Version 3

CCA Software Pty Ltd

DBAUDIT Users Guide

Copyright Notice

Trademark Acknowledgments

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

MVS/XA, MVS/ESA, OS390 and DFSORT are trademarks of IBM Corporation

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 DBAUDIT or those given specific written permission by 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 LICENCE AND MAINTENANCE AGREEMENTS.

CCA Software Pty Ltd
PO Box 423,
Blackburn, Vic. 3130
Australia

Phone: +61-3-9894 0055
Fax: +61-3-9894 0039

email:info@ccasoftware.com.au
web: http://www.ccasoftware.com.au/

Version 4.00, September 2015

Table of Contents

Chapter 1

Introduction

DBAUDIT validates the logical integrity of an ADABAS database. It takes as input an ADASAV backup and user supplied verification rules and produces output files reporting integrity errors.

Most ADABAS databases consist of scores of files, hundreds of keys, thousands of fields and millions of records, operated on by thousands of programs written by many people to different standards and specifications over many years. Users and application systems assume their database contains implicit structures and relationships between records. For example:

- **purchase order lines are assumed to "belong" to a purchase order header;**
- **the supplier number on a purchase order is assumed to be a pointer (or "foreign key") to a record on a supplier file.**

Additionally, records are assumed to conform to certain rules, and fields are assumed to contain "valid" data.

Complex Systems

However, the realities of complex systems usually result in the application level structures of the database containing errors, although the physical integrity of the database may be intact.

Such problems can be the result of many different causes including;

- **unpredictable user actions,**
- **unforeseen processing circumstances (such as batch jobs being canceled),**
- **unforeseen data conditions,**
- **changing specifications during application development and maintenance,**

incomplete testing and, occasionally, physical integrity lapses by the data base management system.

The consequences of undetected application level inconsistencies include unpredictable ("chaotic") behavior of the application system, cascading propagation of errors, a loss of faith in the system by its users, management and auditors, and wasted effort by users and programmers to manually detect and correct the errors.

The Tool

DBAUDIT provides the DBA with a tool to detect and isolate application data corruption. Verification rules can specify inter-file ("referential integrity"), intra-file and intra-record criteria. Output from DBAUDIT can be easily used as input to application specific verification, analysis and exception reporting procedures.

DBAUDIT reads an ADASAV backup to obtain a static and consistent picture of the database. If the ADASAV backup was produced against an active ADABAS nucleus, the protection log will also be read by DBAUDIT. Users of the QDUMP incremental backup product can supply the latest QDUMP backup dataset along with the latest ADASAV to be read by DBAUDIT.

DBAUDIT makes just 1 pass of the ADASAV backup, extracting and sorting data as required to perform the specified validations. DBAUDIT will read the ADASAV backup volumes in parallel for minimum elapsed time and maximum throughput. It contains algorithms to optimize input processing and disregard entire or partial volumes that don't contain relevant information.

Validation Procedures

The number of validations that can be performed concurrently is dependent on the number of sort operations that can be initiated within the DBAUDIT region, which will be dependent on region size, operating system configuration and sort options.

The validation procedure is based on "SELECT" statements, which specify a file number and an optional "WHERE" clause. Each SELECT statement defines the set of data on which the following "RULE", "PROCESS" and "JOIN" statements operate. The results of validations can be directed to any number of user specifiable output files. Any number of SELECT, RULE, PROCESS and JOIN statements may be supplied. A file may be "SELECTed" any number of times.

The RULE, PROCESS and JOIN statements specify different types of validations.

The RULE statement is used to specify inter-file, i.e. referential integrity rules. The PROCESS statement is used to specify intra-record validations and processes. JOIN

statement is used to bring together and list data from two separate records, based on a common primary and foreign key.

In addition to its referential integrity and data validation capabilities, DBAUDIT can be used to extract data from the ADASAV backup, optionally normalizing the data in a format suitable for subsequent sequential processing or loading into a relational database.

DBAUDIT can extract data from the ADASAV backup over an order of magnitude faster than can be achieved by NATURAL, COBOL or assembler programs using the ADABAS call interface. DBAUDIT facilitates the production of control totals, counts, and simple reports that can be used "as is" or fed into application specific processing.

Cost of Implementation

The cost of implementing these functions without DBAUDIT is so high that they have been neglected. However, the "hidden" costs of inconsistent and incomplete data can be much higher. DBAUDIT makes the verification of application data possible because:

its simple yet powerful command language lets DBAs, programmers or application controllers familiar with the file structure and processing rules of the application specify the verification rules and processes without needing to write programs;

DBAUDIT runs against a backup of the database:

providing a consistent image (unhindered by ADABAS's lack of read integrity);

eliminating contention for the database disk drives whilst verifying;

DBAUDIT processes all requests and input volumes in parallel, making just one pass of the backup and only reading the relevant backup volumes (i.e., those that contain data to be analyzed);

DBAUDIT does not use ADABAS calls to locate and decompress data, but instead maps, decompresses and processes data using extremely efficient I/O, parallel I/O and manipulation algorithms which minimize both CPU and elapsed time.

Chapter 2

INSTALLATION

With the introduction of V3.00, all DBAUDIT Software from CCA Software is now supplied in one of two formats:

Compressed 'zip' file as an email attachment,

Release files on a CD, at an extra cost.

The use of email attachments and/or CD facilitates fast transmission of new releases, upgrades and fixes. All fix levels are provided as upgraded binary modules as email attachments.

Installation from E-MAIL Attachment or CD

The general installation file consists of one compressed folder (or zip file) of the form:

DBvYNNx-release.zip

Where:

YNN is the current version and release level

x is the fix level of the release

In this release you will be delivered DBv300b-release.zip.

Inside this zip file you will find the following files:

Readme.txt – a text file containing last minute information about the release

DBv300y-Release-Notes.pdf – release notes for this fix level in PDF format

DBv300-Users-Guide.pdf – The Users Guide in PDF format

DBv300in.cmp - Source/JCL/example User exits – binary compressed EBCDIC format file.

DBv300yL.cmp - Load Library – binary compressed EBCDIC format.

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 and ASCII files.

Installation process

Pre-allocate two datasets as follows:

'XXXXX.DBV300.INSTALL.CMP' 2 cyls,

'XXXXX.DBV300y.LOAD.CMP' 2 cyls,

Both INSTALL and LOAD datasets have a DCB=(LRECL=80, BLKSIZE=3120, RECFM=FB, DSORG=PS);

Load the CNTL & LOAD mainframe files to disk, use a binary FTP or file transfer, the files must be loaded without using ASCII to EBCDIC translation or CR/LF and the LRECL and BLKSIZE as specified above.

On the mainframe allocate two datasets:

'XXXXX.DBV300.INSTALL' approx 2 cyls
DCB=(LRECL=80,BLKSIZE=3120,RECFM=FB,DSORG=PO)

'XXXXX.DBV300y.LOAD', approx 3 cyls
DCB=(LRECL=0,BLKSIZE=<same as ADABAS>,RECFM=U,DSORG=PO)

Go to TSO native mode (option 6 in ISPF), TYPE
RECEIVE INDSN('XXXXX.DBV300.INSTALL.CMP') <enter>

When prompted the following additional parameters may be used to define the output INSTALL dataset, DATASET('XXXXX.DBV300.INSTALL') SHR, this should extract approximately 30 members.

Repeat Steps above but this time using the LOAD library as input and output to the correct dataset. This step should extract approx 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 populated the installer will be in a position to undertake a series of test runs of DBAUDIT V3.00 to ensure correct installation.

xxxxxx.DBV300.INSTALL install library contains JCL, example user-exits and DBAUDIT execution jobs.

xxxxxx.DBV300y.LOAD – binary executable DBAUDIT 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 V300 Load Library.

Apply Product Protection Code

DBAUDIT has a Product Protection Codeword. This codeword is at least 20 bytes long and will need to be supplied so that DBAUDIT will run on your system. The SOLD (or trial) code for your site will allow DBAUDIT to run, and is available from your local affiliate or CCA Software.

The code is supplied to DBAUDIT as PART of the DBAUDIT EXEC card as follows:

```
//AUDIT74 EXEC PGM=DBAUDIT,PARM=BIPLBJPHHJJHJHMKIHKH
```

Chapter 3

Operations

Operational Overview - a Tutorial Introduction

This section provides a tutorial introduction to DBAUDIT. By changing the sample JCL provided on the distribution library and submitting the job first hand experience is gained in the use of DBAUDIT.

Source member AUDJCL1 contains sample JCL required to execute DBAUDIT against a stand-alone ADASAV backup (i.e., an ADABAS backup taken when the database was "down").

The input statements supplied to DBAUDIT will perform the following processing on the NATURAL system file, FNAT:

- **check that all source programs have corresponding object modules, and vice versa (this processing "rule" will encounter many contrary examples on the typical FNAT file, as SOFTWARE AG does not distribute the source code for many programs);**
- **enumerate the different source libraries, showing the name of each library and the number of records in it;**
- **enumerate the different object libraries, showing the name of each library and the number of records in it;**
- **extract all source programs from NATURAL library "SYSDBA".**

Prepare JCL to execute DBAUDIT

Following the comments at the start of the JCL, we will now prepare this job for execution.

Step 1 **Change the JOB card to site requirements.**

Step 2 **Make sure region and time parameters will be sufficient and match the job class (unless your FNAT file is extremely large, this job should not take more than 2 - 4 CPU seconds on a 20 MIP processor). A region size of 4 MB should be sufficient.**

Step 3 **Change the STEPLIB DSN to the name of the DBAUDIT load library.**

Step 4 **Specify an ADASAV dump as input. One DD name is required for each dataset produced by the ADASAV backup. Dependent on the DRIVES= parameter supplied to the ADASAV utility when the backup was generated, 1 or more datasets will have been created, being written to the //DDSAVE1, //DDSAVE2 etc. DD names in the ADASAV JCL. DBAUDIT needs to read those datasets.**

They must be supplied to DBAUDIT in the same order as they were generated from ADASAV (i.e., DDSAVE1 output from ADASAV becomes the DDSAVE1 input to DBAUDIT; the DDSAVE2 output from ADASAV becomes the DDSAVE2 input to DBAUDIT, and so on).

If you can arrange the backup of a small ADABAS database to be resident on disk whilst you are experimenting with DBAUDIT, delays waiting for tape mounts will be removed and you will be encouraged to experiment and explore DBAUDIT processing options.

Step 5 **Verify that the SYSOUT options for the output datasets are adequate (change output DD statements to reference datasets with variable length, blocked records if required). The RULE process will probably generate several thousand lines of output to the //OUTPUT DD (beware of System S722 abend!), whilst the //SRCLIB and //OBJLIB files will be very small.**

Step 6 **Change all occurrences of the string 'NNN' to your FNAT system file number (leading zeros are optional, i.e., '9', '09' and '009' are all equivalent).**

Step 7 **Submit the job.**

When the DBAUDIT job completes, check the job log and return code. Glance at the run summary, and then view each of the 4 output files produced by DBAUDIT:

the default output file "OUTPUT" will be reporting on mismatches between source and object. A short error message will be followed by the key in error (the library and program/module name) and the ISN;

the SRCLIB output file will contain one record for each NATURAL source library, showing the name of the library followed by an unpacked decimal count of the number of records in the library;

the OBJLIB output file will contain one record for each NATURAL object library, showing the name of the library followed by an unpacked decimal count of the number of records in the library;

the PROGRAMS output file will contain all the source code from library SYSDBA. Each record will contain 1 line of source code, preceded by the source library and program name.

EXEC Card

This section describes the relevant parameters with regard to the EXEC statement in the execution JCL for DBAUDIT.

Region Size

DBAUDIT's region requirements are mainly dependent on:

Input Buffer Requirements

DBAUDIT should be configured to process its input files as fast as possible. Large block sizes and many buffers on the input ADASAV datasets will improve input processing, but at the expense of valuable below-the-line memory.

Although DBAUDIT will close input datasets which do not contain any required data, the peak buffer requirements will occur at the start of the DBAUDIT job, and memory must be available to meet this peak.

As discussed below, it is recommended that any input datasets containing only Associator or known to contain data storage extents that are not of interest be given DCB=BUFNO=1, whereas the "valuable" input datasets be given as large a BUFNO as practical. Assuming:

input ADASAV blksize of 32KB;

4 input ADASAV datasets, 3 of which contain data storage RABNS;

we aim at providing 250K of buffer space per "valuable" input dataset, as a rule of thumb, then we should set BUFNO=8 on DDSAVE2, DDSAVE3 and DDSAVE4 (32 * 8 = 256KB) and set BUFNO=1 on DDSAVE1.

This would result in approx. 800KB being assigned to input buffers.

Output Buffer Requirements

DBAUDIT output processing volumes are unlikely to approach the input volumes, and so output throughput is less critical. However, requests producing large amounts of output (such as LIST/LISTALL/JOIN) should be assigned to datasets with larger block sizes and more buffers.

When DBAUDIT is being used to produce dozens of output files, be sure to calculate the buffer requirements and adjust region size (or input and sort region requirements) accordingly.

Sort Space

DBAUDIT invokes the system sort when necessary to process the following input statements:

```
RULE/JOIN (2 sorts invoked - one for the left hand side and one
for the right hand side of the rule/join);
TOTAL .. AGAINST
COUNT with or without AGAINST
UNIQUE
NULL UNIQUE
```

DBAUDIT EXEC card parameters enable the specification of lower and upper bounds for the memory to be used per sort, as well as the maximum memory per sort to be allocated below the 16MB line (OS390 users only).

Additionally, DBAUDIT tries to optimize sort performance (including memory assignment) by calculating the "size" of each sort before invoking the sort function.

Allocating large amounts of memory to the sort certainly reduces I/O to sort work datasets, and will usually dramatically improve sort times, unless real memory shortages cause severe paging.

DBAUDIT buffers and code

DBAUDIT allocates internal buffers, each 4KB in size, for communications between input, sort and output tasks. The default number of buffers is 100, but this can be changed dynamically by the BUFFERS= EXEC card PARM. The size of other DBAUDIT work areas and code does not exceed 100KB.

General Guidelines

In general, OS390 users should allocate a region large enough to support the number of sorts and sort size they are using. If 20 concurrent sorts, each with 1MB of storage are to be invoked, allocate at least 25MB on the job or exec card.

Non XA/ESA users must be more careful when specifying sort processing and sort size. Assuming a private region size of 8MB, and 200KB sort space per sort, and 2MB allocated to input/output buffers, DBAUDIT buffers and code, then approx. 6MB should be available for support 6MB/200KB = 30 concurrent sorts. Large sorts may take a longer elapsed time with only 200KB of memory. Non XA/ESA users should specify the largest region size they can.

Execution Time

The CPU time taken by DBAUDIT is dependent upon:

- **the number of RABNS to be decompressed (i.e., the total number of data storage RABNS for files that are referenced in the DBAUDIT user input);**
- **the number of fields to be decompressed;**
- **the complexity of the selection and validation logic;**
- **the volume of output generated;**
- **the number of sorts, the memory available to them and the number of records to sort. Given the number of factors and their variability, it is very hard to estimate the CPU time, and so it is suggested that you run DBAUDIT with a high CPU time limit.**

It is recommended that prior to commencing a large scale production DBAUDIT job a series of small trial DBAUDIT jobs be run limiting the number of records processed. This will allow an estimate to be made of the CPU and memory requirements and the run time for the full production job.

Parameters

DBAUDIT accepts user input from 2 sources:

- **PARMS file, containing DBAUDIT processing specifications such as the SELECT, RULE, PROCESS and JOIN statements;**
- **JCL EXEC PARM parameters.**

  The purpose of the EXEC PARMS are:

  - **to supply the product execution code required by all users as of V3.00;**
  - **to override default or zapped "constants" used to effect DBAUDIT processing.**

The format of the EXEC card JCL parameter string is:

```
,PARM='<parm type>=<parm value>,....'
```

Chapter 6 contains the details of all parameter types and their default values, valid ranges, usage considerations and abbreviations.

It is suggested that when first using DBAUDIT the default settings for parameters should be used. Among the first parameters you may wish to experiment with will be:

> **MAXSORTK. This parameter specifies the maximum memory a sort can use (both above and below the 16MB line). Its default value (400KB) is probably too high for a non XA\ESA site wishing to perform large numbers of concurrent sorts, and is probably too low for an XA\ESA site wishing to perform very large sorts.**
>
> **SORTMSGCLASS. This parameter specifies the sysout class used for sort messages and diagnostics. Most sites will probably be uninterested in these details, and may wish to send them to the "flush" sysout class.**

The CODE= parameter will be required on every DBAUDIT run. The value of this parameter is specific to your CPU serial number and license agreement (i.e., it may expire at the end of a trial period - you will be given 50 days notice of the expiry by DBAUDIT message 467I). CCA Software or your local distributor will supply the code to you.

The defaults for all parameters may be permanently changed by zapping the appropriate load module (see Chapter 7). The CODE parameter can be bypassed by permanently zapping the licence code into DBAUDIT, this zap will be supplied by CCA Software or your local affiliate upon request.

INPUT/OUTPUT FILES

ADASAV Input Files

DBAUDIT reads the datasets produced by the ADABAS ADASAV backup utility. ADASAV is normally used to produce several output datasets "in parallel" by using the "DRIVES=" option.

This technique helps to reduce the elapsed time of the ADASAV backup, especially when backing up to cartridges or tape drives connected to the mainframe by multiple I/O channels.

The ADASAV backup job specifies a separate file (DDNAME) and dataset for each of these parallel output "streams". DBAUDIT needs to be supplied with the exact same file names and DSN. E.g., if the ADASAV JCL looked like this:

```
//DDSAVE1   DD DSN=ADABAS.DB051.BACKUP1(+1),DISP=(NEW,CATLG),
//                UNIT= CART
//DDSAVE2   DD    DSN=ADABAS.DB051.BACKUP2(+1),DISP=(NEW,CATLG),
//                UNIT=CART
//DDSAVE3   DD    DSN=ADABAS.DB051.BACKUP3(+1),DISP=(NEW,CATLG),
//                UNIT=CART
//DDKARTE   DD    *
      ADASAV SAVE,DRIVES=3,
//*
```

Then the DBAUDIT JCL to read the most recent backup would look like this:

```
//DDSAVE1         DD    DDSN=ADABAS.DB051.BACKUP1(0),DISP=SHR,
//                      DCB=BUFNO=1
//DDSAVE2         DD    DSN=ADABAS.DB051.BACKUP2(0),DISP=SHR,
//                      DCB=BUFNO=8
//DDSAVE3         DD    DSN=ADABAS.DB051.BACKUP3(0),DISP=SHR,
//                      DCB=BUFNO=8
```

The reasons for the use of the DCB=BUFNO JCL parameter will be discussed in a later section.

If ADASAV is used to produce DUAL output datasets, then either the "original" or "DUAL copy" can be used by DBAUDIT.

Output files from ADASAV can be supplied concatenated to DBAUDIT. E.g., assume the ADASAV backup was run with DRIVES=5, but only 3 drives are available when DBAUDIT is to be run. DBAUDIT could be run with one of the many variants of the following JCL:

```
//DDSAVE1         DD    DSN=ADABAS.DB051.BACKUP1(0),DISP=SHR
//                      DCB=BUFNO=1
//                DD    DSN=ADABAS .DB051.BACKUP2(0),DISP=SHR,
//                      DCB=BUFNO=1
//DDSAVE2         DD    DSN=ADABAS.D B051.BACKUP3(0),DISP=SHR,
//                      DCB=BUFNO=8
//DDSAVE3         DD    DSN=ADABAS.DB0 51.BACKUP4(0),DISP=SHR
//                      DCB=BUFNO=8
//                DD    DSN=ADABAS.DB051.BACKUP5 (0),DISP=SHR
```

The basic rule to be followed when concatenating datasets is that the RABN numbers read by DBAUDIT must be ascending within a DDSAVEx file, and ascending across DDSAVEx files. This last requirement means that the first RABN read by each DDSAVEx file must be greater than the first RABN read by the preceding DDSAVEx file.

DBAUDIT will read an ADASAV backup produced whilst the ADABAS nucleus was active. However, if updating was being carried out against the database whilst the ADASAV backup was being taken, then it is very unlikely that the ADASAV backup contains a complete and consistent copy of the database. Such a backup will need to be augmented with either a protection log or a QDUMP incremental backup.

This issue is discussed in separate sections following.

The DBAUDIT JCL statements describing the ADASAV input data sets can usually be coded without any options other than DISP=SHR, unless the ADASAV datasets have not been catalogued, in which case manual coding of UNIT= and VOL=SER= parameters will be required. However, the BUFNO parameter can be used to greatly improve DBAUDIT throughput and tune virtual memory use.

Coding FREE=CLOSE on the DBAUDIT datasets is not required and should not be specified, as DBAUDIT will dynamically de-allocate its files (and associated tape/cartridge units) as soon as possible.

Coding the BUFNO Parameter

In most cases, DBAUDIT elapsed time is likely to be highly dependent on the time taken to read the ADASAV backup. Although DBAUDIT contains optimization methods to skip large parts of the backup which it knows contain no relevant information (such as all of the ASSOCIATOR except for GCB, FCB's and FDT's and DATA STORAGE for files un-referenced by any input parameter specifications), it is beneficial to expedite the reading of what is likely to be a still large amount of data storage.

The BUFNO parameter determines the number of input buffers set aside for reading the backup. The product of BLKSIZE and BUFNO is the size of the input buffer for each dataset. By increasing this value input throughput is improved, but only up to a point, after which little or no gains are evident and paging on the buffer pool may actually degrade performance.

The exact point will vary on machine configuration and accompanying workloads, but 250 KB is probably a good (and generous) estimate of a desirable and cost effective buffer pool size, per input dataset. Divide the ADASAV dataset blocksize (usually 32KB) into 250KB to determine an appropriate value for BUFNO.

If the ADASAV backup has been produced to 3 or more different datasets, it is very likely that one or more datasets will contain nothing but ASSOCIATOR blocks. As DBAUDIT only reads the first few blocks of ASSOCIATOR, supplying a large buffer for these datasets would not improve performance, and would tie up virtual memory that could be better used by other "active" input dataset buffers or for sorting. Hence, it is strongly recommended that datasets containing only ASSOCIATOR blocks be given a BUFNO of 1 (DCB=BUFNO=1), unless there are a large number of files defined to the database, then a BUFNO of 2 on the first dataset may be more appropriate.

The split of ASSOCIATOR/DATA STORAGE blocks to the ADASAV datasets is reported by the ADASAV utility.

Dynamic Deallocation of DDSAVEx files

DBAUDIT reads the first few ASSOCIATOR blocks to produce a map of the RABNS it will need to process. Each input dataset is then read to determine how DATA STORAGE blocks are split over the input datasets. DBAUDIT then knows which input datasets contain no relevant information, or when the last required block has been read from an input dataset. As soon as possible, DBAUDIT will close an input dataset and dynamically de-allocate it (the equivalent of coding FREE=CLOSE on the JCL DD statement). The operating system will then de-allocate the dataset and the tape or cartridge unit assigned to the dataset, and in so doing will make the unit available for another job in the system.

Protection log input

When an ADASAV online database backup is produced, the active ADABAS nucleus writes extra information to the protection logs which is read by the ADASAV RESTORE function if that online backup is used in a database restore.

This "extra information" is actually all the blocks written to the database during the course of the ADASAV backup.

If there is any chance of files being updated during the online ADASAV that will be read and processed by DBADIT the protection log dataset(s) generated by ADABAS during the ADASAV must be supplied. The protection log(s) are read by DBAUDIT through ddname "PLOG". When protection logs are being used, some additional sort related datasets must also be present in the DBAUDIT JCL.

For example, if 2 protection log datasets were generated by the ADABAS nucleus during the course of the online ADASAV backup, then the following DD statements should be added to the DBAUDIT JCL:

```
//*                Concatenate as many PLOGS as were written
//*                during the ADASAV, just as you would if
//*                supplying the PLOGS an ADASAV RESTORE
//*
//PLOG       DD    DSN=ADABAS.DB051.PLOG.G00300V00,DISP=SHR
//           DD    DSN=ADABAS.DB051.PLOG.G00301V00,DISP=SHR
//*
//SORTWK01   DD    UNIT=SYSDA,SPACE=(CYL,(10,10))
//SORTWK02   DD    UNIT=SYSDA,SPACE=(CYL,(10,10 ))
//SORTWK03   DD    UNIT=SYSDA,SPACE=(CYL,(10,10))
//SYSOUTS    DD    SYSOUT=*    SORT MESSAGES
//*
```

DBAUDIT reads the protection log(s), searching for all special ADASAV dump blocks written between the start and end ADASAV checkpoints on the plog. Blocks that pertain to files being processed by DBAUDIT are extracted and sorted in reverse-timestamp, RABN number order to be later merged with (and take precedence over) blocks from the ADASAV backup.

Unless very heavy update activity was being performed on the files of interest to DBAUDIT during the ADASAV backup, it is likely that few blocks will be extracted from the protection log, and hence sort requirements will be relatively small.

QDUMP incremental backup input

Users of the QDUMP incremental backup product can supply DBAUDIT with both:

- **their latest complete ADASAV backup (multi or single user mode);**
- **their latest QDUMP incremental backup (multi or single user mode).**

Protection logs are NOT required, and MUST NOT be supplied. The QDUMP backup is read through file "QDUMP". If more than one output file was produced by QDUMP (i.e., DD statements OUTPUT1, OUTPUT2, ... were present in the QDUMP JCL), then all output QDUMP datasets must be concatenated to the QDUMP DD statement. As with protection log processing, some additional sort related datasets must also be present in the DBAUDIT JCL.

For example, if 2 QDUMP datasets were output by the QDUMP incremental backup, then the following DD statements should be added to the DBAUDIT JCL:

```
//*   Concatenate all QDUMP output datasets to the QDUMP DD
//*
//QDUMP     DD    DSN=ADABAS.DB051.QDUMP1.G00154V00,DISP=SHR
//          DD    DSN=ADABAS.DB051.QDUMP2.G00154V00,DISP=SHR
//*
//SORTWK01  DD    UNIT=SYSDA,SPACE=(CYL,(20,10))
//SORTWK02  DD    UNIT=SYSDA,SPACE=(CYL,(20,1 0))
//SORTWK03  DD    UNIT=SYSDA,SPACE=(CYL,(20,10))
//SYSOUTS   DD    SYSOUT= *   SORT MESSAGES
//*
```

DBAUDIT reads the QDUMP datasets for blocks that pertain to files being processed by DBAUDIT. These blocks are extracted and sorted to be later merged with (and take precedence over) blocks from the ADASAV backup.

The sort requirements will depend on how many blocks have been backed-up by QDUMP belonging to files being processed by DBAUDIT. This is turn, will be dependent on 2 factors:

- **how long since a full backup was produced;**
- **the number of DIFFERENT data storage blocks updated in the relevant files.**

DBAUDIT Run Summary file

DBAUDIT generates a run summary, reporting the following information:

- **DBAUDIT release and zap level information, start and end timestamps;**
- **parameters in effect for this run;**
- **echo of input specifications;**
- **database id, name and timestamp, read from the ADASAV backup;**
- **input statistics, reporting the following by DDNAME and grand total:**
- **RABNS read from the dataset;**
- **RABNS analyzed (i.e., RABNS pertaining to files being processed in this run);**
- **Records analyzed (actual ADABAS records extracted and partially decompressed from the analyzed RABNS);**

Output blocks (i.e., number of output blocks generated by this input task to be processed by a sort or output task);

Output records (i.e., number of output records generated by this input task to be processed by a sort or output task);

Buffer waits (i.e., the number of times this input task had to wait for a free internal buffer into which the output records are blocked);

file statistics, reporting the following by ADABAS file being processed and grand total:

File name;

RABNS read;

Records analyzed;

Output records;

sort statistics, reporting the following for each sort and grand total:

Process/Rule/Join ID (Each sort is associated with the PROCESS, RULE or JOIN specification it is processing for - PROCESSes invoke "simple" sorts, whereas RULEs and JOINs give rise to 2 sorts, sorting the data from the left hand side (LHS) and the right hand side (RHS) of the rule, and another task to match the results of the LHS and RHS sorts;

Input blocks (i.e., number of blocks passed to the sort by input tasks);

Records sorted (i.e., number of records passed to the sort within the input blocks);

Output blocks (i.e., number of output blocks generated by this sort task to be processed by an output task);

Output records (i.e., number of output records generated by this sort task to be processed by an output task);

Buffer waits (i.e., the number of times this sort task had to wait for a free internal buffer into which the output records are blocked):

Sort key length;

Sort record length;

output statistics, reporting the following by output DDNAME and grand total:

Output blocks;

Output records;

The SUMMARY file is opened with the following DCB attributes:

```
DCB=(LRECL=133,RECFM=FBA)
```

Input Parameter file

DBAUDIT gets its processing directives from the "PARMS" file. The content of this file is the subject of the next chapter, (4. INPUT AND OUTPUT).

The PARMS DD statement will typically be supplied as an instream dataset as follows:

```
//PARMS     DD     *
      SELECT FILE 12 WHERE.....
/*
```

but may, of course, be specified as a dataset. DBAUDIT requires the PARMS file to have fixed record format, with a minimum record length of 72 bytes. Only the first 72 bytes of each record are parsed. The following DCB attributes would be considered "normal":

```
RECFM=FB,LRECL=80
```

Output files

The output generated by RULEs and PROCESSes is written to files optionally specified by the user by the DDNAME clause on each RULE and PROCESS statement. The default DDNAME is "OUTPUT".

Up to 200 different output files may be specified in a single DBAUDIT run. The output from multiple RULE and PROCESS statements may be generated to just a single output file. If no DDNAME clauses are specified, all the output from all the RULE and PROCESS statements will be directed to the DDNAME "OUTPUT".

DBAUDIT does not write headers or trailers to its output files, and hence it is possible that output files may be empty (the DBAUDIT user exit 1 can be used to modify many aspects of DBAUDIT output processing, including the writing of file headers and/or trailers).

DBAUDIT opens all of its output files with the following DCB attributes:

```
DCB=(LRECL=4000,BLKSIZE=6000,RECFM=VB)
```

The LRECL and BLKSIZE parameters can be overridden by dataset label or JCL DCB attributes. Record format cannot be overwritten - it must be VB. The maximum record length that DBAUDIT supports is 4080 bytes.

As mentioned in the preceding input file section, the BUFNO parameter can be used to tradeoff increased I/O throughput against virtual memory usage. The default operating system BUFNO (5) will usually be appropriate for DBAUDIT output files. However, output files generating large volumes of output will be assisted with a large BUFNO.

Similarly, output files to which only a handful of records will be written will not be impeded with a BUFNO of 1, and hence virtual memory will be conserved and diverted to more productive uses, such as sort space or input buffers.

This point is particularly relevant if many output files have been specified.

PROCESS, RULE and JOIN sort files

DBAUDIT requires that the sort files used for processing protection logs or QDUMP incremental backups be statically allocated by the user in the DBAUDIT JCL. However, all the files allocated by "dynamic" sorts, as invoked by PROCESS, JOIN and RULE specifications will be allocated dynamically by DBAUDIT and the sort function.

They can, however, be pre-allocated by the user in JCL should the default dynamic allocations be unsuitable. (See also the EXTRASORTPARMS parameter described in the EXEC Card Parameters chapter on page 51).

Two types of file are dynamically allocated by the sort process:

- **a sort messages file;**
- **one or more sort work files.**

The sort message file is dynamically allocated by DBAUDIT prior to invoking the system sort. It is allocated to a SYSOUT class specified by the SORTMSGCLASS parameter.

The sort message file contains messages from the sort function such as record counts, resources used etc. The //SORTDIAGS DD statement will, if present in the DBAUDIT JCL, cause the sort function to write additional diagnostic information to the sort messages file, which may assist in tuning (or understanding) the sort. However, SORTDIAGS should be used with caution, as it may degrade sort performance.

To "suppress" the sort messages, use the SORTMSGCLASS parameter to set the sysout class to a class which is "flushed" at your site (see Chapter 5).

Alternatively, to capture the sort messages into a dataset, pre-allocate the sort message files. The name of all dynamic sort message files used by DBAUDIT has the form:

> **"SnnnMSGS" where nnn may be a number between 000 and 999, which will depend on the number of output datasets and sorts.**

However, this number is repeatable from run to run if the same user input specifications are given to DBAUDIT.

The sort work files are dynamically allocated by each invoked sort according to your site default for the following sort installation parameter:

> **DYNALLOC - specifies number of and unit type for dynamically allocated sort work datasets. DBAUDIT does not attempt to override this parameter, unless the EXTRASORTPARMS parameter which may be used to modify SORT options - see chapter 6 for a description.**

Should the defaults prove inappropriate, either change the system defaults or add the required sort work definitions to the DBAUDIT JCL.

The name of sort work dataset to be dynamically allocated by the sort tasks have the form "SnnnWKmm" where nnn may be a number between 000 and 999, which will depend on the number of output datasets and sorts, and mm is the number of the sort work dataset for that particular sort (e.g., 01, 02, 03 if 3 sort datasets are dynamically allocated).

Hand coding the sort work definitions will only be useful if the default allocations are grossly inappropriate, such as allocating a large number of heavily used sort work datasets onto just 1 or 2 disk volumes.

Most XA/ESA users will find the setting of generous sort memory limits and the use of advanced sort techniques which automatically take advantage of the XA and ESA architecture will make the sort work dataset issue unimportant.

EXCPVR

Depending on your installation's SORT product and options chosen by system programmers when it was installed, the sort program may routinely page fix some or all of its buffers (refer to your sort product documentation on the use of the EXCPVR sort option). This will normally provide optimum CPU performance, at the expense of removing the use of large amounts of memory from concurrent system workloads (other jobs, TSO users, etc.). However, when a single DBAUDIT run is processing a large number of RULE and JOIN requests, the amount of storage page fixed by the sort program may degrade overall system performance. To prevent the sort program from page fixing its buffers, consider using the DBAUDIT EXTRASORTPARM parameter to override the system default EXCPVR setting, e.g.:

```
EXTRASORTPARM='''EXCPVR=NONE'''
```

DBAUDIT Diagnostic file - DDMESS

In case of abnormal termination, DBAUDIT will attempt to write diagnostic summary information to file DDMESS. DDMESS should normally be allocated in the JCL as follows:

```
//DDMESS   DD   SYSOUT=*
```

DBAUDIT only attempts to open DDMESS when a fatal error has been detected.

SYSABEND or SYSUDUMP

In the event of abnormal termination, DBAUDIT will attempt to write diagnostic summary information to file DDMESS. After this process, it will abend with an option to generate a system dump to the SYSABEND or SYSUDUMP DD statement if either is present.

DDMESS will normally provide all the diagnostic information that your DBAUDIT support team will need to identify the problem and its cause. However, for some incidents, they will request information from a SYSABEND or SYSUDUMP.

Chapter 4

INPUT AND OUTPUT

Introduction

This chapter describes the syntax and semantics of the specification language used to describe DBAUDIT processing requests, and the format of the generated output.

DBAUDIT processing is built around 4 statements:

SELECT

RULE

PROCESS

JOIN/JOINNULL

A fifth statement, "PRINT", can be used to write documentation to the DBAUDIT output files, but does not otherwise effect processing.

SELECT is the primary statement - it describes a set of data to which following RULE and PROCESS statements apply. Hence, RULE, PROCESS and JOIN statements may be considered to be "nested" within a SELECT.

The scope of a SELECT (i.e., the RULE, PROCESS and JOIN statements that it encompasses) is delimited by another SELECT or by the end of the parameter input file.

DBAUDIT accepts free-form input. Blanks (including blank lines) can be freely used to separate syntactic elements. Any line with an asterisk in column 1 is treated as a comment.

SELECT statement

The SELECT statement describes the set of data to which the following RULE, PROCESS and JOIN statements apply, by specifying:

the primary file number;

the record selection criteria (optional).

If the record selection criteria are omitted, then all records within the primary file will be processed according to the following RULE, PROCESS and/or JOIN statements.

The term "primary file number" has been used here, because the RULE and JOIN statements specify a file number (the secondary file number) which will often be different from the file specified by the SELECT.

Any number of SELECT statements may be supplied. The same file can be specified on any number of SELECT statements. The syntax of the SELECT statement is:

```
SELECT [ FILE ] <primary-file-number> [ <where-clause> ]<MI>
```

the FILE keyword is optional, and may be used to assist readability;

file-number is a 1 to 3 digit number (leading zeroes optional) between 1 and 255;

the where-clause is optional. If omitted, all records in the file will be selected. If supplied, only records from the file which satisfy the where clause will be accepted for subsequent processing.

The syntax of the where-clause is:

```
WHERE <where-criteria> ENDWHERE
```

The where-criteria consists of one or more relational expressions, optionally preceded by the logical negation operator "NOT" and connected by the logical operators "AND" and "OR". The usual logical operator precedence hierarchy (NOT AND OR) can be overridden by nesting sub expressions within brackets. The syntax of the relational expression is:

```
<field-expression> <relational-operator> <value-expression>
<FOL>eg:
FIELD AA    =     'RED'
FIELD AB    <=     1000
```

The field expression describes the information from the ADABAS record to be compared with the value expression. The syntax of the field expression is:

```
FIELD <ADABAS-field-name>
[   [  [ start-character-position [ - end-char-postn ] ]
      [ /  *
            | start-index-1  [ - end-index-1 ]
[  , *
            |  start-index-2 [ - end-index-2 ]
       ]
 ]
[ CONCAT <field-expression>  ]
```

The ADABAS field name is the 2 character field name from the ADABAS field definition table (FDT). "Virtual" fields (which do not actually exist in the data storage records) such as superdescriptors, subdescriptors, phonetic and hyper descriptors cannot be specified. However, their component fields may be specified. Characters start and end position values can be used to represent sub descriptors, and whole and partial fields can be concatenated to represent super-descriptors.

Field expressions including MU and PE element ranges introduce additional evaluation complexity. Refer to Appendix C for full details. DBAUDIT implements the special ADABAS field name "##" to represent the ISN of a record. This field can be used in any context in which a "normal" ADABAS field could be specified.

Examples of field expressions are:

a single, simple, complete ADABAS field;

FIELD AA;

a sub-string of a single, simple ADABAS field;

FIELD AA(2-4) - 3 bytes starting at byte 2 (i.e., the 2nd byte);

all occurrences of a PE or simple MU ADABAS field;

FIELD P1(/*);

all occurrences of a MU field within a PE;

FIELD P2(/*,*);

selected occurrences (5 through 10) of the 1st char of a PE field;

FIELD P3(1/5-10);

a complicated concatenation;

```
FIELD P3(1-4/*) CONCAT FIELD P1(/*) CONCAT FIELD AA
    CONCAT FIELD BJ(3) CONCAT FIELD BT(/2)
```

The relational operators are:

```
=           Equal
<=          less than or equal
>=          greater than or equal
<           less than
>           greater than
<>          not equal
```

The value expression may be:

a numeric constant (decimal number), signed or unsigned e.g.:

```
0
-54
12345678
```

a quoted character string constant e.g.:

```
'LOBSTER'
```

a quoted hexadecimal string constant e.g.:

```
HEX '0000'
HEX '939682A2A38599FF'
```

the special null designator "NULL" e.g.:

```
NULL
```

a mask value built using the mask elements:

```
N              -   match numeric 0 through 9
A              -   match upper/lower alphabetic
X              -   match anything at all
DD             -   match day of the month
MM             -   match month of the year ('01' through '12')
YY             -   match 2 digit year within century
YYYY           -   match 4 digit year (including century)
quoted string -   exact match on characters within the string
```

Examples:

```
MASK (NNNN)
MASK (AAA'-'NNN)
MASK (DD'/'MM'/'YYYY' A 'XXXNNN')
```

Examples of complete relational expressions are:

```
FIELD AA = 'BLUE'
FIELD AA CONCAT FIELD AB = 'BLUE CAR'
FIELD AA(1) = 'B'<R>FIELD LJ(1-3) = 'SYS'
FIELD JJ(4/*,1-5) = HEX '01'
```

Examples of complete SELECT statements are:

All records from file 14:

```
SELECT 14
```

All records from file 14 with non-null value for field LJ:

```
SELECT 14 WHERE FIELD LJ <> NULL ENDWHERE
```

All records from file 14 with characters 1 through 8 of field LJ = 'SYSSEC'.

```
SELECT 14 WHERE FIELD LJ(1-8) = 'SYSSEC'   ENDWHERE
```

All records from file 14 with characters 1 through 8 of field LJ = 'SYSSEC' and having a character position 3 of any occurrence of MU field LL = '*'.

```
SELECT 14 WHERE FIELD LJ(1-8) = 'SYSSEC  '
AND FIELD LL(3/*) = '*' ENDWHERE
```

All records from file 210 with field BB = 0 and either field AA = 1 and field AB non null or field AA = 2 and field AB is a digit or field BT concatenated with field BS is not equal to 'Z1'.

```
SELECT 210 WHERE FIELD BB=0
AND   (     (FIELD AA = 1 AND FIELD AB <> NULL
OR          (FIELD AA = 2 AND FIELD AB = MASK(N))
OR          NOT (FIELD AT CONCAT FIELD BS = 'Z1'))
ENDWHERE
```

RULE statement

The RULE statement specifies a referential integrity rule. Referential integrity rules allow for relationships between records to be checked. (The RULE statement specifies how many records with matching keys are allowed to exist without reporting an error). The following basic relationships between records are supported:

```
ONE TO ONE
ONE TO MANY
MANY TO ONE
MANY TO MANY
```

The "MANY" relationship supports an extension which allows for the specification of an explicit number or range, e.g.: ONE TO MANY(5-20).

The syntax of the RULE statement is:

```
RULE rule-id
      [estimate-specification] :
      <primary-field-expression>
            <referential-relationship>
FILE <secondary-file-number>
            <secondary-field-expression>
            [ <where-clause> ]
[LIST  [LHS <field-specification>]
            [RHS <field-specification>] ]
            [ <output-specification> ]  ;
```

the rule-id is a character string of between 1 and 8 characters (A through Z, 0 through 9) that should be used to uniquely identify the rule. All output produced from the application of the rule will be preceded by the rule-id.

the optional estimate specification may be used to override the default estimate that DBAUDIT would otherwise make of the number of records that will be sorted on each "side" of the rule. Supplying this estimate is only necessary when the estimate made by DBAUDIT will be so inaccurate as to cause the sort to fail (i.e., DBAUDIT estimates too low, probably due to the presence of MU/PE occurrences in the rule field expressions) or to waste temporary disk sort work space (i.e., DBAUDIT estimate was too high).

The syntax of the estimate specification is:

```
ESTIMATE record-count
```

where record count is a positive number.

the primary field expression describes the field(s) on which the primary file is to be matched with the secondary file. Any field expression (containing concatenated fields, MU/PE expressions and field sub ranges) may be specified.

The referential relationship has the following syntax:

```
ONE|MANY [ ( low-range [ - high-range ]) ]
TO ONE|MANY [ ( low-range[ - high-range ]) ]
```

Examples:

```
ONE TO ONE
ONE TO MANY
MANY(3) TO ONE
MANY(1-9999) TO MANY(1-9999)
ONE TO MANY(0-5)
```

the secondary file number specifies the file number to be matched against the primary file. The primary and secondary file numbers may be the same.

the secondary field expression describes the field(s) on which the secondary file is to be matched with the primary file. Any field expression (containing concatenated fields, MU/PE expressions and field sub ranges) may be specified.

the optional where clause allows the selection of records from the secondary file which are to participate in the match against the primary file (recall that the SELECT statement preceding this RULE statement will have specified any selection criteria on records from the primary file).

the LIST specification can be used to add output fields from either the LHS or RHS records to the matching key value, which is always output. These extra values are only output for LHS/RHS MISSING conditions with the first ISN with the key value causing the error. The extra values may help to identify the record in error. The LIST specification should be used frugally, as it increases the size of the sort record, and hence the time and space required for sorting.

the optional output specification may be used to:

override the default output file name;

specify that ADABAS field names should be inserted into the output record;

specify a heading to appear at the top of the output file.

The syntax of the output specification is:

```
DDNAME output-file-name
[FORMATED]
[HEADING <'heading string'>]
```

The output file name must be a valid JCL DD name, and must be present in the DBAUDIT JCL. The default output file name is "OUTPUT".

If heading strings are specified more than once (on different RULE, PROCESS or JOIN statements) for the same output ddname, then only 1 heading will actually be output. The heading chosen will be for the processing statement against the lowest file number appearing first in the input stream. Examples of RULEs:

```
SELECT FILE 12
RULE          M1:    FIELD AA ONE TO ONE FILE 18 FIELD AA ;
```

field AA in file 12 should be in a one-to-one relationship between field AA in file 18. IE, for every record in file 12 there should be exactly one matching record in file 18, and the matching is to be performed using field AA in both files.

Errors will be reported (ie output) in the following cases:

more than 1 record in file 12 exists with the same value for field AA (i.e., field AA contains duplicates);

more than 1 record in file 18 exists with the same value for field AA (i.e., field AA contains duplicates);

a record exists in file 12 which does not have a match in file 18;

a record exists in file 18 which does not have a match in file 12;

```
SELECT FILE 12
RULE        M2:  FIELD AA ONE TO MANY FILE 18 FIELD AA ;
```

field AA in file 12 should be in a one-to-many relationship between field AA in file 18. IE, for every record in file 12 there should be one or more matching records in file 18, and the matching is to be performed using field AA in both files.

Errors will be reported in the following cases:

more than 1 record in file 12 exists with the same value for field AA (i.e., field AA contains duplicates);

a record exists in file 12 which does not have a match in file 18;

a record exists in file 18 which does not have a match in file 12.

```
SELECT FILE 12
RULE        M3:  FIELD AA ONE TO MANY(0-3) FILE 18 FIELD AA ;
```

as for the previous example, except for each record in file 12, between 0 and 3 records should exist with the same key in file 18. Errors will be reported in the following cases:

more than 1 record in file 12 exists with the same value for field AA (i.e., field AA contains duplicates);

more than 3 records in file 18 occur for a record in file 12;

a record exists in file 18 which does not have a match in file 12.

```
SELECT FILE 12
RULE        M4:  FIELD AA ONE TO MANY FILE 18 FIELD AA
      WHERE  FIELD AA  >'K' OR FIELD BB <>NULL  ENDWHERE
      LIST        LHS FIELD BA CONCAT BB CONCAT BC
      DDNAME F12RULES ;
```

as for the previous straight one to many example, except that only records from file 18 having a value for field AA of greater than 'K' or having field BB not equal to null will be considered.

Also, errors from the application of the rule will be reported to file F12RULES, and missing RHS values will output fields BA, BB and BC from the LHS, along with the LHS record and key.

```
SELECT FILE 12 WHERE FIELD AA <> NULL ENDWHERE
      RULE M4 ESTIMATE 500000:
      FIELD AA CONCAT FIELD P1(1-5/*) ONE TO ONE
      FILE 20 FIELD AB(1-10) ;
```

This example demonstrates the use of more complicated field expressions, including concatenation, field sub range and PE/MU index specification. For each record in file 12 where field AA is not null, the rule will extract each non-null occurrence of the MU field P1, and append the first 5 characters of each occurrence to field AA. (I.e., if a record has 10 occurrences of field P1, then 10 temporary "sort" records would be created to be matched against candidate records from file 20).

The match is to be performed against the first 10 characters of field AB from every record on file 20. The ESTIMATE clause has been provided, because DBAUDIT may make a poor sort size estimate, as it cannot know in advance the number of records to be sorted from the LHS of the expression. There are two major reasons for this:

the impact of the WHERE clause cannot be known;

the number of non-null occurrences of field P1 cannot be known.

The RULE statement produces the following output messages, all prefixed with the rule-id and suffixed with the ISN of the record in error and the key being matched:

```
LHS MISSING:
```

a key from the RHS did not have a matching key on the LHS, and MANY(0-..) was NOT specified on the LHS relational expression.

```
RHS MISSING:
```

a key from the LHS did not have a matching key on the RHS, and MANY(0-..) was NOT specified on the RHS relational expression.

```
LHS COUNT HI
```

a key from the LHS occurred more times than the LHS relational expression indicated - e.g. if ONE TO xxxx was specified, each LHS key should appear only once. If MANY(1-4) TO xxxx was specified, then the LHS key may occur between 1 and 4 times.

```
RHS COUNT HI
```

a key from the RHS occurred more times than the RHS relational expression indicated - e.g. if xxxx to ONE was specified, each RHS key should appear only once. If xxxx TO MANY(1-4) was specified, then the LHS key may occur between 1 and 4 times.

```
LHS COUNT LO
```

a key from the LHS occurred fewer times than the LHS relational expression indicated - e.g. if MANY(2-4) TO xxxx was specified, each LHS key should appear between 2 and 4 times - if it appeared just once, then this message would be output.

```
RHS COUNT LO
```

a key from the RHS occurred fewer times than the RHS relational expression indicated - e.g. if xxxx TO MANY(2-4) was specified, each RHS key should appear between 2 and 4 times - if it appeared just once, then this message would be output.

PROCESS statement

The PROCESS statement allows the following types of processing to be performed on records selected by the previous encompassing SELECT statement:

field value totaling, optionally against another field specification, e.g.:

```
PROCESS TOT1: TOTAL FIELD AB ;

PROCESS TOT2: TOTAL FIELD AB AGAINST FIELD AC
            CONCAT FIELD AD ;
```

field value counting, optionally against another field specification, e.g.:

```
PROCESS COU1: COUNT FIELD AB ;

PROCESS COU2: COUNT FIELD AB CONCAT FIELD AC ;

PROCESS COU3: COUNT FIELD AB AGAINST FIELD AC          CONCAT FIELD
AD ;
```

testing field value uniqueness across all records in the file, optionally testing for uniqueness across the null value as well, e.g.:

```
PROCESS UNIQ1: UNIQUE FIELD AB

PROCESS UNIQ2: NULL UNIQUE FIELD AB
            CONCAT FIELD AC ;

PROCESS UNIQ3: UNIQUE FIELD P1(1-6/*) ;
```

validating field values against logical criteria.

```
PROCESS VAL1: VALIDATE FIELD AB > 0 ENDVALIDATE ;
```

```
PROCESS VAL2: VALIDATE (FIELD D1 = MASK(YYMMDD)
            AND FIELD AB  <> 0
            AND NOT FIELD DB(1) = 'A')
            OR (FIELD D1 = NULL AND FIELD C1 <> NULL)
      ENDVALIDATE ;
```

listing fields from all records satisfying the SELECT criteria, optionally include records composed of null MU/PE values (LISTALL option).

```
PROCESS LIST1: LIST FIELD AA CONCAT FIELD AB ;

PROCESS LIST2: LISTALL FIELD AA(1-3) CONCAT FIELD
            P1(1-10/1-5) ;
PROCESS LIST3: LIST  FIELD AA(1) CONCAT FIELD
            PM(1-10/*,1-5) ;
PROCESS LIST4: LIST  FIELD P1(/*) CONCAT FIELD P2(/*) ;
```

The syntax of the PROCESS statement is:

```
PROCESS <process-id> [estimate-specification] :
                  <process-verb-specification>
                  [ <output-specification> ] [FORMATTED]  ;
```

the process-id is a character string of between 1 and 8 characters (A through Z, 0 through 9) that should be used to uniquely identify the process. All output produced from the application of the process will be preceded with the process-id.

the optional estimate specification may be used to override the default estimate that DBAUDIT would otherwise make of the number of records that may be sorted by the process.

Only some types of process require sorts (UNIQUE, NULL UNIQUE, COUNT, TOTAL .. AGAINST). Supplying this estimate is only necessary when the estimate made by DBAUDIT will be so inaccurate as to cause the sort to fail (i.e., DBAUDIT estimates too low, probably due to the presence of MU/PE occurrences in the process field expressions) or to waste temporary disk sort work space (i.e., DBAUDIT estimate was too high).

The syntax of the estimate specification is:

```
ESTIMATE record-count
```

where record count is a positive number;

the process verb specification varies for each process verb, and is discussed below;

The syntax of the output specification is:

```
[DDNAME output-file-name]
[FORMATTED]
[HEADING '<heading string>']
```

The output file name must be a valid JCL DD name, and must be present in the DBAUDIT JCL. The default output file name is "OUTPUT".

The "FORMATTED" keyword can be used for all verbs. It causes DBAUDIT to insert the 2 character ADABAS field names before each output field, and the string "ISN:" before the record ISN.

Detailed Syntax of Verbs

The detailed syntax of each process verb is as follows:

TOTAL

```
TOTAL <field-specification> [ AGAINST <field-specification> ]
```

TOTAL can only be unused on fields defined as unpacked, packed or binary length 4.

If the AGAINST clause is omitted, the TOTAL verb will output just 1 record, giving the sum of the field over all SELECT'ed records in the following format:

variable length record rdw (OS standard);

process-id (8 bytes);

field total (15 byte unpacked number).

If the AGAINST clause is specified, the TOTAL verb will sort the SELECT'ed records by the AGAINST field specification and generate 1 record for each value of the AGAINST field specification, in the following format:

variable length record rdw (OS standard);

process-id (8 bytes);

AGAINST field specification value (length of AGAINST field value);

field total (15 byte unpacked number).

COUNT

```
COUNT <field-specification> [ AGAINST <field-specification> ]
```

COUNT is used to count the number of distinct values of the field specification. When CONCAT is used, all NULL PE/MU occurrences after the first entry are excluded from the count. When AGAINST is used, ALL occurrences are included.

The COUNT verb always invokes a sort. One output record is generated for each distinct value of the field specification (including the AGAINST specification if any), in the following format:

variable length record rdw (OS standard);

process id (8 bytes);

field specification value;

optionally, AGAINST field specification;

field count (8 byte unpacked number).

UNIQUE and NULL UNIQUE

```
[ NULL ] UNIQUE <field-specification>
```

UNIQUE is used to verify that each SELECT'ed occurrence of the field specification, is unique across all other SELECT'ed records. Normal UNIQUE processing will exclude null occurrences from this process. However, the NULL UNIQUE verb will expand the process to incorporate null values.

The UNIQUE verb always invokes a sort. One output record is generated for each duplicated field value in the following format:

variable length record rdw (OS standard);

process id (8 bytes);

field specification value;

record ISN (10 byte unpacked number).

VALIDATE

```
VALIDATE where-criteria ENDVALIDATE
```

VALIDATE is used to check that each SELECTed record satisfies the logical conditions specified by the where-criteria (refer to the description of the SELECT statement for the syntax of the where criteria).

The VALIDATE verb operates on a record by record basis and does not require a sort. For each record in error, an output record is generated with the following format:

variable length record rdw (OS standard);

process id (8 bytes);

record ISN (10 byte unpacked number).

LIST and LISTALL

```
LIST  <field-specification>
| LISTALL
```

LIST/LISTALL is used to extract data from the ADASAV backup for each SELECTed record. The only difference between LIST and LISTALL is only in the area of MU and PE group processing.

LIST will terminate processing for a record when only null MU or PE values are left, but always prints at least one value even if it is NULL. LISTALL will process all specified occurrences of an MU or PE, even when those occurrences are null. E.g., given field P1, an MU with 5 non null occurrences, then:

```
LIST FIELD AA CONCAT FIELD P1(/*)
```

would generate 5 output records, with the same value of AA suffixed with the 5 non null values of field P1. However:

```
LISTALL FIELD AA CONCAT FIELD P1(/*)
```

would generate 191 output records, the last 186 all having the value of AA suffixed by the null value of field P1.

The format of output produced by LIST/LISTALL without the FORMATTED keyword is:

variable length record rdw (OS standard);

process id (8 bytes);

field specification value, not separated by delimiters;

record ISN (10 byte unpacked number).

(This format is most suitable for processing the output with another program such as a file loader or application specific program.)

The format of output produced by LIST/LISTALL with the FORMATTED keyword is:

variable length record rdw (OS standard);

process id (8 bytes);

field specification values, each preceded by:

```
<blank><ADABAS field name><colon>
```

record ISN (10 bytes, unpacked number) preceded by the string " ISN:".

The JOIN/JOINNULL statement

The JOIN statement is used to combine data from 2 separate records into 1 output record. Records selected by the previous encompassing SELECT statement form the left-hand-side (LHS) record set. The join field is then used to match these records in a 'one-to-many' or 'one-to-one' relationship with records from the right-hand-side (RHS) specification, (which may include a limiting WHERE clause).

JOIN can be used to combine records from the same file, or from different files. The JOINNULL form of the JOIN statement differs only from the JOIN with regards to how missing LHS or RHS records are processed. A missing LHS record occurs when a RHS join key value is not represented in the set of records on the LHS. Similarly, a missing RHS record occurs when a LHS join key value is not represented in the set of records on the RHS.

The JOIN statement will ignore (i.e., produce no output) for LHS or RHS records that do not match. JOINNULL, however, will replace the missing record field values with nulls (binary zeros) and will set the ISN of the missing record to zero before outputting the joined record.

Processing

The following possible join situations are processed as follows:

one LHS record matching one RHS record. This is standard one-to-one - 1 output record will be produced, combining data from both the LHS and RHS record.

one LHS record matching more than one RHS record. This is standard one-to-many - 1 output record will be produced for every RHS record, combining data from the LHS and from each RHS record.

no matching RHS record for a LHS record value. This is a missing RHS record. If the simple JOIN verb has been specified then no output is produced.

If the JOINNULL verb has been specified, then the RHS field values and set to null and the RHS record ISN is set to zero and the record is then output.

no matching LHS record for a RHS record value. This is a missing LHS record. If the simple JOIN verb has been specified then no output is produced.

If the JOINNULL verb has been specified, then the LHS field values are set to null and the LHS record ISN is set to zero and the record is then output.

more than one LHS record with the same matching key. This is duplicate LHS values, either many-to-many, many-to-one or many-to-none - if one or more matching RHS records exist, processing proceeds as for the above cases for the first LHS record. All other LHS records with the same key are ignored (i.e., produce no output).

The syntax of the JOIN statement is :

```
JOIN  | JOINNULL <joinid> [<estimate-specification>] :
                  <field specification> WITH
            FILE <fnr>  <field-expression>
                    [WHERE <where-expression>]
            [LIST [LHS <field-specification>]
                    [RHS <field-specification>]
            [<output-specification>] ;
```

the joinid is a character string of between 1 and 8 characters (A through Z, 0 through 9) which should be used to uniquely identify the process. All output produced from the application of the join will be preceded with the joinid.

the optional estimate specification may be used to override the default estimate that DBAUDIT would otherwise make of the number of records that may be sorted by the join.

The syntax of the estimate specification is:

```
ESTIMATE record-count
```

where record count is a positive number.

LIST specification can be used to add output fields from either the LHS or RHS records to the join key value, which is always output.

the optional output specification may be used to override the default output file name. The syntax of the output specification is:

```
[DDNAME output-file-name]
[FORMATTED]
[HEADING 'heading string']
```

The output file name must be a valid JCL DD name, and must be present in the DBAUDIT JCL. The default output file name is "OUTPUT".

The "FORMATTED" keyword causes DBAUDIT to insert the 2 character ADABAS field names before each output field.

The JOIN verb always invokes two sorts: one each for the LHS and RHS record selections. The format of JOIN output is:

variable length record rdw (OS standard)

join id (8 bytes)

LHS ISN (10 bytes, unpacked ISN) preceded by the string " LHSISN:"

RHS ISN (10 bytes, unpacked ISN) preceded by the string " RHSISN:"

LHS joining field value, followed by any fields specified following a LIST LHS clause, preceded by the string " LHSVAL:"

RHS joining field value, followed by any fields specified following a LIST RHS clause, preceded by the string " RHSVAL:"

If the FORMATTED keyword has been supplied as part of the JOIN output specification, then both the LHS and RHS field value lists will consist of the field values each preceded by:

```
<blank>ADABAS field name <colon>
```

Examples of JOINs:

```
SELECT FILE 43
JOIN J1: FIELD AA WITH FILE 44 FIELD HA
LIST LHS FIELD AB RHS FIELD HB CONCAT HC ;
```

all records from file 43 and 44 will be joined on a common key value, which is field AA in file 43 and field HA in file 44.

```
SELECT FILE 50 WHERE FIELD BA <> NULL ENDWHERE
JOIN J2: FIELD BA WITH FILE 50 FIELD CA
WHERE FIELD CA <> NULL ENDWHERE;
```

records from file 50 will be joined on a common key value, presumably representing distinct record types. The LHS set of records will consist of all records from file 50 with a non-null value of field BA. The RHS set of records will consist of all records from file 50 with a non-null value of field CA. It is possible that some records (i.e., some ISN's) will appear in both LHS and RHS sets.

as no explicit LIST LHS/RHS specification was made, the combined output record built from the join of the two records will contain only:

from the LHS record: ISN, field BA;

from the RHS record: ISN, field CA;

because JOIN (and not JOINNULL) has been specified, records without matching counterparts will be ignored.

the purpose of such a JOIN may be to discover all records having a supposedly exclusive key value. I.e., this JOIN will identify all records not adhering to the business rule that states "the same value may not appear in field BA and field CA", meaning that a field value should be unique across different fields. Of course, these fields could be in separate files, rather than being in the same file as demonstrated in this example.

```
SELECT FILE 60 WHERE FIELD AC = 'A' OR FIELD AC = 'Q'
                       ENDWHERE
JOINNULL J3: FIELD AA CONCAT AB(1-2)
      WITH FILE 72 FIELD DA(1-10)
            WHERE FIELD CD < '19900701' ENDWHERE
      LIST LHS FIELD AC CONCAT AD
      RHS FIELD CB CONCAT CD CONCAT CZ(4-5)
DDNAME JOIN1 HEADING 'Join 60 and 72'
FORMATTED ;
```

records from file 60 with a value of AC = 'A' or 'Q' will be matched with records from file 72 with a value of CD < '19900701'. The matching value is taken from field AA concatenated with the first 2 characters of AB for file 60, and is taken from the first 10 characters of field DA for file 72.

the combined output record built from the join of the two records will contain only:

from the LHS record: ISN, fields AA, AB (first 2 characters), AC and AD

from the RHS record: ISN, fields DA (first 10 characters), CB, CD and characters 4 through 5 from field CZ

because JOINNULL has been specified, records without matching counterparts will generate output records which have null values for the missing field values and zero for the missing ISN.

the relationship between files 60 and 72 based on the join field is assumed to be one-to-many or one-to-one: if duplicate values of field AA concatenated with the first 2 characters of BA exist in file 60, only the first occurring value will be processed and the other records with the same key value will be ignored.

The PRINT Statement

The PRINT statement is used to generate documentation at the start of a DBAUDIT output file, describing the contents of that file. Whereas the "HEADING" clause can be used to conveniently generate just 1 line of description at the start of an output file, PRINT statements can be used to create an arbitrary amount of description.

If both PRINT and HEADING are used to generate output to the same file, then the line(s) specified by the PRINT will appear first followed by the HEADING line.

The PRINT statement can appear at any place in the input stream where any other statement (such as SELECT and PROCESS) could appear. Output is written to the nominated DDNAME in the order it appears in the input stream. If the target DDNAME is omitted, it defaults to the DDNAME "OUTPUT".

The syntax of the PRINT statement is:

```
PRINT '<print-string>' [DDNAME<output-file-name> ] ;
```

The print-string must be enclosed in single quotes and cannot extend over a line boundary. The print-string must not include single quotes - i.e., 2 concatenated single quotes cannot be used to generate a single quote within the string.

Examples of the PRINT statement:

```
PRINT ' List of all male EMPLOYEES having date of birth'
                                    DDNAME LIST1;
PRINT ' in January or November'           DDNAME LIST1;
PRINT '                                          ' DDNAME  LIST1 ;
PRINT '     Field Descriptions:       '          DDNAME LIST1 ;
PRINT '     AE  -  Surname             '         DDNAME LIST1 ;
PRINT '     AC  -  First Name          '         DDNAME LIST1 ;
PRINT '     AA  -  Personnel ID     '            DDNAME LIST1 ;
PRINT '     AF  -  Marital Code     '            DDNAME LIST1 ;
PRINT '                  M: Married      'DDNAME LIST1 ;
PRINT '                   S: Single           ' DDNAME LIST1 ;
PRINT '                  D: Divorced       '    DDNAME IST1 ;
PRINT '                  W: Widowed    '         DDNAME LIST1 ;
PRINT '      AH - Date of Birth      '           DDNAME LIST1 ;
PRINT '                (yymmdd)     '           DDNAME LIST1 ;
PRINT '                                        ' DDNAME LIST1 ;

SELECT  FILE  2 WHERE FIELD AG = 'M' AND
            (FIELD AH(3-4) = 1 OR FIELD AH(3-4) = 11)

PROCESS L1: LIST FIELD AE CONCAT AC CONCAT AA
            CONCAT AF CONCAT AH DDNAME LIST1 ;
```

Chapter 5

EXEC Card Parameters

DBAUDIT Exec Parameters

DBAUDIT accepts user input from 2 sources:

> **PARMS file, containing DBAUDIT processing specifications, such as the SELECT, RULE, PROCESS and JOIN statements;**
>
> **JCL EXEC PARM parameters.**

The purpose of the EXEC PARMS are:

> **to supply the product protection code;**
>
> **to override default or zapped "constants" used to effect DBAUDIT processing.**

The defaults shown below are those applying to DBAUDIT as distributed. These defaults may be changed permanently by zapping module DB2AUDIT, see Chapter 6.

Description of parameters

This section describes each of the input parameters for DBAUDIT and gives abbreviations, default values and valid ranges.

BUFFERS

Abbreviation	**B**
Default	**100, i.e. allocate 400kb to internal buffers**
Purpose	**Set number of internal buffers used to communicate data from input tasks to sort and output tasks and from sort tasks to output tasks. Each buffer is 4KB in size. Increasing the number of buffers will reduce the space available for input and output file buffers, and for sort size below the 16MB line.**
Range	**20-50**
Considerations	**The DBAUDIT summary report shows the number of buffer waits that each input and sort task has suffered. If this number is "excessive" (say, 10% of the number of output blocks) then consider increasing the number of buffers.** **However, if DBAUDIT is short of virtual memory below the line (most likely at non XA sites attempting many concurrent sort tasks), consider reducing this value.**

CODE

Abbreviation	**C**
Purpose	**Provide product authorization code.**
Default	**none**
Range	**20 character alphanumeric string**
Considerations	**Required by all customers.**

EXTRASORT-PARMS

Abbreviation	**E**
Purpose	**Provide additional parameters to your installation sort routine, which may be required depending on installation default sort configuration. The parameter string supplied must always be enclosed within single quotes. The quotes will be removed and appended to the sort OPTION statement by DBAUDIT.**
Default	**None.**
Value	**Any valid sort option string - up to 40 bytes in length.**
Considerations	**The most likely use of this parameter will be to override the installation default for the specification of the number of dynamically allocated sort work files. Some installations will have installed their sort product with DYNALLOC=NO, which will inhibit the automatic dynamic allocations of sort work files and hence restrict DBAUDIT sorts to in-core sorts. This DBAUDIT parameter can be used in these cases to override this default, e.g.:** **PARM='E="DYNALLOC=(SYSDA,2) "'** **which tells the sort utility to dynamically allocate 2 sort work datasets on unit type of SYSDA. Refer to your sort users guide for complete details of the syntax of OPTION statement parameters.** **At some sites, the sort product has been configured to page fix its sort buffers (with the option EXCPVR=ALL). Running large numbers of concurrent sorts in a single DBAUDIT run may cause a large amount of memory to be page fixed and interfere with other workloads. The EXCPVR sort parameter can be overridden in such circumstances as follows:** **PARM='E="EXCPVR=NONE"'** **Note that the above examples use two single quotes within the EXEC card parm string to represent one single quote. This technique is used by the JCL interpreter to distinguish the embedded quote from the JCL parameter terminating quote.**

HIYEAR

Abbreviation	**HIY**
Purpose	**Set high year to be considered as valid as part of the year mask.**
Default	**2099**
Range	**0000 - 9999**
Considerations	**Must be greater than or equal to LOYEAR.**

LOSORTK

Abbreviation	**LOS**
Purpose	**Set a maximum on the amount of memory to be used per sort task below the 16MB line.**
Default	**64 (KB)**
Range	**60 - 3000 (KB)**
Considerations	**This parameter can be used to "encourage" the sort utility to allocate memory above the 16MB line. It should not be used by non XA and non ESA users - they should refer to the discussion of the MAXSORTK parameter. DBAUDIT will temporarily allocate all memory below the line excluding the amount of memory specified by this parameter before invoking the system sort procedure, and will release this memory after the sort has initialized. SORT usually requires some memory below the line, but will often try to allocate a substantial part of its memory below the line unless encouraged by a shortage of below the line memory to allocate from above the line.**

LOYEAR

Abbreviation	**LOY**
Purpose	**Set low year to be considered as valid as part of the year mask.**
Default	**1900**
Range	**0000 - 9999**
Considerations	**Must be less than or equal to HIYEAR.**

MAXRABNS

Abbreviation	**MAXR**
Purpose	**Set the maximum size of a RABN in the database (which will usually be the size of a data storage RABN). This parameter is used by the auxiliary input process which synthesizes ADASAV records from protection log or QDUMP input records.**
Default	**6000 (bytes)**
Range	**4000 - 32000**
Considerations	**Set to at least the size of the maximum of the associator and data storage block size.**

MAXSORTK

Abbreviation	**MAXSORTK**
Purpose	**Set a maximum on the amount of memory to be used per sort task.**
Default	**400 (KB)**
Range	**60 - 8000 (KB)**
Considerations	**This parameter determines the maximum amount of main memory that DBAUDIT will allow the system sort utility to use. Refer to LOSORTK parameter for the method of restricting the amount of memory used below the 16MB line. For XA and ESA sites, allowing the sort to take advantage of large amounts of above the 16MB line virtual memory will probably improve sort performance, unless severe paging impacts DBAUDIT and other jobs.** **DBAUDIT will allocate between MINSORTK and MAXSORTK memory to each sort task based on its estimate of the amount of data to be sorted and the sort record length and a fixed sort overhead.**

MAXSORTNUM

Abbreviation	**MAXSORTN**
Purpose	**Set a maximum on the number of concurrent sort tasks that may be initiated in any one execution of DBAUDIT.**
Default	**50**
Range	**6 - 196**
Considerations	**Each sort task requires both above and below the 16MB line memory, and for medium to large sorts, will require sort work datasets. Running a very large number of medium to large concurrent sorts may actually degrade DBAUDIT performance if the demand for memory and I/O resources causes paging and extended I/O times. The setting of this parameter to act as a "governor" on the resources used by DBAUDIT will depend on factors such as the physical machine configuration and the demands of concurrent jobs on the system.**

MINSORTK

Abbreviation	**MINSORTK**
Purpose	**Set a minimum on the amount of memory to be used per sort task.**
Default	**250 (KB)**
Range	**60 - 8000 (KB)**
Considerations	**This parameter determines the minimum amount of main memory that DBAUDIT will tell the system sort utility that it can use. Refer to LOSORTK and MAXSORTK parameters for further details.**

MINSORTREC

Abbreviation	**MNSORTREC**
Purpose	**Set a minimum on the estimate of records to be sorted by a sort task if no explicit estimate is provided on the RULE, PROCESS or JOIN specification.**
Default	**250000**
Range	**60 - 99999999 (records)**
Considerations	**DBAUDIT provides an estimate to each sort task of the number of records to be sorted.**
	If a user codes an ESTIMATE specification as part of the RULE, PROCESS or JOIN, then that value is used directly. When such an estimate is omitted, DBAUDIT uses the TOPISN of the file being processed to generate an estimate. This estimate may be too high (if stringent SELECT WHERE processing is in effect or the TOPISN value is a poor indicator of the number of records loaded), or it may be too low (if MU/PE processing causes many sort records to be generated for each input record). **The consequences of a too high estimate may be wasted virtual memory and sort work size (allocated as temporary disk space). A too low estimate may cause the sort to fail.** **In an attempt to stop sorts from failing due to a low estimate, this parameter sets a minimum on the record estimate that DBAUDIT will pass to the sort utility in the absence of an explicit user supplied ESTIMATE.**

SORTMSGCLASS

Abbreviation	**S**
Purpose	**Assigns the sysout class for messages produced by the system sort utility tasks invoked by DBAUDIT**
Default	**R (i.e., equivalent to //MSGS DD SYSOUT=R)**
Range	**Any valid sysout class**
Considerations	**During initial trialing and problem determination, set to your normal held TSO sysout class for convenient viewing. Although the volume of sysout message information is usually small, some sites may later wish to redirect this output to a class which is automatically purged. DBAUDIT commonly invokes multiple sort sub tasks, each of which will produce a sort messages file. The presence of the //SORTDIAGS DD statement in the DBAUDIT JCL will cause the sort utility to produce slightly more information to the sort messages file.**

UEX1

Abbreviation	**UEX1**
Purpose	**Specifies the name of user exit 1 (the output task user exit) to be invoked by DBAUDIT before each record is output.**
Default	**none (i.e., blanks, meaning no user exit)**
Range	**A valid user exit 1 load module name.**
Considerations	**Refer to Appendix A for details of user exit functions. Note that if a user exit 1 has been zapped into DBAUDIT as a default, then it cannot be removed by specifying a user exit 1 parameter value of blanks. Instead, a dummy "do nothing" routine such as IEFBR14 would need to be specified to achieve the desired result of inhibiting any user exit processing.**

VERBOSE

Abbreviation	**V**
Purpose	**Determines whether some suppressible DBAUDIT job and system log) messages are produced or suppressed.**
Default	**Y (i.e., produce all messages).**
Range	**either N or Y**
Considerations	**During initial trialing and problem determination, set to Y to obtain more information about DBAUDIT processing. If the messages prove to be distracting, set to N.**

Chapter 6

Changing DBAUDIT Parameter Defaults

Zapping Defaults

DBAUDIT is distributed with default settings likely to be applicable at most sites. Most of these defaults can be changed at run time by the specification of an EXEC card PARM string. The defaults can be changed "permanently" by zapping load module DB2AUDIT. In addition, 2 parameters effecting the allocation of internal tables cannot be changed at run time, but can changed by zaps.

The CODE= parameter is used for license verification and trial management. This code can be zapped with a special zap provided by CCA Software Pty Ltd.

Before zapping DBAUDIT, we strongly recommend that a backup copy of the module is made to expedite recovery in the case of an error.

The table on the following page shows the parameters that may be zapped, their format and offset within the DBAUDIT module. Before zapping any parameters, refer to Chapter 5 for the purpose of the parameter and considerations for its setting.

The unnamed zap areas at the bottom of the following table are used to determine the sizes of internal tables reserved by the parse routine to hold intermediate parsing results and structures used to control processing activity.

Parameter	Offset	Default Value	Sample Zap
BUFFERS	**+0090**	**100**	**VER 0090 0064 REP 0090 0032 set to 50**
CODE			**Provided by CCA Software for SOLD licences**
EXTRASORTPARMS	**+00B4**	**-none-**	**VER00B4 40404040404040404040 VER00BE 4040404040404040 REP00B4C4E8D5C1D3D3D6C37E4D REP00BEE2E8E2C4C16 BF25D40 set 'DYNALLOC=(SYSDA,2)' zap area is 40 bytes long**
HIYEAR	**+00A0**	**2099**	**VER 00A0 F2F0F9F9 REP 00A0 F2F0F0F0 set to year 2000**
LOSORTK	**+0096**	**64 kb**	**VER 0096 0040 REP 0096 0080 set to 128 kb**
LOYEAR	**+009C**	**1900**	**VER 009C F1F9F0F0 REP 009C F1F9F7F0 set to 1970**
MAXRABNS	**+008C**	**6000**	**VER 008C 1770 REP 008C 2000 set to 8192**
MAXSORTK	**+0092**	**400 kb**	**VER 0092 0190 REP 0092 0800 set to 2 Mb**
MAXSORTNUM	**+008E**	**50**	**VER 008E 0032 REP 008E 0064 set to 100**
MINSORTK	**+0094**	**250 kb**	**VER 0094 00FA REP 0094 0080 set to 128 kb**

Parameter	Offset	Default Value	Sample Zap
MINSORTREC	+00A4	250000	VER 00A4 0003D090 REP 00A4 00004000 set to 16000
SORTMSGCLASS	+0098	R	VER 0098 D9 REP 0098 E7 set to X
UEX1	+00AC	NONE	VER 00AC 4040404040404040 REP 00AC E4C5E7F1E2C1D4D7 set to UEX1SAMP
VERBOSE	+0099	Y	VER 0099 E8 REP 0099 D5 set to N
(not named)	+00A8	4 kb	VER 00A8 1000 REP 00A8 3000 set to 12 kb
(not named)	+00AA	8 kb (small)	VER 00AA 2000 REP 00AA 4000 set to 16 kb (large

Chapter 7

Problem Determination Guide

Problem Diagnosis

DBAUDIT has built-in diagnostic code to aid problem determination. All significant events are written to the job and system log. Use the following checklist should a problem occur.

1. **Are the input files to DBAUDIT correct:**

   make sure that the dataset(s) input to DDSAVE1, DDSAVE2, etc. are the output from the same ADASAV backup;

   if protection log input is being provided, make sure that the protection log covers the time of the ADASAV;

   if a QDUMP incremental backup is being input, make sure that the ADASAV backup was run at the start of the QDUMP session used to produce the incremental backup (i.e., that the QDUMP incremental matches the ADASAV complete backup).

2. **Eliminate "obvious" environmental errors, such as:**

   System abend codes:

   322 - CPU time exceeded (increase unless problem seems to be job looping);

   222 - operator cancel;

   722 - maximum output lines exceeded;

   878 - region too small (increase if possible);

   913 - access authorization errors (RACF or ACF2);

   37 - output datasets too small.

3. **Set VERBOSE=Y in the EXEC PARM to force all DBAUDIT messages.**

4. Include DD statement:

```
//SORTDIAGS DD SYSOUT=*
```

This JCL card will force full sort diagnostics to be printed on the job log.

5. Include DD statement:

```
//DDMESS DD SYSOUT=*
```

This JCL statement will enable the output of DBAUDIT diagnostic abend information.

6. If requested by DBAUDIT support, include DD statement:

```
//SYSABEND DD SYSOUT=*
```

This statement will enable the output of system abend information.

Chapter 8

Messages and Codes

Introduction

DBAUDIT writes messages describing its processing status to the job and system log. All messages have the following format:

```
DB2AUDIT    nnnt        xxxxxx...
```

where:

nnn is the message number.

t is the message type:

I - information;

W - warning;

E - fatal error;

xxxx is the message body.

Fatal error messages will cause the DBAUDIT job to abend.

Some information messages will be suppressed if the VERBOSE parameter is set to 'N' (VERBOSE=N).

Messages

This section details all messages produced by DBAUDIT. There is then an explanation, and if required any user action.

001I	**VERSION .... ZAP LEVEL: ....**	
	Meaning	**DBAUDIT displays current version and zap level information on the system log.**
	Action	**Verify that the version and zap level of DBAUDIT is as expected.**
002E	**SUMMARY FILE OPEN FOR OUTPUT FAILED**	
	Meaning	**DBAUDIT writes a run summary to the file 'SUMMARY'. This file could not be opened.**
	Action	**Supply the //SUMMARY DD ... JCL statement. The SUMMARY file is usually assigned to SYSOUT, but may be assigned to a dataset. Its DCB attributes should be:** **RECFM=FBA,LRECL=133,BLKSIZE=2660, BUFNO=1** **Blocksize and Bufno may be varied to site requirements.**
010I	**PLOG OR QDUMP DD CANNOT BE OPENED - ADASAV DUMP IS THE SOLE DATA SOURCE**	

	Meaning	**DBAUDIT will merge protection log or QDUMP incremental backup data with an online ADASAV to enable DBAUDIT to operate on a consistent and complete backup. However, neither the PLOG or QDUMP file was specified in the JCL, and hence this DBAUDIT run will only process the ADASAV backup data.**
	Action	**None (this message will be suppressed if the VERBOSE parameter is set to N).**
011I	**MERGING RECORDS FROM PROTECTION LOG**	
	Meaning	**DBAUDIT will be merging the supplied protection log with the supplied ADASAV backup to provide a consistent and complete backup image.**
	Action	**None.**
012I	**MERGING RECORDS FROM QDUMP INCREMENTAL**	
	Meaning	**DBAUDIT will be merging the supplied QDUMP backup with the supplied ADASAV backup to provide a consistent and complete backup image.**
	Action	**None.**
017E	**TOO MANY INPUT LINES**	
	Meaning	**DBAUDIT can process an input specification file of up to 800 lines.**
	Action	**Reduce the size of the input specification file supplied to the //PARMS DD file.**

020E	**OPEN FOR DDSAVE1 FAILED IN SORTAUX**	
	Meaning	**DBAUDIT could not open file DDSAVE1.**
	Action	**Make sure that DDSAVE1 has been correctly defined in the DBAUDIT JCL, and that the job has authority to read the dataset. Contact DBAUDIT support.**
021E	**UNEXPECTED END OF FILE DDSAVE1 IN SORTAUX**	
	Meaning	**DBAUDIT received an unexpected end of file whilst reading the ADASAV backup.**
	Action	**Make sure that a valid ADASAV backup has been supplied to DBAUDIT via the DDSAVE1, DDSAVE2, ... files. Contact DBAUDIT support.**
039I	**EXTRACT/SORT PHASE COMPLETED NORMALLY**	
	Meaning	**DBAUDIT has completed the preparatory extract and sort of protection log or QDUMP incremental backup data.**
	Action	**None.**
050E	**OPEN FOR DDSAVE1 FAILED IN RDFCBFDT**	
	Meaning	**DBAUDIT could not open file DDSAVE1.**

	Action	**Make sure that DDSAVE1 has been correctly defined in the DBAUDIT JCL, and that the job has authority to read the dataset. Contact DBAUDIT support.**
051E	**OPEN FOR SORTOUT FAILED IN RDFCBFDT**	
	Meaning	**DBAUDIT could not open file SORTOUT. DBAUDIT is attempting to read the sorted extract of PLOG/QDUMP records.**
	Action	**Make sure that SORTOUT has been correctly defined in the DBAUDIT JCL, and that the job has authority to read the dataset. Contact DBAUDIT support.**
052E	**EOF WHILE READING GCB IN RDFCBFDT**	
	Meaning	**DBAUDIT received an unexpected end of file whilst reading the ADABAS General Control Block.**
	Action	**Make sure that a valid ADASAV backup has been supplied to DBAUDIT via the DDSAVE1, DDSAVE2, ... files. Contact DBAUDIT support.**
053E	**EOF WHILE READING FCBS IN RDFCBFDT**	
	Meaning	**DBAUDIT received an unexpected end of file whilst reading the ADABAS File Control Block.**
	Action	**Make sure that a valid ADASAV backup has been supplied to DBAUDIT via the DDSAVE1, DDSAVE2, ... files. Contact DBAUDIT support.**

054E	**NO FCB FOR FILE nnn**	
	Meaning	**DBAUDIT could not find the FCB for file nnn.**
	Action	**Make sure that a valid ADASAV backup has been supplied to DBAUDIT via the DDSAVE1, DDSAVE2, ... files. Make sure that the file nnn was actually loaded in the ADABAS database at the time the ADASAV backup was taken. If it was not, then remove references to the file from the DBAUDIT input specifications.**
055E	**EOF WHILE READING FDTS IN RDFCBFDT**	
	Meaning	**DBAUDIT received an unexpected end of file whilst reading the ADABAS Field Definition Tables.**
	Action	**Make sure that a valid ADASAV backup has been supplied to DBAUDIT via the DDSAVE1, DDSAVE2, ... files. Contact DBAUDIT support.**
056E	**NO FDT FOR FILE nnn**	
	Meaning	**DBAUDIT could not find the FDT for file nnn.**
	Action.	**Make sure that a valid ADASAV backup has been supplied to DBAUDIT via the DDSAVE1, DDSAVE2, ... files. Make sure that the file nnn was actually loaded in the ADABAS database at the time the ADASAV backup was taken. If it was not, then remove references to the file from the DBAUDIT input specifications.**

057E	**GCB NOT FOUND OR INVALID**	
	Meaning	**DBAUDIT expected to find a valid GCB at the start of the ADASAV backup, but didn't.**
	Action	**Make sure that a valid ADASAV backup has been supplied to DBAUDIT via the DDSAVE1, DDSAVE2, ... files, and that these files are assigned correctly to the datasets produced by ADASAV - in particular, this error may indicate that the dataset assigned to DDSAVE1 is not the first dataset component of the backup. Contact DBAUDIT support.**
059E	**NO PROCESSING SPECIFIED**	
	Meaning	**No file SELECTs were supplied to DBAUDIT.**
	Action	**Rerun DBAUDIT providing some input cards.**
080E	**TOO MANY DIFFERENT OUTPUT FILES REQUESTED**	
	Meaning	**DBAUDIT has an internal limit of 200 output subtasks (1 for each output file). This limit has been exceeded.**
	Action	**Reduce the number of different output files to 200.**
081I	**OUTPUT PROCESSOR INITIALIZED**	
	Meaning	**DBAUDIT has initialized all output tasks.**
	Action	**None.**

082I	**OUTPUT PROCESSOR TERMINATING**	
	Meaning	**DBAUDIT has completed all output processing.**
	Action	**None.**
090E	**OUTTASK LOST IN INITIALIZE**	
	Meaning	**An output task initialization has failed.**
	Action	**Contact DBAUDIT support.**
092E	**OPEN FAILED FOR: dddddddd**	
	Meaning	**An output task could not open file dddddddd.**
	Action	**Make sure that the file has been correctly defined in the DBAUDIT JCL, and that the job has authority to write the dataset. Contact DBAUDIT support.**
093I	**dddddddd INITIALIZED**	
	Meaning	**The output task writing file dddddddd is initialized.**
	Action	**None (this message will be suppressed if the VERBOSE parameter is set to N).**
094I	**dddddddd CLOSE**	
	Meaning	**The output task writing file dddddddd has closed the file.**
	Action	**None (this message will be suppressed if the VERBOSE parameter is set to N).**

100E	**DDSAVE1 FILE MISSING**	
	Meaning	**DBAUDIT could not located the JCL definition of file DDSAVE1.**
	Action	**Supply the file definition of DDSAVE1 to the DBAUDIT JCL. Make sure that any other output files from the ADASAV backup are also correctly specified and rerun DBAUDIT.**
101I	**INPUT PROCESSOR INITIALIZED**	
	Meaning	**DBAUDIT has initialized all input tasks.**
	Action	**None**
102I	**INPUT PROCESSOR TERMINATING**	
	Meaning	**DBAUDIT has completed all input processing.**
	Action	**None.**
110E	**INTASK LOST IN INITIALIZE**	
	Meaning	**An input task initialization has failed.**
	Action	**Contact DBAUDIT support.**
113E	**OPEN FAILED FOR: dddddddd**	
	Meaning	**An input task could not open file dddddddd.**
	Action	**Make sure that the file has been correctly defined in the DBAUDIT JCL, and that the job has authority to read the dataset. Contact DBAUDIT support.**

114I	**ddddddd INITIALIZED**	
	Meaning	**The input task reading file ddddddd is initialized.**
	Action	**None (this message will be suppressed if the VERBOSE parameter is set to N).**
115I	**ddddddd CLOSE**	
	Meaning	**The input task reading file ddddddd has closed the file.**
	Action	**None (this message will be suppressed if the VERBOSE parameter is set to N).**
120W	**WARNING EOF FOR: ddddddd ON FIRST RECORD**	
	Meaning	**The input task reading file ddddddd was told that the input file was empty.**
	Action	**Verify that this is reasonable - it would normally be unusual for an ADASAV backup dataset to be completely empty.**
160E	**NO FREE BUFFERS (WRITACUM)**	
	Meaning	**The accumulated total processor cannot find an output buffer. This is an internal error.**
	Action	**Contact DBAUDIT support.**
200E	**SORT REQUIREMENTS EXCESSIVE**	

	Meaning	**Too many concurrent sort tasks would be required to process the input specifications. DBAUDIT limits the maximum number of concurrent sorts to the value of the SORTMAX parameter, or to 200, whichever is the lesser.**
	Action	**Consider either increasing SORTMAX or removing some of the requests requiring sorting (all RULEs, all COUNT and UNIQUE and some TOTAL PROCESSES).**
201I	**SORT PROCESSOR INITIALIZED**	
	Meaning	**DBAUDIT has initialized all sort tasks.**
	Action	**None.**
202I	**SORT PROCESSOR TERMINATING**	
	Meaning	**DBAUDIT has completed all sort processing.**
	Action	**None.**
203I	**NO SORT PROCESSING REQUIRED**	
	Meaning	**This DBAUDIT run does not require any sort services.**
	Action	**None.**
230E	**SORTP LOST IN INITIALIZE**	
	Meaning	**Sort PROCESS task initialization has failed.**
	Action	**Contact DBAUDIT support.**

231E	**SORTP UNKNOWN VERB**	
	Meaning	**Internal error.**
	Action	**Contact DBAUDIT support.**
234I	**SORT xxxx INITIALIZED**	
	Meaning	**The PROCESS sort task xxxx has been initialized.**
	Action	**None (this message will be suppressed if the VERBOSE parameter is set to N).**
235I	**SORT xxxx DONE**	
	Meaning	**The sort process is complete.**
	Action	**None.**
238I	**SORT xxxx INPUT DONE**	
	Meaning	**The PROCESS sort task xxxx has accepted all input records.**
	Action	**None (this message will be suppressed if the VERBOSE parameter is set to N).**
239E	**SORT xxxx FAILED**	
	Meaning	**A sort initiated to process a COUNT or TOTAL DBAUDIT PROCESS has failed, because the operating system sort utility has completed abnormally.**

	Action	**Examine the sort messages associated with this sort task (which will be written to a DDNAME of xxxxMSGS, e.g. S003MSGS) for the cause. The most likely cause is that the capacity of the sort utility exceeded due to a poor estimation by DBAUDIT of the number of records to be sorted. If this is the case, specify the ESTIMATE clause in the specification of the RULE. Otherwise, diagnose problem based on error indicated by the sort message output. (Include the DD statement:** **//SORTDIAG DD DUMMY** **in the DBAUDIT JCL to cause the SORT utility to produce maximum diagnostics). Contact DBAUDIT support.**
240E	**SORTR LOST IN INITIALIZE**	
	Meaning	**A sort RULE controller task initialization has failed.**
	Action	**Contact DBAUDIT support.**
241I	**SORT xxxx OUTPUT STARTED**	
	Meaning	**The PROCESS sort task xxxx has completed sorting and is now outputting records.**
	Action	**None (this message will be suppressed if the VERBOSE parameter is set to N).**
242E	**REGION TOO SMALL TO INITIATE SORT**	

	Meaning	**The operating sort utility could not be initiated because too little memory below the 16MB line was available.**
	Action	**Tune the DBAUDIT sort parameters to reduce sort size (especially below the line sort size (LOSORT) if running under 31-bit addressing),** **OR** **Reduce number of sorts required in the DBAUDIT run by removing some RULE or PROCESS COUNT input requests.**
250E	**DYNALLOC OF SORTMSG FILE FAILED**	
	Meaning	**DBAUDIT dynamically allocates the sort messages files used by the sort subtasks created during processing. The dynamic allocation of a sort message file has failed.**
	Action	**Check that the sort message class being used by DBAUDIT is valid at your site (see parameter summary produced at the start of the DBAUDIT run). Contact DBAUDIT support.**
270E	**SORTLR LOST IN INITIALIZE**	
	Meaning	**A sort task initialization has failed.**
	Action	**Contact DBAUDIT support.**
274I	**SORT xxxxp INITIALIZED**	
	Meaning	**The RULE sort task xxxx has been initialized.**

	Action	**None (this message will be suppressed if the VERBOSE parameter is set to N).**
275I	**SORT xxxxp DONE**	
	Meaning	**The RULE sort task xxxx is completed.**
	Action	**None (this message will be suppressed if the VERBOSE parameter is set to N).**
276I	**SORT xxxxp INPUT DONE**	
	Meaning	**The RULE sort task xxxx has accepted all input records.**
	Action	**None (this message will be suppressed if the VERBOSE parameter is set to N).**
279E	**SORT xxxxp FAILED**	
	Meaning	**A sort initiated to process the left or right hand side of a DBAUDIT RULE has failed, because the operating system sort utility has completed abnormally.**

	Action	**Examine the sort messages associated with this sort task (which will be written to a DDNAME of xxxxMSGS, e.g. S003MSGS) for the cause. The most likely cause is that the capacity of the sort utility exceeded due to a poor estimation by DBAUDIT of the number of records to be sorted. If this is the case, specify the ESTIMATE clause in the specification of the RULE. Otherwise, diagnose problem based on error indicated by the sort message output. (Include the DD statement:** **//SORTDIAG DD SYSOUT*** **in the DBAUDIT JCL to cause the SORT utility to produce maximum diagnostics). Contact DBAUDIT support.**
280E	**INTERNAL ERROR VERB IN OUTPROC**	
	Meaning	**Internal error.**
	Action	**Contact DBAUDIT support.**
281I	**SORT xxxxp OUTPUT STARTED**	
	Meaning	**The RULE sort task xxxx has completed sorting and is now outputting records.**
	Action	**None (this message will be suppressed if the VERBOSE parameter is set to N).**
282E	**REGION TOO SMALL TO INITIATE SORT**	

	Meaning	**The operating system sort utility could not be initiated because too little memory below the 16MB line was available.**
	Action	**Tune the DBAUDIT sort parameters to reduce sort size (especially below the line sort size (LOSORT) if running under 31-bit addressing),** **OR** **Reduce number of sorts required in the DBAUDIT run by removing some RULE or PROCESS COUNT input requests.**
350E	**BAD FDT - NO FIELDS**	
	Meaning	**DBAUDIT has found a corrupt FDT on the ADASAV backup**
	Action	**Check DBAUDIT input to ensure that all files being processed actually exist and are not corrupt in ADABAS. Contact DBAUDIT support.**
399E	**FIELD ERROR(S) DETECTED**	
	Meaning	**DBAUDIT has detected field level errors in the input specifications. Fields have been supplied which do not exist in the ADABAS field definition table for the file.**
	Action	**Refer to the DBAUDIT summary output for detailed error messages.**
402E	**FIELD SPECIFICATION ERROR(S) DETECTED**	

	Meaning	**DBAUDIT has detected field level errors in the input specifications. Fields have been supplied which do not match the ADABAS field definition table for the file.**
	Action	**Refer to the DBAUDIT summary output for detailed error messages.**
464E	**INCORRECT PARMCARD SUPPLIED**	
	Meaning	**Product Code has not been supplied.**
	Action	**Supply the product code obtainable from your DBAUDIT distributor as the value for the 'CODE=' parameter on the DBAUDIT EXEC parameter.**
465E	**CODEWORD DOES NOT MATCH THIS PRODUCT**	
	Meaning	**Product Code has been supplied but is incorrect.**
	Action	**Supply the product code obtainable from your DBAUDIT distributor as the value for the 'CODE=' parameter on the DBAUDIT EXEC parameter.**
467I	**SOFTWARE LICENCE EXPIRES IN nn DAYS**	
	Meaning	**The product code supplied as a parameter to DBAUDIT is only valid for nn more days. After that, DBAUDIT will terminate with message 468E.**
	Action	**Obtain a new product code from your DBAUDIT distributor. You will be able to use DBAUDIT for the next nn days.**
468E	**SOFTWARE LICENCE HAS EXPIRED**	

	Meaning	**The product code supplied as a parameter to DBAUDIT has expired. DBAUDIT cannot be used until a new product code is supplied.**
	Action	**Obtain a new product code from your DBAUDIT distributor.**
469E	**INVALID CODEWORD SUPPLIED**	
	Meaning	**Product Code is invalid.**
	Action	**Supply the product code obtainable from your DBAUDIT distributor as the value for the 'CODE=' parameter on the DBAUDIT EXEC parameter.**
470E	**SOFTWARE NOT LICENCED FOR THIS CPU**	
	Meaning	**Product Code is invalid for this CPU id - your site has probably installed a new CPU, and the DBAUDIT code which checks product code against CPU serial number has detected the change.**
	Action	**Supply the product code obtainable from your DBAUDIT distributor as the value for the 'CODE=' parameter on the DBAUDIT EXEC parameter.** **Be sure to quote your new CPU serial number to your DBAUDIT distributor**
490E	**EXEC CARD PARAMETER ERROR: .....**	
	Meaning	**DBAUDIT has found an error in EXEC card parameter syntax starting at the area printed at the end of this message.**
	Action	**Correct parameters and rerun DBAUDIT.**

491W	**PARAMETER VALUE INVALID - IGNORED**	
	Meaning	**DBAUDIT has found an error in the value supplied for an EXEC card parameter. The DBAUDIT run will continue, using a default value for the parameter.**
	Action	**Check your parameters against the DBAUDIT parameter summary produced by DBAUDIT at the start of the run Correct parameters and if necessary, rerun DBAUDIT.**
492E	**UEX1 COULD NOT BE LOADED: xxxxxxxx**	
	Meaning	**DBAUDIT could not load the requested user exit1 (xxxxxxxx) and will now terminate the job.**
	Action	**Check the specification of the UEX1 parameter. Make sure that the userexit is in the DBAUDIT STEPLIB, and that it is executable. Rerun DBAUDIT.**
493E	**SOFTWARE LICENCE HAS EXPIRED**	
	Meaning	**Your trial period of DBAUDIT has expired.**
	Action	**Contact your DBAUDIT distributor to extend the trial or obtain a permanent license.**
500E	**SYNTAX ERROR(S) DETECTED**	
	Meaning	**DBAUDIT has detected syntax errors in the input specifications.**
	Action	**Refer to the DBAUDIT summary output for detailed error messages.**

501E	**PARMS DD OPENED FOR INPUT FAILED**	
	Meaning	**DBAUDIT could not open the PARMS file, which contains the DBAUDIT input specifications.**
	Action	**Supply input specifications using the //PARMS DD ... JCL statement. Parameters may be "in stream", or from a dataset with DSORG= PS,RECFM= FB. DBAUDIT only processes the first 72 bytes of each record.**
504E	**EXPECTED PART 2 RECORD NOT FOUND**	
	Meaning	**While processing protection log records, DBAUDIT has found a sequence error on the protection log file.**
	Action	**Ensure the protection log supplied to DBAUDIT covers the period of the online ADASAV backup. Contact DBAUDIT support.**
505E	**UNEXPECTED PART 2 RECORD NOT FOUND**	
	Meaning	**While processing protection log records, DBAUDIT has found a sequence error on the protection log file.**
	Action	**Ensure the protection log supplied to DBAUDIT covers the period of the online ADASAV backup. Contact DBAUDIT support.**
506E	**EOF ON PLOG BEFORE SYN2 FOUND**	

	Meaning	**While processing protection log records, DBAUDIT has not been able to find the SYN2 record checkpointing the end of the online ADASAV backup.**
	Action	**Ensure the protection log supplied to DBAUDIT covers the period of the online ADASAV backup. Contact DBAUDIT support.**
507E	**EOF ON PLOG BEFORE SYN1 FOUND**	
	Meaning	**While processing protection log records, DBAUDIT has not been able to find the SYN1 record checkpointing the start of the online ADASAV backup.**
	Action	**Ensure the protection log supplied to DBAUDIT covers the period of the online ADASAV backup. Contact DBAUDIT support.**
710E	**INTERNAL ERROR - LOGIC OPERATION**	
	Meaning	**Internal error.**
	Action	**Contact DBAUDIT support.**
711E	**INTERNAL ERROR - UNKNOWN MASK TYPE IN PROCLOG**	
	Meaning	**Internal error.**
	Action	**Contact DBAUDIT support.**
900I	**NORMAL END**	
	Meaning	**DBAUDIT processing has completed normally, without any detected errors.**
	Action	**Process DBAUDIT output.**

920E	**AUXILARY INPUT SORT FAILED**	
	Meaning	**The sort of protection log or QDUMP records failed, probably due to lack of sort work space.**
	Action	**Check the SORTMSGS/SYSOUTS output from the sort, which will identify the error. Correct the problem and rerun.**
931E	**NO SDWA PASSED TO ABEND EXIT**	
	Meaning	**DBAUDIT abended and problem determination exits could not complete due to absence of an important abend diagnostic system control block, the SDWA.**
	Action	**Rerun DBAUDIT in a larger region. Contact DBAUDIT support.**
932E	**DDMESS OPEN FAILED IN ESTAE EXIT**	
	Meaning	**DBAUDIT abend and problem determination exits could not open the diagnostic output file, DDMESS.**
	Action	**Supply the //DDMESS DD statement in the DBAUDIT JCL and rerun the job.**
933E	**ABEND EXIT ENTERED**	
	Meaning	**DBAUDIT abend and problem determination facility has been invoked. It will attempt to write vital diagnostic information to the file, DDMESS. The job will then abnormally terminate.**

	Action	**Some errors may be self explanatory and preceded by other messages which diagnose the problem. If this is not the case, contact DBAUDIT support.**
950E	**TASK ABEND DETECTED**	
	Meaning	**DBAUDIT abend and problem determination facility has been invoked following the abnormal completion of a DBAUDIT subtask. The job will now be abnormally terminated with user abend code 950. If the SYSUDUMP or SYSABEND files are present in the DBAUDIT JCL, a dump will be produced.**
	Action	**See message 933E.**
955E	**xxxxxxx TABLE OVERFLOWED**	
	Meaning	**DBAUDIT parameter parsing failed because the space allocated to table xxxxxxxx was exceeded.**
	Action	**Reduce the size and contents of the DBAUDIT parameter file, by removing PRINT statements or SELECT/RULE parameters etc.,** **OR** **Refer to chapter 6 for instructions on increasing the size of the DBAUDIT parse table via a zap.**
960E	**PLOG/QDUMP SPECIFIED FOR DIRECT DB READ**	

	Meaning	**A PLOG or QDUMP DD statement was present in the DBAUDIT JCL, although DBAUDIT will be processing the database direct from disk, rather than from an ADASAV backup. (The DD statement for DDASSO was present in the DBAUDIT JCL.)**
	Action	**PLOG or QDUMP input can only be provided when DBAUDIT is processing against an ADASAV backup.**
961E	**DDDATA NOT FOUND**	
	Meaning	**Although DD statement DDASSO was present in the DBAUDIT JCL (indicating that the database was to be read direct rather than from an ADASAV backup), DD statement DDDATA was not found.**
	Action	**Supply both DDASSO and DDDATA when running DBAUDIT direct against the database.**
962E	**DIRECT DATABASE READING NOT SUPPORTED**	
	Meaning	**DBAUDIT V1.0 does not support direct reading of the data base.**
	Action	**Supply an ADASAV backup using the DDSAVE1, DDSAVE2, ... DD statements.**

DBAUDIT USER EXIT 1

Overview

User exit 1 can be used to edit and suppress records to be output from DBAUDIT. It provides sites with the flexibility to customize DBAUDIT output or even produce their own output formats.

DBAUDIT attaches 1 subtask to generate output to each output dataset specified in the DBAUDIT input specification "DDNAME" clause. If no output ddname is explicitly assigned for a PROCESS or RULE, then DBAUDIT defaults the ddname to "OUTPUT".

Hence, 1 or more output subtasks may be attached by DBAUDIT. Each of these will operate independently and asynchronously. Only 1 copy of the user exit is loaded. Hence, the user exit must be reentrant and most importantly, be aware of issues that arise when performing processing in a multi-tasking and asynchronous environment.

Only experienced assembler programs should attempt to write a DBAUDIT user exit. If the user exit is to perform file output or other operations that could cause interaction between separate invocations of the user exit, the task of writing the exit should only be attempted by a programmer familiar with multi tasking issues and concepts such as file sharing, ENQ/DEQ logic and other serialization techniques.

A sample user exit 1 is distributed with DBAUDIT (member UEX1SAMP in the installation source dataset). This user exit is intended to demonstrate the interface specifications and some of the simple types of processing that could be performed.

The user exit does not need to be coded in assembler. However, the restrictions of strict reentrancy, standard linkage conventions and the requirements of low overhead and possible serialization and inter-task communications probably make assembler the most suitable environment.

Detailed Technical Issues

On entry to the user exit, standard register conventions apply:

```
R1    -     address of parameter list (last entry flagged with
x'80')
RD    -     address of caller's save area
RE    -     return address
RF    -     user exit entry point address
```

The user exit must save and restore all registers, except for RF, whose content on return to DBAUDIT determines whether DBAUDIT will write the record:

If RF is 0, DBAUDIT will write the record otherwise, DBAUDIT will not write the record.

The user exit is entered in 24 bit mode, and must return control in 24 bit mode. All parameters are passed as 24 bit address. The parameter list passed to the user exit is:

+0 Address of the record about to be written. The record starts with a binary halfword inclusive length. The user exit is called one final time for each output subtask just before the output file is closed - on this call, this parameter is binary zero.

+4 Address of 8 byte, blank padded DDNAME - this is the filename to which the invoking task is outputting.

+8 Address of the DCB to which the invoking task is outputting. This DCB will be open (even on the final call). The DCB is a QSAM output DCB, opened in 'PUT LOCATE' mode:

```
MACRF=PL,DSORG=PS,RECFM=VB).
```

+12 Address of a GLOBAL WORD which may be used by the user exit - extreme care must be taken if the user exit decides to initialize or update this field, as several instances of the user exit may be concurrently active and attempting this action. Hence this field should be updated within an ENQ or with the COMPARE and SWAP instruction.

+16 Address of a task related word which may be used by the user exit. As only a single task (and user exit) has access to this word, serialization is not required.

+20 Address of a single byte containing X’00' for a "normal" detail line, X’01' for a heading line.

The 1st parameter points to the potential output record, with the following format:

```
+0    -     halfword inclusive length
When 6th parameter is X'00' (detail line) then:
+2    -     RULE of PROCESS name - 8 characters, blank padded;
+10 - body of the output record.
When 6th parameter is X'01' (headingline) then:
+2    -     heading line text
```

The user exit MUST NOT alter the first parameter, or the memory to which it is pointing (the output record) IN ANY WAY. The options the user exit has include:

accept the record, return with RF = 0;

suppress the record, return with RF non zero;

write a record to the DBAUDIT output file, probably based on the supplied input record, and then return with RF non zero to suppress DBAUDIT writing the record;

write a record to another output file, probably based on the supplied input record and maybe some table lookups, etc. and maybe returning to DBAUDIT with RF non zero;

accumulate statistics, produce messages to the log or for inclusion at the end of the DBAUDIT output file.

The user exit can perform any required processing. However, it is absolutely essential that potential authors of user exits understand the issues of multi tasking. A particular instance of these issues is file sharing: imagine a user exit is required to write data gleaned from DBAUDIT output to a separate output dataset. If this dataset will only be opened by 1 task, then standard reentrant coding techniques will suffice.

However, if DBAUDIT processing has been arranged such that several invocations of the user exit will want to write to the same QSAM output file, then non-trivial problems arise. For example:

the invocation that opened the output file must close it;

all invocations must share a single DCB to the file;

PUT processing must be serialized amongst the invocations.

DBAUDIT assists this process by providing both global and task related user words, which can be used to base "common" and task specific storage - but, be careful !!

Appendix

B

Sample Run Summary Output

Description

The output shown on the following pages can be generated by modifying the sample JCL and DBAUDIT parameters as described in the JCL and running the job. The summary report written to the ddname summary is the only output shown, the other output produced by the example job can run into 30,000 lines of output on a typical system file.

Appendix C

MU and PE field processing

PE/MU Processing

DBAUDIT allows MU and PE ranges to be specified as part of the field expression used by the SELECT-WHERE, PROCESS and RULE constructs. How DBAUDIT interprets these ranges to process expressions is the subject of this appendix.

DBAUDIT correlates the input specifications with the ADABAS file and field definitions. It insists that MU, PE and MU within PE fields be appropriately specified with explicit occurrence ranges. That is, 1 index expression must be specified for simple MU and PE fields, and exactly 2 index expressions must be specified for MU fields within a PE group.

Index expressions can be either:

the special character "*", which signifies that all occurrences appearing in the record are to be potentially processed (which will be range 1-191 for PE and MU fields).

a numeric constant, such as "5", which signifies that only 1 occurrences is to be considered.

a range, such as "1-20" or "30-191" or "23-23" which specifies low and high occurrences to be considered.

For the purpose of this appendix, the following extract of an FDT for ADABAS file number 55 will be assumed:

Level	ADABAS field name	Type	Length	Options
01	M1	A	4	MU
01	M2	A	4	MU
01	PG	PE		
02	P1	A	1	
02	P2	A	2	
02	P3	N	1	
02	P4	N	1	MU
02	P5	A	1	MU

The most significant rule DBAUDIT applies to MU/PE fields involved in comparison expressions is:

DBAUDIT will compare occurrences of the MU/PE field until a TRUE value is found, or until no more occurrences can be compared due to range restrictions.

Processing Example

Imagine DBAUDIT is processing a record with the following values for the above fields:

```
M1(1) = 'A' M1(2) = 'B'       M1(3) = 'B'
M2 - entirely null
P1(1) = 'A' P1(2) = 'B'       P1(3) = 'C'
P2(1) = nullP2(2) = 'BB'      P2(3) =  null
P3(1) = nullP3(2) = null      P3(3) = 100
P4(1/1) = 1 P4(1/2) = 2
P4(2/1) = null
P4(3/1) = 3
```

The expression: FIELD M1(/*) = 'B' will return TRUE, after following this logic:

> **the user specified '*' for the range, so all values present in the record will be tested until a TRUE value is found, or until no more occurrences exist in the record;**
>
> **M1(1) is tested, result is FALSE;**
>
> **M1(1) was not the last occurrence, so check next;**
>
> **M1(2) is tested, result is TRUE;**

Because a TRUE result has been found, testing will stop.

It is important to note that testing stops as soon as a TRUE value is found: i.e., in the above example, M1(3) is not tested.

The expression: FIELD M1(/*) <> 'B' WILL ALSO RETURN TRUE (!!!) because the expression evaluated on occurrence M1(1) will yield a TRUE result.

If the desired effect is to return a TRUE result only if NO occurrence of the MU has a value of "B", then the following construct MUST BE USED:

```
NOT (FIELD M1(/*) = 'B')
```

With the above data, M1(2) will cause the inner expression to return a TRUE result, which will be negated by the NOT to given a resultant value of FALSE.

The expression: FIELD M1(/*) = NULL will return FALSE, because only the first 3 occurrences of M1 will be tested. However the following expressions will return TRUE:

```
FIELD M1(/4) = NULL
FIELD M1(/4-191) = NULL
FIELD M2(/*) = NULL
FIELD M2(/1) = NULL
FIELD M2(/2-191) = NULL
```

The above outcomes are paralleled for simple fields within periodic groups. For example, the following expressions all yield TRUE values:

```
FIELD P1(/*) = 'C'
FIELD P1(/2-3) = 'C'
FIELD P1(/3-9) = 'C'
FIELD P1(/3) = 'C'
NOT (FIELD P1(/*) <> 'B')
NOT (FIELD P1(/*)  <> 'X')
FIELD P1(/1)  <> 'B'
FIELD P2(/1) = NULL
FIELD P2(/*) = NULL
FIELD P2(/*) = 'BB'
FIELD P3(/*) >= 100
```

Two indices must be explicitly provided when specifying MU fields within a PE. The following expressions will return TRUE values:

```
FIELD P4(/*,*) > 0
FIELD P4(/*,1) >= 3
FIELD P4(/*,1-2) > 1
FIELD P4(/*,*) >= 2
FIELD P4(/*,*) <> NULL
FIELD P4(/*,1) = 3
FIELD P4(/*,1) = NULL
FIELD P4(/1,*) = 2
FIELD P4(/1-3,1-2) = 3
FIELD P4(/1-3,1-2) <> 3
FIELD P4(/1-3,1-2) <> NULL
NOT (FIELD P4(/*,*) = NULL)
FIELD P4(/2,*) = NULL)
```

The following expressions will return FALSE values:

```
FIELD P4(/*,*) = NULL
FIELD P4(/*,*) > 3
FIELD P4(/2,*) <> NULL
FIELD P4(/1,*) > 2
```

Several expressions may be combined together to form the basis of a SELECT-WHERE clause, or a VALIDATION rule. Be sure to understand that each expression is evaluated INDEPENDENTLY. For example:

```
SELECT FILE 55 WHERE FIELD P1(/*) = 'A'
                   AND FIELD P3(/*) = 100
            ENDWHERE
```

will result in the above record being selected, because P1(1) = 'A' and P3(3) = 100 - i.e., it doesn't matter that the occurrences satisfying the criteria were different for each field.

Examples

Introduction

This section has several examples that use relevant input parameters and discuss these examples in some detail..

Assume file 12 is a NATURAL FUSER file. This rule will verify that every source program has a corresponding module and vice-versa. (Field LJ is used by NATURAL to store source code program names and field LL is used to store object module names. The 1st 16 bytes of each specifies the NATURAL library and program/module name.)

```
SELECT FILE 12 WHERE FIELD LJ <> NULL ENDWHERE
      RULE MATCH1: FIELD LJ(1-16) MANY TO MANY
      FILE 12 FIELD LL(1-16) WHERE FIELD LL <> NULL ;
```

If we were only interested in reporting on object modules without source, we could rewrite the request as either:

Example1A

```
SELECT FILE 12 WHERE FIELD LJ <> NULL ENDWHERE
      RULE MATCH1: FIELD LJ(1-16) MANY TO MANY (0-999)
            FILE 12 FIELD LL(1-16) WHERE LL <> NULL ;
```

The "(0-999)" clause means that 0 through 999 occurrences of each distinct LL(1-16) value are acceptable.

Example1B

```
SELECT FILE 12 WHERE FIELD LL <> NULL ENDWHERE
      RULE MATCH3: FIELD LL(1-16) MANY (0-999) TO MANY
            FILE 12 FIELD LJ(1-16) WHERE LL <> NULL ;
```

Here the major SELECT value has been changed to be on the LL field.

Example 2

Assume file 1 is the Natural demo EMPLOYEES file and file 2 is the Natural demo VEHICLES file.

Selected fields within file 1 are:

```
AA - personnel-id
AE - surname
AG - sex
```

Selected fields within file 2 are:

```
AA - registration number
AC - personnel-id
AD - make
AE - model
```

To verify that each vehicle is associated with a valid employee:

SELECT FILE 2

```
      RULE OWNER: FIELD AC MANY(0-999) TO ONE
            FILE 1 FIELD AA
            LIST LHS FIELD AD CONCAT AE CONCAT AA
            HEADING 'CARS WITHOUT VALID OWNERS' ;
```

To "join" together the employee and vehicle files on the common key, personnel-id, producing as output a list of each MALE employee which owns a car, and some details of each car:

```
SELECT FILE 1 WHERE FIELD AG = 'M' ENDWHERE
      JOIN CARS: FIELD AA WITH FILE 2 FIELD AC
            LIST LHS FIELD AD CONCAT AE CONCAT AA
            HEADING 'MALE EMPLOYEES AND THEIR CARS' ;
```

To produce a similar list restricting the selection to male employees owning a FORD:

```
SELECT FILE 1 WHERE FIELD AG = 'M' ENDWHERE
      JOIN CARS: FIELD AA WITH FILE 2 FIELD AC
                  WHERE FIELD AD = 'FORD' ENDWHERE
            LIST LHS FIELD AD CONCAT AE CONCAT AA
            HEADING 'MALE EMPLOYEES AND THEIR CARS' ;
```

Example 3

Assume the following:

File 23 contains purchase orders headers. Field PA contains the unique purchase order number. Field SA contains the supplier number. Field DA contains the date placed, field TA contains the total value of the purchase order, field BR contains the branch code of the branch raising the order and field S1 contains the status code of the purchase order.

File 24 contains purchase orders lines. Field LA contains the purchase order number plus line number - the 1st 8 characters of the field are the purchase order number.

File 30 contains supplier information. Field AA contains the supplier number, field NA their name, multi-valued field SS their address.

File 31 contains former supplier information, no longer referenced by any purchase orders. Fields are as for file 30. We could perform some processing on these files as follows:

```
SELECT FILE 23 WHERE S1 <> 'Z' ENDWHERE
RULE PO1: FIELD PA ONE TO MANY FILE 24 FIELD LA(1-8) ;
RULE SUP: FIELD SA MANY(0-99999) TO ONE
            FILE 30 FIELD AA ;
            PROCESS D1: VALIDATE FIELD DA
                  MASK(YYYY'/'MM'/'DD)
            ENDVALIDATE ;
            PROCESS V1: VALIDATE FIELD TA >= 1 AND
                  FIELD TA 50000
            ENDVALIDATE ;
```

Rule PO1 will check each 'active' purchase order (i.e., purchase orders with any status other than 'Z') as follows:

each purchase order has a least 1 line record

each line record is associated with a purchase order header (i.e., there are no "orphan" purchase order lines) Rule SUP will check each active order to make sure that the supplier actually exists.

Process D1 will check each purchase header record and verify that the DA field has a format YYYY/MM/DD (e.g., 1991/12/31).

Process V1 will check each active purchase header record and verify that the TA field has a value between 1 and 50000.

To find out which suppliers don't appear on any purchase orders, the following select and rule could be used:

```
SELECT FILE 30
      RULE S2 : FIELD AA MANY TO MANY (1-99999) FILE 23
      FIELD SA ;
```

To verify that all suppliers have a non-null name, the following select and rule could be used:

```
SELECT FILE 30
      PROCESS NAME: VALIDATE FIELD NA <> NULL   ENDVALIDATE ;
```

Another method of achieving the same result, which offers the added flexibility of producing other fields from the supplier record (such as the first 2 address lines) is:

```
SELECT FILE 30  WHERE FIELD NA = NULL ENDWHERE
PROCESS NAME: LIST FIELD SA CONCAT SS(/1) CONCAT
                  SS(/2) ;
```

To verify that each supplier code on the current supplier file is unique, the following process could be used:

```
SELECT FILE 30
      PROCESS UNIQSUPP: UNIQUE FIELD SA ;
```

To verify that no supplier code exists on both the current and former supplier file, the following join could be used:

```
SELECT FILE 30
      JOIN DUPSUPP: JOIN FIELD SA WITH FILE 31 FIELD SA
      LIST LHS FIELD NA  RHS FIELD NA ;
```

If we wanted to extract some data for purchase orders that have a value of DA < '1990/01/01' or a value of TA > 20000, and originated from branch 'ADP' or 'POLICY' we could provide the following request:

```
SELECT FILE 23 WHERE
            (FIELD DA < '1990/01/01' OR FIELD TA > 20000) AND
            (FIELD BR = 'ADP' OR FIELD BR = 'POLICY')
            ENDWHERE
PROCESS E1: LIST FIELD PA CONCAT SA CONCAT DA
            CONCAT TA CONCAT BR DDNAME PODATA ;
```

The LIST process will write the field PA, SA, DA, TA and BR and the ISN of the record to the dataset referenced by the DDNAME "PODATA" in the DBAUDIT JCL for each record satisfying the WHERE criteria.

If we wanted to add the supplier name to the above report, we could join purchase order headers to the supplier file as follows:

```
SELECT FILE 30
            JOIN J1: FIELD AA WITH FILE 23 FIELD SA WHERE
            (FIELD DA < '1990/01/01' OR FIELD TA > 20000) AND
            (FIELD BR = 'ADP' OR FIELD BR = 'POLICY')
            ENDWHERE
      LIST LHS FIELD NA
            RHS FIELD PA CONCAT SA CONCAT DA
            CONCAT TA CONCAT BR DDNAME PODATA
HEADING 'Selected Purchase orders by Supplier' ;
```

The LIST process will write the field PA, SA, DA, TA and BR and the ISN of the record to the dataset referenced by the DDNAME "PODATA" in the DBAUDIT JCL for each record satisfying the WHERE criteria.

If we wanted to count the number of purchase orders allocated to each supplier:

```
SELECT FILE 30
PROCESS CS1: COUNT FIELD SA ;
```

If we wanted to count the number of purchase orders allocated to each supplier by year by branch:

```
SELECT FILE 30
PROCESS CS2: COUNT FIELD SA AGAINST FIELD DA(1-4) CONCAT BR ;
```

If we wanted to find the total value of purchase orders, by supplier:

```
SELECT FILE 30
      PROCESS CS1: TOTAL FIELD TA AGAINST FIELD SA ;
```

If we wanted to find the total value of purchase orders, by supplier by year by branch:

```
SELECT FILE 30
      PROCESS CS2: TOTAL FIELD TA AGAINST
      FIELD SA CONCAT DA(1-4) CONCAT BR ;
```

JOINFILE Utility

Introduction

The JOINFILE utility enables 2 sequential, sorted datasets to be "joined" on a common key. Both datasets must be sorted on the common key. The key field can appear at any location within each file, but must, of course, be the same length in each file. Both files must have their record format as variably blocked (i.e. RECFM=VB in the DCB). The 2 input datasets have DDNAMEs of IN1 and IN2. A "one (or none)" to "many (or none)" relationship is assumed between the 2 input files, IN1 and IN2. That is, duplicate keys present in IN1 are considered as errors, and are ignored for joining purposes. The errors will be reported to the error file (ddname ERR) if it is present in the JCL.

Output

The JOINFILE utility has been designed to work with the output from DBAUDIT, but it will operate on any files meeting the above requirements. It is included with DBAUDIT to allow DBAUDIT to produce an output file built up from 2 DBAUDIT output files, hence supporting joins on 3 or more files/record types without requiring user programming.

Input

The input to JOINFILE may be files output from the DBAUDIT JOIN statement, in which case automatic key location can be specified (see below). Alternatively, input to JOINFILE may be output from a DBAUDIT LIST (etc.) statement, which has been sorted on the desired key subsequent to being produced by DBAUDIT.

JCL and PARAMETERS

Typical JCL to run JOINFILE is:

```
JCL-RIGHT = //      JOB card
//JOINSTEP   EXEC PGM=JOINFILE,PARM=' parameters - see below'/
STEPLIB DD   DSN=XXX.DBAUDIT.Vxxx.LOAD,DISP=SHR
/IN1     DD   DSN=xxx.input. file1,DISP=SHR
//IN2     DD   DSN=xxx.input.file2,DISP=SHR
//OUT     DD   DSN=xxx.output. file,DISP=SHR
//ERR     DD   DSN=xxx.error.file,DISP=SHR /* optional */<EO>
//*
```

IN1 and IN2 are the 2 input files. As stated above, basically IN1 and IN2 are treated as being in a "one to many" relationship, although missing records are tolerated and may be considered "normal".

The OUT file contains the result of the JOIN. Depending on the use of the INCLUDENULL/DROPNULL parameters described below, it will or will not contain join records created with 1 side of the join being null (i.e., a record is missing on either the left hand or right hand side).

The ERR file is optional. If it is included, then it will contain:

> **any records from the LHS file (IN1) with duplicate keys;**
>
> **any records from LHS or RHS without matching records if and only if the DROPNULL parameter has been specified.**

Parameters

The execution parameter syntax is:

```
INCLUDENULL | IN1= JOIN | (y-length),IN2= JOIN |
DROPNULL col-spec1 col-spec2
```

Either INCLUDENULL or DROPNULL must be specified.

Either JOIN or col-spec1 must be specified as the starting column of the matching key from file IN1. If JOIN is specified, JOINFILE assumes that the IN1 file has been output from DBAUDIT's JOIN/JOINNULL function, and the key is located automatically be scanning for "LHSVAL:" and "RHSVAL:" literals.

Either JOIN or col-spec2 must be specified as the starting column of the matching key from file IN2.

Examples of parameters are:

```
INCLUDENULL,IN1=10(4),IN2=12
```

The above parameters will match files IN1 and IN2, including in the output file OUT any records without matches in both files with the missing record being replaced with null values. The key starts at column 20 in file IN1 and column 12 in file IN2, and is 4 bytes long.

Note: the length of the variable record "record descriptor word" (RDW) is not included in the column counting, and the 1st column in the actual record is numbered as column 1.)

If IN1 contained:

```
col:...1....+....2....+....3
      Record 1 AAAA from IN1
      Record 2 BBBB from IN1
      Record 3 CCCC from IN1
```

and IN2 contained:

```
col: ....+....1....+....2....+....3
      From IN2:  AAAA xyz
      From IN2:  AAAA abc
      From IN2:  CCCC
      From IN2:  DDDD
```

then the OUT file would contain:

```
col: ....+....1....+....2....+....3...+....4...+
      Record 1 AAAA from IN1From IN2:  AAAA xyz
      Record 1 AAAA from IN1From IN2:  AAAA abc
      Record 2 BBBB from IN1.................(...=nulls)
      Record 3 CCCC from IN1From IN2:  CCCC
            From IN2: DDDD       (.=nulls)
```

If the ERR file was present in the JCL, it would contain:

```
col: ....+....1....+....2....+....3...+....4...+
      LHS DUP KEY:Rec 3 CCCC from IN1

DROPNULL,IN1=10(4),IN2=12
```

as above, but non-matching records are not to be output to the OUT file, but are to be written to the ERR file (if present in the JCL), prefixed with an appropriate message:

Assuming the same input as before, the output to OUT would be:

```
= col: ....+....1....+....2....+....3...+....4...+
      Record 1 AAAA from IN1From IN2:  AAAA xyz
      Record 1 AAAA from IN1From IN2:  AAAA abc
      Record 3 CCCC from IN1From IN2:  CCCC
```

and the output to ERR would be:

```
col: ....+....1....+....2....+....3...+....4...+
      RHS MISSING:Record 2 BBBB from IN2
      LHS DUP KEY:Rec 3    CCCC from IN1
      LHS MISSING:From IN1:  DDDD

      INCLUDENULL,IN1=JOIN(8),IN2=16
```

match files IN1 and IN2, including in the output file OUT any records without matches in both files with the missing record being replaced with null values. The key starts at column 16 in file IN2 and is 8 bytes long. The IN1 file is presumed to have been output from the JOIN/JOINNULL function of DBAUDIT, and JOINFILE will automatically locate the key to be used for matching by scanning the records of IN1 for the 'LHSVAL:" and "RHSVAL:" literals output by DBAUDIT. If the value following the "LHSVAL:" literal is null, the value following the "RHSVAL:" literal is used as the key to match.

```
INCLUDENULL,IN1=JOIN(8),IN2=JOIN
```

as above, except that both IN1 and IN2 were output from DBAUDIT's JOIN/JOINNULL function, and keys are to be automatically located by JOINFILE.

A

D

E

I

M

O

P

Q

R

S

U

V