

Version

**3.05**

Treehouse Software, Inc.

# ADAREORG User Guide



A Product of CCA Software Pty Ltd

## **Copyright Notice**

Copyright 1996 - 2015 CCA Software Pty Ltd.

All Rights Reserved.

## **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.

## **Requirements for Confidentiality**

This document contains trade secrets and proprietary information of CCA Software Pty. Ltd. Reproduction and/or modification of this document without the prior written approval of CCA Software Pty. Ltd. is prohibited. Use of this document is limited to licensed users of ADAREORG or those with specific written permission of CCA Software Pty. Ltd.

THE SOFTWARE WHICH IS DESCRIBED IN THIS DOCUMENT IS SUBJECT TO LIMITATIONS ON USE, RELEASE, DISCLOSURE AND DUPLICATION, AND TO REQUIREMENTS FOR CONFIDENTIALITY, PROTECTION AND SECURITY WHICH ARE SET OUT IN THE SOFTWARE LICENSE AND MAINTENANCE AGREEMENTS.

**Treehouse Software, Inc.**  
**2605 Nicholson Road, Suite 1230**  
**Sewickley, PA 15143 USA**  
**Phone: 724.759.7070**  
**Fax: 724.759.7067**  
**e-mail: [tsi@treehouse.com](mailto:tsi@treehouse.com)**  
**<http://www.treehouse.com>**

Version 3.05a, April 2015

# Table of Contents

<b>INTRODUCTION.....</b>	<b>1</b>	PROCESSING .....	355
OVERVIEW.....	1	USER EXIT 0 - INITIALIZATION .....	356
SCOPE .....	1	USER EXIT 1 - KEY MATCHING .....	356
EFFICIENCY .....	2	USER EXIT 2 - SELECTION/ MODIFICATION .....	357
ADABAS VERSION COMPATIBILITY .....	2	USER EXIT 3 - OUTPUT FORMAT.....	367
ADAREORG VERSION.....	2	<b>INSTALLATION.....</b>	<b>377</b>
<b>PROCESSING OVERVIEW .....</b>	<b>3</b>	RELEASE CONTENTS .....	377
INPUT .....	3	<i>Installation Procedure Overview....</i>	378
EXECUTION MODES .....	4	<i>Procedure to Install ADAREORG for</i>	
<b>OPERATIONS .....</b>	<b>7</b>	<i>MSP/EX - only.....</i>	40
OPERATION JCL .....	7	<i>Install the Source Library .....</i>	40
ADACMP EXIT MODE.....	7	<i>Install the Load Library.....</i>	40
STAND-ALONE MODE.....	7	INSTALLATION FROM 3480/3490	
DESCRIPTION OF DDNAMES .....	8	CARTRIDGE NO LONGER AVAILABLE. .	41
OUTPUT.....	11	APPLY PRODUCT PROTECTION CODE....	41
<i>Output files .....</i>	11	<i>Contents of the Install library.....</i>	42
<i>End of processing messages.....</i>	11	<i>Load library .....</i>	42
<b>PARAMETERS.....</b>	<b>13</b>	<b>INDEX.....</b>	<b>42</b>
FILE DEFINITIONS .....	13		
<i>General Syntax Rules .....</i>	13		
<i>USERISN.....</i>	14		
<i>Generating Input File Cards.....</i>	15		
CONTROL & PROCESSING PARAMETERS	16		
<i>General Syntax Rules .....</i>	16		
<i>Control Parameters.....</i>	17		
<i>Processing Parameters.....</i>	19		
<b>FIELD FORMAT CONVERSIONS..</b>	<b>254</b>		
LENGTH CONVERSION .....	254		
DATA TYPE CONVERSION.....	254		
<i>From ALPHA (type A).....</i>	266		
<i>From UNPACKED (type U).....</i>	266		
<i>From BINARY (type B).....</i>	266		
<i>From FULLWORD (type F).....</i>	266		
<i>From PACKED (type P).....</i>	266		
<i>From Floating Point (type G).....</i>	267		
<i>Long Alphanumeric fields (type LA)</i>	278		
<b>ERROR MESSAGES.....</b>	<b>299</b>		
FAULT DIAGNOSIS .....	299		
ADABAS MESSAGES .....	299		
SYSTEM ABENDS .....	30		
USER ABENDS .....	31		
PROCESSING MESSAGES .....	322		
<i>Codeword errors.....</i>	323		
<i>DDPARM errors .....</i>	323		
<b>USER EXITS.....</b>	<b>355</b>		



## **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.

## **INTRODUCTION**

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 V3.02 supports ADABAS V8.1 and V8.2 only, the installer must specify ADAVER=8 otherwise there will be an abnormal termination. A future release of ADAREORG V3.XX will support V6.x, V7.1, V7.4, V8.1 and V8.2.+ databases. All ADABAS field types are supported, with the exception of BLOBs and some of the special features in ADABAS 8.22. These will be supported in a future upgrade.

### **ADAREORG Version**

This manual is specific to **ADAREORG Version 3.02** and subsequent fix levels, which is Year 2000 compliant. The ADAREORG version number is printed on all ADAREORG Execution Report page headers.

## 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.

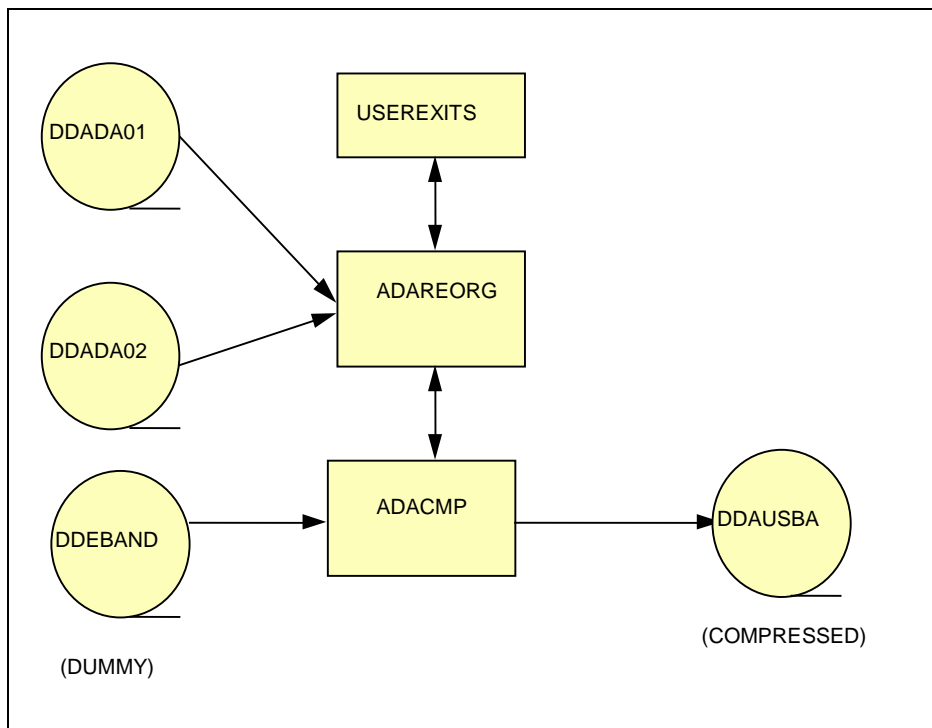


Figure 1: ADACMP exit mode - compressed output.



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.

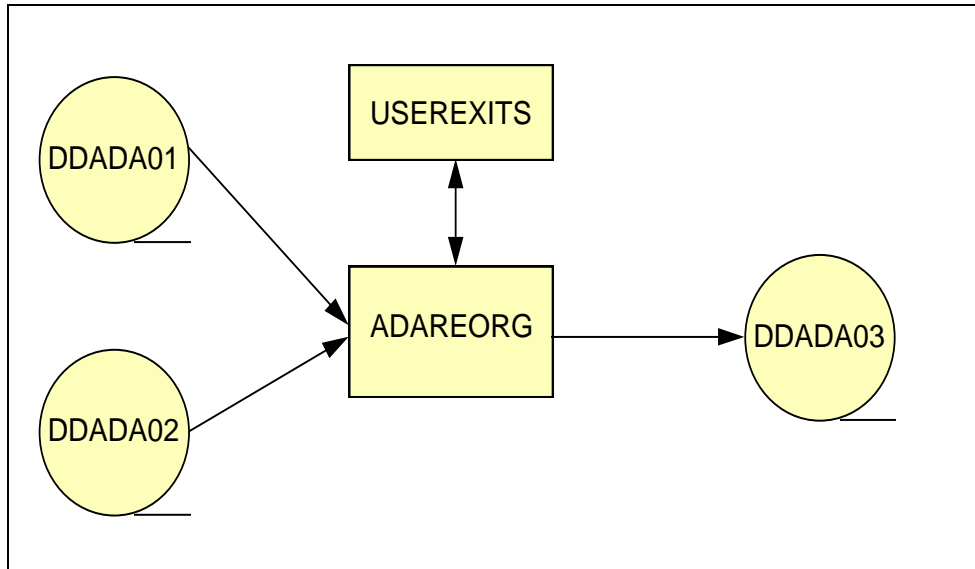


Figure 2 - Stand-alone mode - decompressed output.



## 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	<p>The sole or first input file. Depending on the type of file, the following considerations apply:</p> <ol style="list-style-type: none"> <li>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 <u>full</u> unload be taken (i.e. without the MODE=SHORT option) so the unloaded file can double as a backup of the ADABAS file.</li> <li>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.</li> <li>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.</li> </ol>
DDADA02	Optional	<p><b>Not supported V302.</b> 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.</p>

DDNAME	Usage	Description
		<p>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).</p>
DDADA03	Conditional	<p>This file is produced only if ADAREORG is run in <u>stand-alone</u> 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.</p> <p><b>Note:</b> LA output fields default to 16381 bytes when empty, thus output block sizes should be appropriately sized.</p> <p>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.</p>
DDWAN01	Conditional	<p>Contains the ADACMP cards describing DDADA01. <u>Not</u> needed when DDADA01 is a compressed unloaded file (as ADAREORG can derive the file description from the input).</p> <p>It may consist of in-stream data, a sequential file or a PDS member. A record length of 80 bytes is assumed.</p>
DDWAN02	Conditional	<p>Required only if DDADA02 is present., and subject to the same considerations as for DDWAN01 above.</p>

**OPERATIONS**

<b>DDNAME</b>	<b>Usage</b>	<b>Description</b>
<b>DDWAN03</b>	Mandatory	<p>Contains ADACMP cards describing the reorganized output file built in DDADA03 (or in DDAUSBA if running as an ADACMP exit).</p> <p>When running as an exit, DDWAN03 would normally be the same as DDKARTE - in fact both DD statements can point to the same physical dataset.</p>
<b>DDPARM</b>	Optional	<p>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.</p>
<b>DDCARD</b>	Conditional	<p>Must be present when running in exit mode only (i.e. with "EXIT" as a parameter in DDPARM). This card must contain:</p> <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <p>ADARUN PROGRAM=ADACMP, UEX6=ADAREORG</p> </div>
<b>DDKARTE</b>	Conditional	<p>ADACMP cards - needed in <b>exit mode</b> only. Normally identical to DDWAN03.</p>
<b>DDEBAND</b>	Conditional	<p>Needed in <b>exit mode</b> only. Must be set to DUMMY.</p>
<b>DDAUSBA</b>	Conditional	<p>Needed in <b>exit mode</b> 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.</p>
<b>DDDRUCK</b>	Conditional	<p>Needed in <b>exit mode</b> only. Contains ADACMP messages. Usually specify as SYSOUT=*</p>

DDNAME	Usage	Description
DDFEHL	Conditional	Needed in <b>exit mode</b> 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 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.





## 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" and "HEADER" are the only ADACMP keywords recognized by ADAREORG. If other keywords are supplied, they will be ignored.

Example:

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

<b>Super Descriptors</b>	Super/Sub/Phonetic-descriptor field definitions can be present but are ignored by ADAREORG.
<b>Comments</b>	Bytes 72-80 are ignored. Bytes 46 to 72 may be used for comments depending on the rules for the relevant utility.
<b>Level Numbers</b>	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.
<b>PE/MU fields</b>	PE fields are accepted. Members of a PE must have levels greater than 1. Note that PE or MU occurrence counts should <b>NOT</b> 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.

## PARAMETERS

<b>Standard Length</b>	This must be present. ADAREORG does NOW support undefined lengths, ie. 0.
<b>Standard Format</b>	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.
<b>Definition Options</b>	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 ADAREORG 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 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, 7 or 8, meaning ADABAS V5, V6, V7, V8.

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 V302, default value is 8 for ADABAS V8.**

**CODE**    CODE=XXXXXXXXXXXXXXXXXXXXXX

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

## PARAMETERS

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)
```

## PARAMETERS

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 UXna(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.

## PARAMETERS

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
```

## MODE

The MODE parameter currently only takes a single sub-parameter CONTINUE, which may be abbreviated to CONT.

EG.

```
MODE CONT.
```

When MODE CONT is used, a message will be outputted to DDNAME DDFAILX for each failed input record. The number of failed records will be included in the statistics at the end of the ADAREORG run, and ADAREORG will finish with RC=4. DDFAILX should not have any LRECL or BLKSIZE DCB information defined. If MODE is not supplied, ADAREORG will simply terminate as usual, upon receiving faulty input.

## 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**



## 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.

## ERROR MESSAGES

### **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 there from. 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.

## **ERROR MESSAGES**

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.

### **LOB Field Support**

Actual LOB files (as opposed to LOB base files), are not modified by ADAREORG. LB field references in LOB base files are passed through to the output. Since SAG have implemented LOBs as dual files, deleting LOB base file records in AR can result in the actual LOB file getting out of sync with the LOB base file. Both AR and Adabas will still work, but the consequence is that the LOB file will contain some objects that are no longer referred to anywhere. I.e. the LOB file will be bloated.

Adabas will not allow pairing of an existing LOB file with a new LOB base file that has a different number to the original LOB base file. The implication is that either the same file number needs to be used for the new LOB base file as for the old one, or the FCB needs to be modified to synchronize it with the LOB file (see ADADBS MODFCB in the Adabas utilities manual for how to modify the FCB).

AR is not able to extract LOB values directly from the LOB file, as it has no access to it. However it is possible to use ADACMP to decompress a LOB base file using LOBVALUES=YES, then feed this decompressed file into ADAREORG (do not use the EXPAND parameter in ADAREORG).

Note however that AR does not yet support input files decompressed with HEADER=YES, so record lengths that would exceed the block size are not supported, and some LB fields may very well be too long.

### **System field support**

System fields are normal fields where the content is supplied by Adabas itself rather than programmatically. Hence minimal special support is required for these fields.

### **Logically deleted field support:-**

Logically deleted fields do not show up in ADAREORG at all. It is as if they really don't exist. The content is not accessible, and hence also not available in the output.



## 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.

## ERROR MESSAGES

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:

<b>Abend Code</b>	<b>Description</b>
<b>U001</b>	failure attempting to print the first page of output (by module REODOCP).
<b>U002</b>	error detected while processing DDWAN01. This will be accompanied by other error messages. It could be caused by being unable to open the file.
<b>U003</b>	error detected while processing DDWAN02. This will be accompanied by other error messages.
<b>U004</b>	error detected while processing DDWAN03. This will be accompanied by other error messages.
<b>U005</b>	error detected while processing DDPARM. This will be accompanied by other error messages.
<b>U006</b>	failure to open DDADA01.
<b>U007</b>	failure to open DDADA02 (where DDWAN02 was present).
<b>U008</b>	this message code no longer used.
<b>U009</b>	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).
<b>U010</b>	failure while processing a DDADA02 record against the internal record representation of data.
<b>U011</b>	failure to open DDADA03.
<b>U012</b>	failure while building a record for DDADA03.
<b>U013</b>	failure processing FDT for DDADA01. Indicates that the respective FDT failed to correctly decompress.
<b>U014</b>	failure processing FDT for DDADA02. Indicates that the respective FDT failed to correctly decompress.
<b>U015</b>	decompression of data failed on DDADA01. See also the WTO message beginning with "ADADEC".
<b>U016</b>	decompression of data failed on DDADA02. See also the WTO message beginning with "ADADEC".

## ERROR MESSAGES

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: XXXXXXXXXXXX.....
```

- where XXXXXXXXXXXX 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 - xxxxxxxxx

- 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  
 EXTRANEIOUS 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 - xxxxxxxxx

- where xxxxxxxx will be one of the following:

KEY MISSING OR REDUNDANT  
 KEY FIELD NOT FOUND



## 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).

## USER EXITS

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.



## Installation

### Release Contents

The release consists of one compressed zip file:

- **ARvxxxxy-release.zip**

**Where: AR** – internal code for ADAREORG, **xxx** is the version and **y=** is a single character defining the fix level, so **ARv300a** is version **300** fix level **a**.

Note: For a given version of ADAREORG [vXXX] the JCL/Install library and also the Users Guide will generally not change between specific fix levels.

### Installation Procedure Overview

1. Save the email attachment and/or PC file to disk.
2. Unzip the supplied release libraries to a directory on a PC. There should be six files in total:
  - readme.txt; a text file containing last minute information on the release;
  - ARv305-UsersGuide.pdf – The ADAREORG Users Guide;
  - ARv305-Release-Notes.pdf – The ADAREORG Release Notes;
  - ARv305IN.TRS – ADAREORG JCL, samples and Macro Library – EBCDIC TERSED compressed – z/OS only;
  - ARv305L.TRS – ADAREORG Load library – EBCDIC TERSED compressed z/OS only – will not appear in MSP/EX release.
3. Allocate two temporary files on the mainframe for the upload files, details in following section.
4. Load the two mainframe files (ARv305N, ARv305L) to the mainframe- using FTP or whatever file transfer mechanism is used onsite – binary mode only, without any translation as the files are already in EBCDIC format.
5. Unpack the datasets using the information in the following sections, Note this procedure will be different for MSP/EX and OS390 [z/OS] users.

## Procedure to Decompress the ADAREORG Install JCL and Load Libraries – z/OS only

When decompressing the release zip file do not change file extensions on the PC platform, this can lead to problems with CR/LF on binary files.

1. On the mainframe pre-allocate two datasets as follows:  
 ‘&SYSUID..AR305N.TRS’ 2 cyls, ‘&SYSUID..AR305L.TRS’ 2 cyls, (both INSTALL and LOAD datasets have a DCB=(LRECL=1024, RECFM=FB, DSORG=PS). The BLKSIZE should be some sensible multiple of 1024, such that the blocks fits on a track. Where &SYSUID. is just your userid on the mainframe.
2. Transfer the AR305N.TRS & AR305L.TRS files to the mainframe. Use a binary FTP or file transfer, the files must be loaded without using ASCII to EBCDIC translation or CR/LF and with the LRECL and BLKSIZE as specified. If you pre-allocate the datasets as specified here above, then the FTP server on the mainframe should just overwrite them, keeping the attributes as specified. This considerably simplifies the FTP transfer, as the only thing you need specify in the FTP client is “bin” to ensure a binary transfer. This is also the reason why the mainframe files should be named as specified.
3. On the mainframe run the following job, where the names of the INFILE files should remain the same, but the names of the OUTFILE files may be changed to meet site standards. TRSMAIN is available free from IBM. Once the TERSE job has been run, and the INSTALL & LOAD libraries exist, the INFILE files may be deleted.
- 4.

```

//XXXXXTRS JOB
MSGLEVEL=(1,1),CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//UNTERSE EXEC PGM=TRSMAIN,PARM='UNPACK'
//SYSPRINT DD SYSOUT=*
//INFILE DD DISP=SHR,DSN=&SYSUID..AR305.TERSED
//OUTFILE DD
DISP=(,CATLG),SPACE=(CYL,(2,2,45),RLSE),UNIT=SYSDA,
// DSN=&SYSUID..ADAREORG.V305.LOAD
//UNTERSE EXEC PGM=TRSMAIN,PARM='UNPACK'
//SYSPRINT DD SYSOUT=*
//INFILE DD DISP=SHR,DSN=&SYSUID..AS509SIN.TERSED
//OUTFILE DD
DISP=(,CATLG),SPACE=(CYL,(2,2,45),RLSE),UNIT=SYSDA,
// DSN=&SYSUID..ADAREOG.V305.INSTALL

```

The INSTALL dataset should contain approximately 30 members and the LOAD library approximately 70 members.

At this point both the JCL and Load libraries will have been populated and the members are ready for tailoring and testing. There is no longer a requirement to run the build step to create the load modules under z/OS [the decompress process has already created the load modules].

Once the above JCL/Source and Load Libraries are transferred to the mainframe and decompressed the installer will be in a position to undertake a series of test runs of ADAREORG V3.05 to ensure correct installation.

**xxxxxx.AR305.INSTALL** - library contains JCL, example user-exits, macros and ADASTRIP execution jobs.

**xxxxxx.AR305.LOAD** - binary executable ADASTRIP application

The supplied example JCL must be modified to conform to local site standards, making appropriate changes to the dataset names in these members to match the ones locally at the site, especially the JOBLIB/STEPLIB card to point to the V50 Load Library.

### **Procedure to Install ADAREORG for MSP/EX - only**

On MSP/EX, prior to FTPing the release files to mainframe disk, undertake the following steps:

#### **Install the Source Library**

Unzip the file ASxxxxyin.zip and upload the unzipped \*.txt files into a previously created mainframe dataset called [hlq].ARvxxx.INSTALL where [hlq] is your high-level DSN qualifier.

Dataset Attributes:

Organization:	<b>PO</b>
Record Format:	<b>FB</b>
Record Length:	<b>80</b>
Blocksize:	<b>Multiple of 80</b>

#### **Install the Load Library**

FTP the file ARxxxxyob.jcl [where xxx is the supplied version and y is fix level, eg v302] to the install library. The load library should have the following attributes:

Dataset Attributes:

## INSTALLATION

Organization: **PO**  
Record Format: **U**  
Record Length: **0**  
Blocksize: **Your choice, suggest same as ADABAS**

This member contains JCL to produce the load library members and will need to be modified to your library name. So, please modify both the SYSLIB and SYSLMOD DD statements to point to this library. You will also need to review the JOBCARD and modify it to meet your local site standards before running the job.

**PLEASE NOTE THE JCL TO EXTRACT THE LOAD MODULES CONTAINS A WARNING THAT YOU SHOULD READ BEFORE EXECUTION.**

**NOTE: This file [ARxxxjob.jcl] is in EBCDIC form and needs to be loaded as a BINARY file NOT as a .txt file, that is WITHOUT ASCII → EBCDIC TRANSLATION. You should NOT, REPEAT NOT, use the ASCII CRLF option for the upload of this file.**

Once the above Source and Load Libraries are in position and populated you should be in a position to run ADAREORG.

The install library contains example JCL, parameters and user exits. Please ensure you modify these to conform to your local standards making appropriate changes to the dataset names in these members to match the ones at your site, especially the JOBLIB/STEPLIB card to point to the ADAREORG Load Library.

### Apply Product Protection Code

ADAREORG requires a Product Protection Code, the codeword is at least 20-bytes in length and will need to be supplied so that ADAREORG will run on your system.

A code provided previously, which hasn't expired, may still work with this new release **BUT** it is recommend that all customers request new product codes from their local support representative.

The code is supplied to ADAREORG as part of the ADAREORG EXEC card as follows:

```
//STRIP61 EXEC PGM=STRIP, PARM='12345678901234567890'
```

**OR**

The code may be permanently zapped into the ADAREORG load modules, this zap must be created by CCA and takes the place of the CODE parameter.

An example only, of this zap is supplied in the install dataset. In order to run ADAREORG, you will either need a codeword or zap supplied by CCA.

Attempting to apply the codeword zap to a previously zapped load module will fail, to get around this simply comment out the VER's OR apply the zap to a fresh copy of the load module. It is recommended that all original install libraries be backed up before applying any zaps.

## Contents of the Install library

The first dataset is the ADAREORG source library, containing as a minimum 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. It is recommended that a backup copy of all install files be taken, particularly the load library prior to applying any zap to the load modules for example the codeword.

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

# Index

## A

ABENDS, 28  
ACCEPT parameter, 18  
ADABAS  
    LF commands, 15  
    messages, 27  
    version support, 2  
ADACMP  
    cards, 15, 39  
    exit mode, 4  
ADAREORG  
    messages, 30  
    version, 2  
ADAREP, 15  
APPLY parameter, 20

## C

Codeword error messages, 30  
comments, 13  
control parameters, 17  
conversion  
    from Alpha, 24  
    from Binary, 24  
    from Floating Point, 24  
    from Fullword, 24  
    from Packed, 24  
    from Unpacked, 24  
    Long Alphanumeric fields, 25  
    **periodicity**, 25  
    Wide Alphanumeric fields, 25

## D

data type conversion, 23  
DDADA01, 8  
DDADA02, 9  
DDADA03, 9  
DDAUSBA, 11  
DDCARD, 10  
DDDRUCK, 11  
DDEBAND, 11  
DDFEHL, 11  
DDKARTE, 10  
DDNAMES, 8  
DDPARM, 10  
DDPRINT, 8  
DDPRINTX, 11  
DDWAN01, 10  
DDWAN02, 10  
DDWAN03, 10  
DEFINE parameter, 22

## E

error messages  
    Codeword, 30  
    DDPARM, 31  
    processing, 30  
execution modes, 4, 7  
EXIT mode, 4, 11, 27

## F

FXDATA field, 34

## I

INC, 17  
ISN, 14, 15, 27

## J

JCL, 7  
    exit mode operation, 7  
    stand-alone mode operation, 8

## L

LA field, 9, 25  
length conversion, 23  
LET parameter, 21  
level numbers, 13  
LIMIT parameter, 22

## M

messages, 27  
modes of execution, 4

## N

NATFDT, 15, 39

## O

output files, 12

## P

parameters  
    ACCEPT, 18  
    APPLY, 20  
    DEFINE, 22  
    LET, 21  
    LIMIT, 22  
    REJECT, 20  
PE/MU, 13, 25, 27, 30

## **I N S T A L L A T I O N**

periodicity, 25  
PREDICT, 15  
processing parameters, 16, 18

### **R**

REJECT parameter, 20  
release contents, 35

### **S**

stand-alone mode, 5  
standard format, 14  
standard length, 14  
superdescriptors, 13  
syntax rules

ADACMP, 13  
ADAREORG, 16

### **T**

tape dataset, 39

### **U**

User Exits, 33  
USERISN, 9, 13, 14  
UTIL03, 15

### **V**

variable fields, 18, 21