu, Japan.

Treehouse **Software** Inc.

409 Broad St, Suite 140
Sewickley, PA 15143

Phone: 412-741.1677
Fax: 412-741-7245
Email: tsi@treehouse.com
Web: www.treehouse.com

Version 232d, August 2005

Table of Contents

Chapter

1

Introduction

Overview

The traditional methods of making all but simple changes to the physical structure of established ADABAS databases are time consuming and cumbersome. Responding to requirements from application developers is not quick and easy for the DBA. Tuning opportunities that arise as experience is gained with new applications, or as end-user loads change, tend to be deferred.

ADAREORG transforms this situation. ADAREORG is a utility that enables the physical structure of a database to be changed with a minimum of effort and a minimum of risk.

ADAREORG enables the DBA to offer a higher level of service, not only in responding to the traditional sorts of requirements, but in going further. It is a simple matter to create appropriate test databases. A proto-typing or evolutionary approach to applications development may be easily supported; ADAREORG is useful in adding external data to an ADABAS database, or selecting ADABAS data for supply to a non-ADABAS system.

Scope

ADAREORG will accept one or two files as the data source. Where there are two, the user specifies the field in each file that is to be the key, and the basis upon which they should be joined. Input files may be:

a compressed ADABAS file;
a decompressed ADABAS file;
a raw data file;
a combination of any two of the above.

It is possible to create subsets of existing files by filtering input records based on specific selection criteria. It is also possible to limit the size of the output file to a specified maximum number of records. These features are useful in creating test databases, limited in size and content.

The power of ADAREORG extends below field level. For example, data in a specified position in an input field may be written to a nominated position in an output field. It is also possible to insert a constant in a specified position in an output field.

ADAREORG also allows fields within a record to be completely re-ordered, so that frequently accessed fields are placed at the beginning. This improves format buffer translation time and packing density, while being transparent to NATURAL programs.

ADAREORG can accommodate data format conversions, including changes in length, conversions between field types, moving fields into and out of periodic groups, etc. Without ADAREORG, such requirements are often serviced in a way that leaves files full of confusing, defunct fields.

ADAREORG can produce a compressed, re-structured file from compressed input files without the need for decompressed intermediate datasets. Decompression and compression is performed on a record by record basis, all within a single step. This eliminates the need for large amounts of temporary disk space.

Efficiency

ADAREORG offers improved efficiency through:

decreased DBA effort ;
reduced disk space usage;
reduced processing time compared with one-off NATURAL programs - due to ADAREORG's efficient ASSEMBLER code, and it's ability to produce a compressed output file from compressed input files in a single step.

The efficiency and convenience of ADAREORG enables the DBA to quickly and easily change physical database structures as required.

ADABAS version compatibility

ADAREORG V2.31 supports ADABAS V5.3, V6.1, V6.2 and ADABAS V7.1 and V7.4 databases seamlessly. All ADABAS field types are supported.

ADAREORG Version

This manual is specific to **ADAREORG Version 2.32** and subsequent fix levels, which is Year 2000 compliant. The ADAREORG version number is printed on all ADAREORG Execution Report page headers.

Chapter 2

Processing Overview

Input

Essential input to ADAREORG is one or two data source files - whether compressed or decompressed ADABAS files, raw data or a mixture.

In addition, the following are required:

ADACMP parameter cards describing the input records, (if using decompressed data) - ADAREORG will use the FDT if present; ADACMP parameter cards describing the output records to be produced. Except in the case of new fields, the same ADABAS 2-character short names from the input records are automatically used in the output records - thus providing a logical basis for the re-structure; control and processing parameters. Control parameters specify such things as whether or not input files are compressed, and the execution mode. Processing parameters affect the content of the output file.

Execution Modes

ADAREORG can be run in either of two modes: ADACMP Exit Mode, or Stand-alone Mode.

1. ADACMP Exit mode is used when compressed output is required, i.e. for typical database file reorganizations.

   With EXIT mode, the main program being executed is the compression utility, ADACMP. ADAREORG is specified to this utility as an exit routine to be called to perform user processing. It should be noted that, if the run abends, abend codes may have been issued either by ADARUN or by the compression utility.

   In ADACMP Exit Mode no decompressed files are produced. Processing of the input file is conducted on a record by record basis. Any compressed input records are firstly decompressed by ADADEC, then reorganized, and finally re-compressed by the ADACMP compression utility. User Exits provide the facility for tailoring the output.

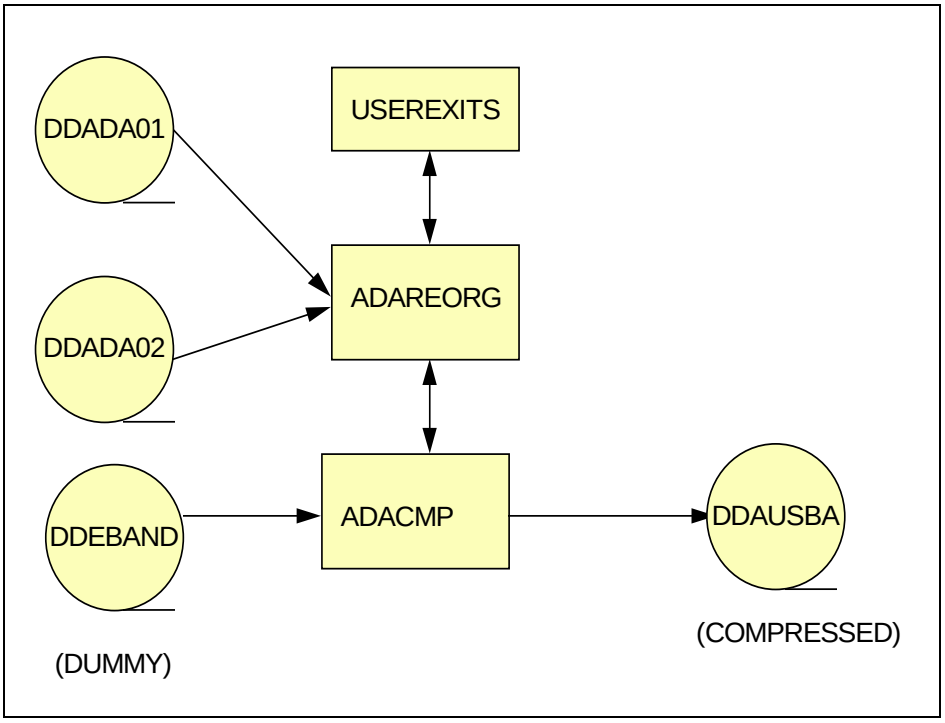

Figure 1: ADACMP exit mode - compressed output.

2. Stand-alone Mode is used whenever decompressed output files are required. Possible examples include:

   a requirement to produce input data for a non-ADABAS system, (such as a flat file ready for input to a relational database), or
   a need to submit a reorganized file to an existing verification program which requires a decompressed 'flat' file.

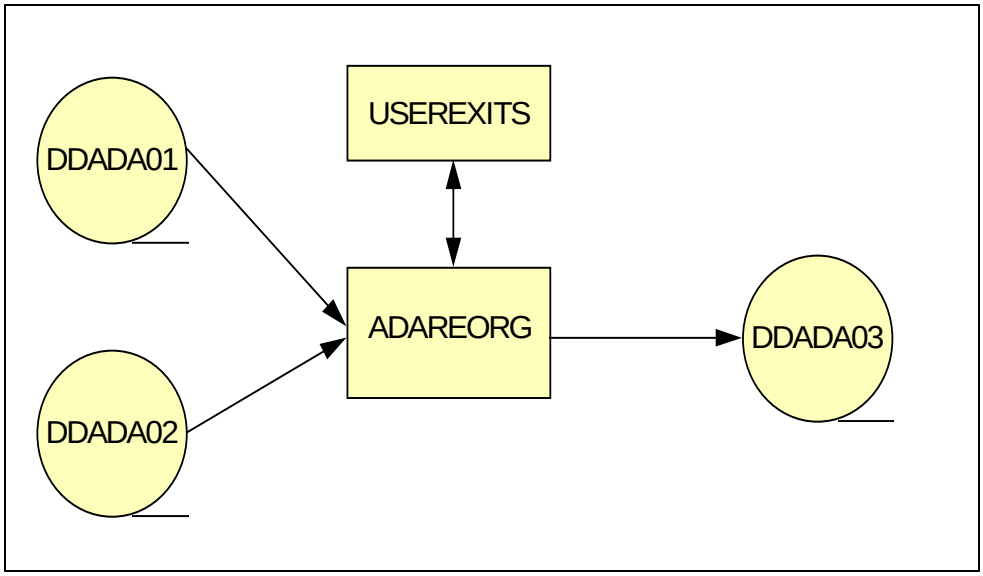

Figure 2 - Stand-alone mode - decompressed output.

Chapter

3

Operations

Operation JCL

Sample JCL is shown below for both execution modes of ADAREORG. This JCL can also be found in members JCL and JCL1 of the release install JCL library.

ADACMP exit mode

```
//STEP01   EXEC  PGM=ADARUN
//STEPLIB   DD    DSN=       need access to ADABAS load library
//*                           and the ADAREORG program
//DDPRINTX  DD  SYSOUT=*     documentation and error messages
//DDPRINT   DD  SYSOUT=*     ADACMP  messages
//DDADA01   DD  DSN=         input data, file 1
//DDADA02   DD  DSN=         optional second input file, file 2
//DDWAN01   DD  DSN=         ADACMP description of DDADA01
//DDWAN02   DD  DSN=         ADACMP description of DDADA02
//DDWAN03   DD  DSN=         ADACMP description of output file
//DDPARM    DD  DSN=         ADAREORG parameters
//DDCARD    DD  DSN=         ADARUN parameters
//DDKARTE   DD  DSN=         usually same as DDWAN03
//DDEBAND   DD  DUMMY        for ADACMP - must be dummy
//DDAUSBA   DD  DSN=         compressed output
//DDRUCK    DD  DSN=         ADACMP messages
//DDFEHL    DD  DSN=         ADACMP rejected records
```

Stand-alone mode

```
//STEP01    EXEC  PGM=      ADAREORG
//STEPLIB   DD    DSN=      need access to ADAREORG and the
//*                           ADABAS load library
//DDPRINTX  DD    SYSOUT=*  documentation & error messages
//DDADA01   DD    DSN=      input data, file 1
//DDADA02   DD    DSN=      optional second input file 2
//DDADA03   DD    DSN=      decompressed output file
//DDWAN01   DD    DSN=      optional ADACMP description of
DDADA01
//DDWAN02   DD    DSN=      optional ADACMP description of
DDADA02
//DDWAN03   DD    DSN=      ADACMP description of output file
//DDPARM    DD    DSN =     ADAREORG parameters
```

Description of DDNAMES

DDNAME	Usage	Description
DDPRINT	Mandatory in exit mode.	Required by ADACMP - not needed in standalone mode. Normally assigned to SYSOUT=*.
DDADA01	Mandatory	The sole or first input file. Depending on the type of file, the following considerations apply: 1. If the input file is a compressed ADABAS file which has been unloaded via ADAULD, then "EXPAND 1" will be required as input to DDPARM. It is also essential that a full unload be taken (i.e. without the MODE=SHORT option) so the unloaded file can double as a backup of the ADABAS file. 2. If the input file is decompressed, it should have been created via the ADADEC decompression utility, using the parameters RECFM=VB and BLKSIZE=10000. 3. Alternatively, the file may contain raw data (such as a sequential dataset with RECFM of F, FB, V or VB and records of any length). This data must be consistent with the descriptions in DDWAN01. Note that an ISN may or may not be needed in the first 4 bytes, and that 1-byte binary occurrence counters must precede any PE and MU data.
DDADA02	Optional	If used, the nominated second input file will be merged (according to the rules set out by the parameters in DDPARM) to DDADA01 to produce the reorganized output file. Both DDADA01 and DDADA02 must be ordered on specified key fields. The output record structure will be generated for records with matching key values. If DDADA02 is compressed, then the "EXPAND 2" parameter must be included as input to DDPARM. (Note, however, that this parameter should NOT be supplied unless two input files are being used).

DDNAME	Usage	Description
DDADA03	Conditional	This file is produced only if ADAREORG is run in stand-alone mode. The resulting output is decompressed and its description is in DDWAN03. If DCB information is not specified then it will default to RECFM=VB and BLKSIZE=20000, otherwise RECFM can be F, FB, V or VB and the record length must be large enough to accommodate the largest record built. **Note:** LA output fields default to 16381 bytes when empty, thus output blocksizes should be appropriately sized. If "USERISN" is specified in DDWAN03, the first 4 bytes of the record will contain the ISN obtained from DDADA01 or DDADA02, if available. If the "ISN" parameter is specified in DDPARM, ISN's will be generated internally by ADAREORG. Repeating fields (MU) or groups (PE) will be preceded by a 1-byte binary occurrence counter.
DDWAN01	Conditional	Contains the ADACMP cards describing DDADA01. Not needed when DDADA01 is a compressed unloaded file (as ADAREORG can derive the file description from the input). It may consist of in-stream data, a sequential file or a PDS member. A record length of 80 bytes is assumed.
DDWAN02	Conditional	Required only if DDADA02 is present., and subject to the same considerations as for DDWAN01 above.
DDWAN03	Mandatory	Contains ADACMP cards describing the reorganized output file built in DDADA03 (or in DDAUSBA if running as an ADACMP exit). When running as an exit, DDWAN03 would

DDNAME	Usage	Description
		normally be the same as DDKARTE - in fact both DD statements can point to the same physical dataset.
DDPARM	Optional	This file contains ADAREORG control and processing parameters. It may be in-stream data, a sequential file or a PDS member. Record length is 80 bytes.
DDCARD	Conditional	Must be present when running in exit mode only (i.e. with "EXIT" as a parameter in DDPARM). This card must contain: `ADARUN PROGRAM=ADACMP,UEX6=ADAREORG`
DDKARTE	Conditional	ADACMP cards - needed in **exit mode** only. Normally identical to DDWAN03.
DDEBAND	Conditional	Needed in **exit mode** only. Must be set to DUMMY.
DDAUSBA	Conditional	Needed in **exit mode** only. ADACMP will generate it with RECFM=VB, LRECL, BLKSIZE as appropriate for input to ADALOD. This is the output file as described in DDWAN03 and DDKARTE. Typical values are LRECL=9996 and BLKSIZE=10000.
DDDRUCK	Conditional	Needed in **exit mode** only. Contains ADACMP messages. Usually specify as SYSOUT=*.
DDFEHL	Conditional	Needed in **exit mode** only. One record is produced for each invalid record rejected by ADACMP. See ADACMP documentation for detailed explanation. The first few bytes usually contain, among other things, the name of the field in error.
DDPRINTX	Mandatory	Normally assigned to SYSOUT=*. Contains 1 page of documentation, plus images of the input in DDWAN01, DDWAN02, DDWAN03 and DDPARM. Also contains

DDNAME	Usage	Description
		record counts, any error messages, and (in the case of an abend during processing) a small formatted dump.

Output

Output files

The output file will usually be compressed by ADACMP, so its ADACMP description should be complete. While it is possible to define contiguous fields as one large field in the DDWAN03 description, there is not a great deal to be gained from this - so the output description used by ADACMP found in DDKARTE should be complete (including super- and sub-descriptor information).

End of processing messages

At the end of processing the following counts will be printed:

```
RECORDS WRITTEN          :
RECORDS READ FROM ADA01  :
RECORDS READ FROM ADA02  :
REJECTED MATCHES         :
```

Note: For the purposes of these messages, ADABAS records are counted (which are not necessarily the same as physical records). Also, if input is compressed, the 1st "record" is an FDT and is not counted.

Chapter 4

Parameters

File Definitions

General Syntax Rules

Descriptions of the input file(s) and the output file must use standard ADACMP syntax as described in the ADABAS utilities manuals. Either form, including ADACMP "FNDEF" syntax, may be used for ADAREORG, but the correct cards for the compression utility concerned must be supplied to DDKARTE.

"USERISN" is the only ADACMP keyword recognized by ADAREORG. If other keywords are supplied, they will be ignored.

Example:

```
ADACMP COMPRESS
ADACMP FNDEF='01,FF,2,A,MU'
```

Super Descriptors — Super/Sub/Phonetic-descriptor field definitions can be present but are ignored by ADAREORG.

Comments — Bytes 72-80 are ignored. Bytes 46 to 72 may be used for comments depending on the rules for the relevant utility.

Level Numbers — ADAREORG deals only with elementary fields, so whilst group fields are ignored, levels within the group structure are not. Note that this applies to ADAREORG parameter input also.

PE/MU fields — PE fields are accepted. Members of a PE must have levels greater than 1. Note that PE or MU occurrence counts should **NOT** be used on ADACMP cards. If a MU or PE is specified, ADAREORG expects a 1-byte binary counter to appear in the input record preceding the first occurrence. In the output file it will generate the binary counter before the variable number of occurrences of the multiple-valued field.

Note: As with standard ADACMP definitions PE's may not be nested but may contain MU's.

Standard Length This must be present. ADAREORG does NOW support undefined lengths, ie. 0.

Standard Format Currently this must be A,U,B,P, or F. The G (floating point) option of ADABAS Version 5 is partially supported - the fields are internally translated to B format by ADAREORG.

Definition Options The standard definition options for field descriptions may be used ie. NU, FI, DE, UQ, MU and PE. However, only MU and PE are used by ADAREORG.

USERISN

If an input file is decompressed and contains an ISN (as would be the case if the data were created by ADADCU using the ISN option or by CCA's ADASTRIP utility), its description must include "USERISN".

If an input file is compressed, "USERISN" in its description will cause ADAREORG's internal decompression routine to retain the input ISN's in the decompressed records.

If an ISN is to be produced in the output file then its description must contain "USERISN".

An ISN may be included in the output file in one of two ways:

ISN's may be carried through from the input file. Given that the input file description contains "USERISN", the inclusion of "USERISN" in the output file description will cause ADAREORG to retain the input ISN's in the output. Note that when the input file is compressed, and USERISN is specified for the output file, ISNs are automatically carried through to the output.

If there are two input files, both with "USERISN" in their descriptions, ADAREORG will take the ISN's from file 1. If file 1 has no ISN but file 2 is being used and does have an ISN, then this will be used for the output. Carrying through ISN's from the input will be necessary where there are ISN-based links in the database.
ISN's may be generated by ADAREORG as a result of the inclusion of "ISN" among the ADAREORG parameters. ISN's generated in this way will overwrite any carried through from the input. This ADAREORG option will seldom be required - if ISN's are not included in the output file, then ADALD1/ADALOD will insert them.

Generating Input File Cards

ADACMP cards can be generated for existing files in the following ways:

using PREDICT;
using UTIL03 of the old batch data dictionary;
from an ADABAS ADAREP report;
from a program that issues LF commands to ADABAS and hence builds ADACMP cards.

Note, however, that regardless of which method is used, it is very important that the cards are correct.

An example of the last method listed above is provided in member NATFDT in the installation source library. NATFDT is easy to run and ensures that ADACMP cards generated reflect the current physical structure of the file to be reorganized - however, field order is not always correct, which can cause obvious problems. It is provided as a sample only.

If only a few fields from the input files are being selected by ADAREORG, then their ADACMP cards may be abbreviated, as follows:

contiguous un-referenced fields may be defined as one large field;
fields occurring after the last referenced field may be omitted - although if a PE group has any of its elements referenced, then its ADACMP description must be left intact.

If an input file is decompressed, one input field can be defined as two in order to generate output from part of a field. This can also be achieved via ADAREORG's "LET" Parameter.

Note - If using an unloaded file as input to ADAREORG, it is not necessary to provide ADACMP cards to describe the input file. The description will be obtained directly from the internal FDT located in the first block of the dataset.

Control & Processing Parameters

ADAREORG's parameters are described here in two groups: **control parameters** which must be set to ensure proper running of the job; and **processing parameters** which enable the DBA to accomplish nearly any re-structuring task.

General Syntax Rules

The following rules are applicable to both types of parameters:

Maximum length of a parameter record is 72 bytes.
The parameter keyword (which may be abbreviated to 3 bytes) must start in position 1. Valid keywords and their respective abbreviations include:

ADAVER	ADAVER
EXPAND	EXP
EXIT	EXI
ISN	ISN
INC	INC
KEY	KEY
ACCEPT	ACC
ACCEPTO	ACCEPTO
REJECT	REJ
REJECTA	REJECTA
LET	LET
DEFINE	DEF
APPLY	APP

The syntactical delimiters '=' '<' '>' ',' '(' and ')' may be surrounded by blanks to improve readability.
Comments can be appended to individual records by simply inserting a period before the comment text. Any text following a period is ignored by ADAREORG.
Parameters cannot be continued onto another record.
The use of square brackets [ ] indicates that the enclosed items are optional.

Note: Keywords must be in upper case - lower case keywords are not recognized.

Control Parameters

Parameter	Format / Explanation
ADAVER	ADAVER n Where n=5,6, or 7, meaning ADABAS V5, V6 or V7. This card specifies the version of ADABAS used to produce the input file. If the input is a flat file, then use the version number of ADABAS that will be used to process the output. This parameter is **optional as of V232a, default value is 7 for ADABAS V7**.
CODE	CODE=XXXXXXXXXXXXXXXXXXXX The Codeword is at least 20 bytes long and is supplied by your local distributor once you have advised them of your CPU ID. It must be the first input parameter in DDPARM and **must not be in quotes**. This parameter is mandatory.
EXPAND	EXPAND {f} Where f = 1 or 2. This parameter must be submitted for each compressed input file, ie. DDADA01, DDADA02 or both, signifying that it is to be decompressed by ADAREORG via a call to ADADEC, eg. `"EXPAND 1".`
EXIT	EXIT This parameter must be present if ADAREORG is to be run as an ADACMP exit. Note that it has only one acceptable value: "EXIT".
INC	INC nnnnnnnn This parameter forces Adareorg to print a progress message on the joblog after processing every nnnnnnnn input records. Records from both input files are counted in this total.
ISN	ISN This parameter will cause ADAREORG to generate an ISN (Internal Sequence Number) equal to the sequence number of the output record. E.g. the first output record will receive an ISN of 1 and will increment by 1 for each output record. The ISN is stored in the first 4 bytes of each output record (or internal record

image if running as an ADACMP exit).

Note that this parameter has only one acceptable value ("ISN") and must be used in conjunction with the "ADACMP USERISN" parameter to be effective.

KEY KEY [f:]xx

This parameter must be present when there are two input files. It specifies the basis upon which they are to be joined, where:

f = file number (optional for first file) - valid values are 1 or 2;
xx = ADABAS field name of key field.

Note that::

the input records must be sorted in key order;
duplicate key values within a file are permissible;
"unmatched" key values (i.e. those which occur in only one file) are permissible, but will not generate any output;
each input record will only be used once to generate an output record - thus a key duplicated in one file but unique in another will only produce one output record (the first occurrence of the duplicated record will be used);
if key matching is not sufficient for your requirements, it can be overridden by using a user exit 1 - see below for details;
Variable fields can be used for key matching. However, results can be unreliable; for example, if the same field is short in one record but long in the next, it may result in non-selection of expected records.

Examples:

```
KEY   1:AA
KEY   2:BB

KEY   AA
KEY   2:BB
```

Both examples resolve identically: field AA from file 1 and field BB from file 2 are to be taken as the key fields for joining the two files.

Processing Parameters

Parameter **Format / Explanation**

ACCEPT The ACCEPT or ACC statement defines that input records from file **n** will be accepted only if the value contained between bytes **i** and **j** (inclusive) of field XX is equal to the specified constant.

```
ACCEPT [f:]XX(i,j) = kkkkkk
ACCEPT [f:]XX(i,j) < kkkkkk
ACCEPT [f:]XX(i,j) > kkkkkk
```

Where:

f = input file number, which can be either 1 or 2 (defaults to 1 if omitted);
XX = 2 character ADABAS field name;
i = byte offset of the start position within the field to be examined (the first byte in the field is byte nr. 1);
j = byte offset of the ending position within the field to be examined;
kkkkkk = a string describing a constant.

A constant can be defined in one of the following three ways:

"EMPTY" - this means a null, blank or zero of appropriate length;
"CHAR or CHA (a)", where "a" is a character string, including blanks, which must not include any of the following delimiters:
'=' '<' '>' ',' '(' ')'
"HEX (x)", where "x" is a hexadecimal string.
It is not necessary to explicitly give a constant a data type, because ADAREORG will assume the constant is of the same type as the ADABAS field to which it refers. ADAREORG will also perform any necessary truncation or padding of the constant automatically. Hence, the keywords CHAR and HEX are provided only to allow any possible constant to be defined. They should not be confused with ADABAS data types.

In the ACCEPT or ACC statement, note that:

if f is omitted, 1 is assumed;
if i & j are omitted, the whole of the specified field will be compared with the specified constant;
it is not permitted to specify i or j without specifying both.

Examples:

```
ACCEPT AA = HEX(000F)
ACC 2:AB < CHA (M N)
```

Up to 20 ACCEPT, REJECT or APPLY cards in total may be supplied. Note that ACCEPT does not unconditionally accept an input record for processing, as it may be rejected by subsequent ACCEPT or REJECT statements.

ACCEPTO The ACCEPTO ("accept-OR") statement differs from the ACCEPT statement, in that when the condition on this card is met, the record satisfying the condition is immediately accepted, without consideration of **following** parameter cards. If the condition specified on the card is **not** met, then the card is IGNORED. This can have a "surprising" consequence, ie suppose that the only card specified is ACCEPTO, and NONE of the records in the file meet the condition on the card. Nevertheless, all records will be accepted! This is because the ACCEPTO card is ignored when its condition is not met, and the DEFAULT action for ADAREORG is to **accept** all input. I.e. if no selection parameter cards are specified at all, then all input is accepted.
*ACCEPTO can be used to accept a **subset** of records that would otherwise be rejected, by placing it in a parameter deck **before** the REJECT cards.*

REJECT The form of REJECT or REJ is identical to ACCEPT (as is the logic) except that REJECT describes input records that should be rejected if the comparison is true.

```
REJECT [f:]XX(i,j) = kkkkkk
REJECT [f:]XX(i,j) < kkkkkk
REJECT [f:]XX(i,j) > kkkkkk
```

NOTE: all supplied ACCEPT and REJECT statements are logically **AND**ed together to form one definitive filter argument, ie. a record must satisfy all specified criteria in order to be accepted.

If the "KEY" parameter is used then ACCEPT/REJECT processing is done after joining the two records.

Examples:

```
REJECT  AA = HEX(0F)
REJ 2:AB = CHA(MN)
```

REJECTA The REJECTA ("reject-and") statement differs from the REJECT statement, in that it will only reject those records for which the conditions on all REJECTA cards are met. (Any REJECT will reject a record **immediately** if its condition is met).

APPLY The APPLY or APP parameter is used to invoke a user exit previously defined in a DEFINE statement.

APPLY UX**na**(**xx**)

Where:

"**n**" is either 2 or 3 (representing User Exit 2 or User Exit 3).
"**a**" is an option for User Exit 3, where a letter of the alphabet may be used to define up to 27 exists;
and "**xx**" is the ADABAS 2-character short name. There can be many APPLY cards per user exit if needed. If UX2 or UX3 is defined then 1 or more APPLY cards must be present for each DEFINE.

LET The LET parameter allows a value to be assigned to an output field, or part of an output field.

```
LET  XX(i,j)  = kkkkkk
```
or
```
LET XX(I,j) = [n:] YY (p,q)
```

Where:

XX and YY are 2-character ADABAS field names.
XX is defined in DDWAN03, YY is defined in DDWAN01 and/or DDWAN02;
i,j = start and end byte offsets of the part of the output field (XX) to be changed;
kkkkkk = a constant
p,q = start and end byte offsets of the input field (YY) to be used as input to the change.
n = input file number, which can be either 1 or 2 (defaults to 1 if omitted);

It effectively says:

Write the defined string **kkkkkk** into output field **XX** between bytes **i** & **j** inclusive (the string should be defined as for the ACCEPT statement), or

Take the contents of bytes **p** through **q** of input field **YY** from input file **n**, and write them into bytes **i** through **j** in output field **XX**.

Note that::

The data written into the specified output position overlays any data that would have otherwise been there. See next point.
If the field named in XX also occurs in the input file, the data in the "unchanged" part of the output field XX (i.e. that part which is outside of the specified i to j byte range)` will come from field XX in the input file.
if n is omitted, 1 is assumed;
rules for i, j and p,q are as for ACCEPT, i.e. they must always appear in pairs;
for a given output field a maximum of 2 LET statements are allowed (except where XX is not also an input field name - in which case 3 LET statements are allowed);
the rules for the constant are as for ACCEPT;
if XX is a variable length field, then (i,j) may not be specified.

Examples:

```
LET  AC(3,5) = HEX(616161)
LET  AC(3,5) = 2:BH(1,3)
LET  AA = EMPTY
```

DEFINE The DEFINE or DEF parameter specifies a user exit to be used during ADAREORG processing.

```
DEFINE UXna=xxxxxxxx
```

Where:

n has the value of 0, 1, 2, or 3;

a is as described in the APPLY statement previously;
xxxxxxxx is the name of a load module which is the user exit loaded at the start of ADAREORG processing.

Each user exit can only be defined once and must be available to ADAREORG in a STEPLIB. Sample user exits can be found in the installation source library with names starting with the characters UX.

The user exits are described in more detail in the appendix (see User Exits).

LIMIT This parameter will cause processing to terminate as soon as a pre-determined number of output records (**n**) have been

generated.

LIMIT n

If **n** is 0, ADAREORG will not attempt to open the input files. This may be useful in order to validate parameters.

Example:

LIMIT 9999

Chapter

5

Field Format Conversions

This chapter describes how ADAREORG processes field format and data type conversions, and what results can be expected in each case.

Length Conversion

Alpha fields are moved progressively from left to right, starting with the leftmost byte. The target field is truncated if the source is longer, or blank-filled if the source is shorter.

Binary is moved similarly but is padded with null's.

Packed and unpacked data is moved progressively from right to left, starting with the rightmost byte and truncated if necessary. Surplus bytes in the target field are padded with null's (packed) or zeros (unpacked).

Data Type Conversion

Warning - Data type conversions should be undertaken with care, as some results are non-reversible, eg. when truncation of data occurs. Always make a backup copy of the input file prior to undertaking data type conversion.

Special Note - An explanation of the terminology used to describe data type conversion results, is as follows:

⇒ A "**simple conversion**" is defined as one where the field type is changed to reflect the desired result, but the field contents remain essentially unaltered in any other way. This can only be achieved in instances where the source format and the target format are compatible, eg. "alpha-to-binary" or "packed to unpacked".

⇒ A "**complex conversion**" is defined as one where the source format and the target format are incompatible and field contents may therefore be physically altered in some way, eg. "alpha-to-numeric". Complex conversions can result in truncation or other necessary modification to data.

From ALPHA (type A)

Alpha-to-binary is treated as a simple conversion.
Alpha-to-numeric (of any type) is a complex conversion. Any trailing blanks or excess data beyond 32 characters is dropped, non-numeric characters are translated to zeros, and the result is then treated as if it were unpacked.
Conversion to G type will accept a decimal point and a "+" or "-" sign in the input data but exponents are not yet supported.

From UNPACKED (type U)

As UNPACKED fields are essentially compatible with any other type, all conversions from UNPACKED are treated as simple conversions.
Note - up to 27 signed digits can be accommodated in a U type field.

From BINARY (type B)

Binary-to-alpha is treated as a simple conversion.
Binary-to-fullword is treated as a complex conversion. Only the rightmost 4 bytes are retained (padded on the left with null's if necessary) and the resulting field type changed to fullword.
Binary to any other form of numeric is straightforward after conversion to fullword first.

From FULLWORD (type F)

Fullword to either packed or unpacked is treated as a simple conversion.
Fullword to binary or alpha is also treated as a simple conversion.

From PACKED (type P)

Packed-to-alpha is treated as a complex conversion.
Packed-to-binary is treated as a simple conversion.
Packed-to-unpacked or any fullword numeric form is treated as a simple conversion.
Note - up to 27 digits of signed packed decimal (14 bytes) can be accommodated in type P field.

From Floating Point (type G)

Floating point (type G) is now fully supported as a proper numeric type.

Conversion to A type will result in an alphanumeric string in engineering format, e.g. +1.23456789E+10 . Conversion from a 4 byte G type field will produce a 15 byte A type field, from an 8 byte G type field will produce a 24 byte A type field. Shorter A fields will result in right hand truncation and complete mangling of the actual value, as the exponent will be (at least partially) missing.

When converting to UNPACKED, the number is truncated, not rounded.

Note - As floating point numbers can far exceed the capacity of packed decimal, it is recommended to carefully check these conversions.

Note – Conversion of any numeric type to an A type now results in the A type field having a leading sign, either "+" or "-" as appropriate. This differs from type U, where the sign is contained implicitly in the last byte.

Long Alphanumeric fields (type LA)

LA fields are specified by adding LA to the end of a normal field specification. Input may only be from other alphanumeric fields (either long or short), and is treated as a simple conversion.

Wide Alphanumeric fields (type W)

These are assumed to be Unicode fields, both as input to, and output from conversions. "A" type input fields are assumed to contain EDCDIC data, when a W field is the output. W to W however is a simple conversion, and no translation takes place, hence the content need not necessarily be Unicode.
All W fields must have an even length, as must any subfield selections therefrom. This is also true for comparison constants. This is because W fields use 2 bytes for each character. Subfield selection must also start on an odd numbered byte within the parent field.

Periodicity (PE/MU) Conversion

The following rules apply when manipulating **Periodic** fields (PE) and/or **Multi-valued** fields (MU). Please take careful note of the following field processing performed by ADAREORG, as data may be irretrievably truncated during "periodicity" conversions.

> Where a simple field is built (wholly or partly) from a PE or MU field, the first occurrence will be used. If the source field is a MU in a PE, then the first MU occurrence in the first PE group occurrence will be used.
> If a PE group or MU field is built solely from simple fields and/or constants, it will contain 1 occurrence.
> Where a MU is a field built from a mixture of fields (including a PE or MU or both) the periodicity of the constructed field will be that of the component field with the lowest periodicity. Note that constants are not counted - they "come along for the ride" on each occurrence but do not determine the output periodicity - unless all component fields are constants.
> A PE group built from a mixture of fields is handled in a similar way as the MU field described above. The periodicity is acquired from the component field with the lowest periodicity in the periodic group. For this purpose, the periodicity of a MU field in a PE group is taken as the highest PE index - the first MU occurrence will always be used for the data to be moved.

If a MU within a PE is to be built from a simple PE element, it is assumed to have a MU periodicity of 1 and the usual PE periodicity. If it is to be built from a simple MU element, it is assumed to have a MU periodicity of 1 and the PE periodicity is taken to be the MU periodicity.
Note - when an MU within a PE is to be built from an MU outside a PE, then the new MU has 1 occurrence in each PE occurrence, and there will be as many PE occurrences as there are occurrences of the MU outside the PE.
Note: "Periodicity" means the maximum index of a field, with the proviso that the periodicity of a MU field in a PE group will be the maximum MU index where the PE index is set to 1. That is, the periodicity of an MU within a PE is taken to be that of the MU in the "first occurrence" of the PE field. This is then used (with the periodicity of the other component fields) in determining the final periodicity of the new field, according to the third point above, eg. a new MU field is to be built from two different PE fields. In a given input record, the first PE field may occur 10 times. The second PE field may only occur twice in the same record. The new MU field will then have 2 occurrences, in which the first 2 occurrences of the first PE field are combined with the only 2 occurrences of the second PE field. The remaining 8 occurrences of the first PE field are ignored.

Chapter 6

Error Messages

Fault Diagnosis

Although ADAREORG has proven to be very robust and reliable, it is wise to check that decompressed output is as expected. This can be achieved by using the "LIMIT" parameter to generate a test output file with a small number of records (eg. LIM 99) and then browsing the output file with SPF or the DBAUDIT utility. As in any data conversion exercise, it is also prudent to retain the original unloaded file until satisfied that the reorganized file has been tested and validated.

Other checks include:

ADAREORG output report of ADACMP cards - are they consistent with the input data?
are user ISN's needed?
ensure that PE and MU occurrence counts are NOT included in the ADACMP cards. Such counts will be ignored by ADAREORG - but not by ADACMP - so the data will probably be corrupted;
ensure that all PE's and MU's have the correct level numbers;
are output file block sizes, record lengths and space allocated large enough?
if possible, browse the output records for validity. If a suitable application exists for the data, use it to check that the records (and fields) pass application edits;
ensure that the physical output file is as expected, in terms of the number of records loaded and the space occupied by those records.

If invalid data is input to ADAREORG (eg. a compressed file is input without the appropriate "EXPAND" parameter, or ADACMP cards are inconsistent with the file structure), it will probably fail with an error S002. An incorrect file is very unlikely to get through both ADAREORG and ADACMP without failing..

ADABAS Messages

ADARUN and ADACMP error messages are documented in the ADABAS messages and codes manual. The dataset DDFEHL may contain details of any records rejected by ADACMP.

Note that when ADAREORG is being run in EXIT mode, ADARUN and ADACMP get initial control. This means that:

ADARUN cards must be correct and complete (as for any ADABAS utility run);
abend codes may have been issued by ADARUN.

ADAREORG produces summary output even when an error is found. If this output is not evident after an abend, then it probably indicates that ADAREORG did not get executed - and the abend is probably due to ADARUN (perhaps an incorrect ADARUN parameter).

System ABENDS

System Abend Code	Description
S002	This may occur if invalid data is entered.
S806	This usually means that you have an incorrect STEPLIB in your JCL, e.g. no library containing ADADEC, ADAREORG or another required module.
S80A	This may indicate that the region is too small.

User ABENDS

In the event of an abnormal termination, the WTO message "ADAREORG - ABNORMAL TERMINATION" will be issued. All open files will be closed and the job step will ABEND with one of the following codes:

Abend Code	Description
U001	failure attempting to print the first page of output (by module REODOCP).
U002	error detected while processing DDWAN01. This will be accompanied by other error messages. It could be caused by being unable to open the file.
U003	error detected while processing DDWAN02. This will be accompanied by other error messages.
U004	error detected while processing DDWAN03. This will be accompanied by other error messages.
U005	error detected while processing DDPARM. This will be accompanied by other error messages.
U006	failure to open DDADA01.
U007	failure to open DDADA02 (where DDWAN02 was present).
U008	this message code no longer used.
U009	failure while processing a DDADA01 record against the internal record representation. This usually indicates that the actual input record layout doesn't match the FDT that is provided (or contained in the file).
U010	failure while processing a DDADA02 record against the internal record representation of data.
U011	failure to open DDADA03.
U012	failure while building a record for DDADA03.
U013	failure processing FDT for DDADA01. Indicates that the respective FDT failed to correctly decompress.
U014	failure processing FDT for DDADA02. Indicates that the respective FDT failed to correctly decompress.
U015	decompression of data failed on DDADA01. See also the WTO message beginning with "ADADEC".
U016	decompression of data failed on DDADA02. See also the WTO message beginning with "ADADEC".

Abend Code	Description
U045	ADAESI (ADABAS External Security Interface) has rejected ADAREORG as not allowed to execute. Make the necessary definitions to ADAESI, then rerun the job.

Processing Messages

The processing of ADACMP cards by ADAREORG results in a printout of every card image. An incorrect card will be followed by the message:

```
*** ERROR IN ADACMP CARDS
```

If an occurrence count is specified on a PE or MU, a warning message is printed as follows:

```
*** WARNING - PE/MU COUNT FOUND
```

This is a potentially important warning and should be heeded! It alerts the user to the fact that the count will be ignored by ADAREORG but not by ADACMP - therefore if the actual number of occurrences exceeds that specified, the data will be corrupted.

Codeword errors

These will be output in the joblog in the following format: (followed by a U000 abend message):

```
ADAREORG: XXXXXXXXXX.........
```

- where XXXXXXXXXX is one of the following:

```
INVALID CODEWORD SUPPLIED
INCORRECT PARMCARD SUPPLIED
NO ADAVER SPECIFIED!
CODEWORD DOES NOT MATCH THIS PRODUCT
SOFTWARE LICENCE EXPIRES IN NN DAYS
SOFTWARE LICENCE HAS EXPIRED
SOFTWARE LICENCE IS NOT FOR THIS CPU
```

DDPARM errors

Processing of DDPARM input also results in a printout of every card image. An incorrect card will be followed by the message:

```
*** LAST PARAM CARD IN ERROR - xxxxxxxx
```

- where xxxxxxxx will be one of the following:

```
LEFT BRACKET EXPECTED
COMMA EXPECTED
RIGHT BRACKET EXPECTED
EQUALS EXPECTED
BAD FIELD OR FUNCTION
NO FUNCTION DATA
EXTRANEOUS DATA
BAD FILE NUMBER
FIELD NOT FOUND
BAD START BYTE
BAD START/END BYTE
TOO MANY LETS FOR THIS FIELD
IMPROPER OPERATOR
ODD LENGTH HEX
BAD HEX
TOO MANY ACC/REJ CARDS
CARD END EXPECTED
BAD FIELD
DUPLICATE KEY
UX0=, 1=, 2=, 3= MODULE NAME EXPECTED (1-8 CHS)
USER EXIT UNLOADABLE
DUPLICATE DEFINE UX
UX2, 3 EXPECTED
BAD FIELD NAME
USER EXIT UNDEFINED
BAD FILE NUMBER
TOO MANY ACC/REJ/APP CARDS
```

The following message may appear after an ACCEPT or REJECT involving a PE or MU field:

```
*** WARNING - ONLY 1ST OCCURRENCE USED
```

At the end of DDPARM input, errors may be detected which pertain to input card(s) other than the immediately preceding one. The following messages may appear:

```
*** KEY CARD IN ERROR - xxxxxxxx
```

- where xxxxxxxx will be one of the following:

```
KEY MISSING OR REDUNDANT
KEY FIELD NOT FOUND
```

Chapter 7

User Exits

This section describes the user exit facilities provided by ADAREORG. All user exits are field-level based (except for UX0 initialization) and - depending upon the processing being done in the user exit - may add significantly to processing resource requirements.

User exits must be defined to ADAREORG via the DEFINE statement described in the Parameters chapter. There are four user exits available (named UX0 through UX3). Each user exit can be defined only once and the load module must be available to ADAREORG through a STEPLIB in the execution JCL. Note, however, that User Exit 3 may specify 27 different exits (one for each letter of the alphabet, and one with no letter - see page 21 for details.

Processing

Each user exit is given control in 24-bit mode with a standard parameter list and is not required to be reentrant. The parameters are described in the supplied macro REOMACRO, which can be found in the distributed source library together with sample user exits.

User Exit 0 - Initialization

UX0 is the initialization exit. It can be used to establish special data areas for other user exits by returning their addresses as a token to ADAREORG - which then passes them to all other active user exits. UX0 will only be called once (and if it is not supplied, the other user exits will receive a zero-valued token).

User Exit 1 - Key Matching

UX1 is the key-matching exit, and it is only appropriate if there are two input files. It completely replaces the standard key-matching process if used. Depending on the specific action of the User Exit, UX1 will likely be called approximately as many times as there are records in one of the input files.

User Exit 2 - Selection/ Modification

UX2 is used to select and modify records according to supplied parameters. One or more APPLY UX2 cards are used to tell ADAREORG which field values are to be given to the exit at each call. UX2 will be called **n*m** times, where **n** is the number of output records and **m** is the number of fields examined per record (ie. the number of APPLY cards).

Each call to UX2 can either:

Accept the record unconditionally;
Reject the record unconditionally;
or
Allow ACCEPT/REJECT cards to determine the outcome.

Note - all fields in APPLY UX2 cards will be validated by UX2 before any ACCEPT/REJECT cards are processed. As the FXDATA field can point to variable fields as well, note that the first 1 or 2 bytes could be length fields (depending on the field type). Another possible use for UX2 is to modify the value in the input data field to affect the ultimate output value, although this can usually be done in User Exit 3.

User Exit 3 - Output Format.

UX3 is used to modify the data content of output fields. One or more APPLY UX3 cards will tell ADAREORG which field values are to be passed to the exit at each call. UX3 will be called **n*m** times, where **n** is the number of output records and **m** is the number of fields examined per record (i.e. the number of APPLY UX3 cards supplied).

Each call to UX3 is made after the field value has been generated. UX3 then inspects and reformats the output values if needed. It is not possible to change the length of a field at this stage, but the data can be changed without limitation of any kind - even to illegal values.

The parameter interface to this exit includes the field name, type and length - enabling individual fields to be processed case by case.

Sample exits are provided in the source library (included on the distribution tape). They include exits to:

replace blank by zero;
convert date to Natural D format;
convert date from Natural D format;
convert from EBCDIC to ASCII.

Chapter

8

Installation

Release Media Contents

The release media will generally contain two PC zip files (unless specified differently in the release notes). They will have names like:

ARV232.INSTALL - containing source code and JCL
ARV232.LOAD - containing load modules.

Install library

The first dataset is the ADAREORG source library, containing the following members:

```
JCLTESTx    Executes ADAREORG in stand-alone mode

JCLxxx            Executes ADAREORG in ADACMP Exit Mode

NATFDT            Sample program written in NATURAL.

DOCxxx            Describes release history & some extra
documentation.

Uxxxx       Example user exits.
```

- where "x" is an identifying character string. The source library contains sample JCL to run ADAREORG in both **Exit** and **Stand-alone** modes, the NATURAL source for a program to extract the FDT of a file (NATFDT), sample user exits (UXn), and some additional documentation

The NATURAL program NATFDT can be loaded into the DBA NATURAL application. It runs in batch mode to generate ADACMP cards directly from the FDT. The fields generated do not contain super or sub-descriptor entries, although it flags their presence as comments.

Load library

The second Dataset contains the load modules for ADAREORG. As ARV222.LOAD only contains a few members, it is suggested that they be moved to the DBA utilities load library - thus preventing the creation of an additional load library to manage.

Note: When moving ADAREORG modules to another load library, remember to check block sizes. If the blocksizes of the two libraries are different, it is recommended to move the modules using the linkage editor.

Unloading Release Tape

Allocate two partitioned datasets for the source and executable members respectively, prefixing the tape dataset names with a suitable high-level qualifier (such as DBA). Use your standard blocksizes for source and load libraries. As shown in the JCL, one cylinder of space will be ample.

Release notes supplied with the tape will detail the volume/serial number and any changes made to the tape content.

Unload JCL

To unload the tape, use JCL similar to the following:

```
//STEP1    EXEC  PGM=IEBCOPY (JSECOPY for Fujitsu sites)
//INFILE1  DD    DSN=ARVnnnz.INSTALL,UNIT=TAPE,DISP=OLD,
//         LABEL=(1,SL),VOL=(,RETAIN,SER=xxxxxx)
//OFILE1   DD    DSN=XXX.ARVnnnz.INSTALL,
//         DISP=(NEW,CATLG,DELETE),UNIT=SYSDA,
//         SPACE=(CYL,(1,1,15)),VOL=SER=yyyyyy
//INFILE2  DD    DSN=ARVnnnz.LOAD,UNIT=AFF,DISP=OLD,
//         LABEL=(2,SL),VOL=SER=xxxxxx
//OFILE2   DD    DSN=XXX.ARVnnnz.LOAD,
//         DISP=(NEW,CATLG,DELETE),UNIT=SYSDA,
//         SPACE=(TRK,(10,2,15),RLSE),VOL=SER=yyyyyy
//SYSPRINT DD    SYSOUT=*
//SYSIN    DD  *
   COPY I=INFILE1,O=OFILE1
   COPY I=INFILE2,O=OFILE2
/*
```

- where:

xxxxxx is the volume/serial number of the tape as supplied by your distributor.

yyyyyy is the volume/serial number of the DASD device where the release datasets are to be located.

nnnz where **nnn** is the version number and **z** is the fix number.

Index

R

S

T

U

V