DISTRIBUTION LIST

B1800/B1700 SOFTWARE PRODUCT SPECIFICATIONS

DETROIT

T. Freeman - Prod. Mgmt.
F. D'Angelo - PROD. MGMT.
F. Schoeman - International
J. Dormady - International
W. Varns - BMG
D. Kosinski - BMG
B. Dent - CSG
D. Dahm - Corp. Eng.
W. B. Hunt - SSG
C. J. Tooth - SSG
D. Hill - TC, BM, & SS
P. E. Pleming - Int'l FE

U.S. AND EUROPE

S. Oksalo - (Plymouth)
R. H. Mack - TSG (Plymouth)
J. H. Pedersen (Plymouth)
J. Berta (Downingtown)
W. Minarcik (Paoli)
G. Smolnik (Paoli)
R. Boruta - F&SSG (Paoli)
J. Murtaugh (Tredyffrin)
A. Kosla (McLean)
A. LaCivita - F&SSG (McLean)
R. Ledford - F&SSG (McLean)
R. Sutton - F&SSG (McLean)
Mgr, WADC (Irvine)
W. H. Keil (Pasadena)
H. M. Townsend (Pasadena)
N. Cass - Pat. Atty. (Pasadena)
D. C. Swanson (Mission Viejo)
H. O. Wickman (Mission Viejo)
J. J. Dowling (Westlake)
H. N. Riley (El Monte)
W. E. Feeser (Austin)
J. C. Allan (Glenrothes)
W. McKee (Cumbernauld)
I. J. Carradine (Cumbernauld)
Mgr, NPSGrp (Ruislip)
P. R. Evans (Middlesex)
B. Hammersley (Croydon)
J. Gerain (Pantin)
J. Cazanove (Villers)
J. C. Wery (Liege)
R. Bouvier (Liege)
S. Samman (Liege)

SANTA BARBARA PLANT

R. S. Bunker
J. Hale
R. Shobe
K. Meyers
A. van der Linden
T. Cardona
R. Bauerle
J. Henige
E. Yardi
J. Darga
B. Ross-Smith
E. Munsch - 2
G. Hammond - 2
J. Morrison - 6

Distribution list current as of 01/24/79

DOCUMENT TRANSMITTAL

PAGE ____ OF ____

CONTROL DOC. None

SYSTEM B1800/B1700

PL FILE NO. ____

PROJ. NO. ____ COORD. ____ CHKR. Kale TO RECORDS [illegible] DISTRIBUTION [illegible]

DOC PFX	DOCUMENT NUMBER	REV	UNIT NUMBER	DL NO.	PL FILE NO.	ENG. RECORDS ONLY						REMARKS
						QTY REV PAGE	COPIES EACH PAGE	SEPIAS	MT REQ'D	CD OK REQ'D		
PS	2219 0532	C				37	65					
DOCUMENT TRANSMITTAL												

Burroughs Corporation

COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

P.S. 2219 0532

B1800/B1700 DMSII AUDIT & RECOVERY

PRODUCT SPECIFICATION

REV LTR	REVISION ISSUE DATE	APPROVED BY	REVISIONS	
A	4/6/78	J Hale	MARK VII.0 RELEASE - Original Release	
B	7/11/78	J Hale	MARK 8.0 RELEASE	
			Page	Change
			A-2 & A-3	BIT LENGTH for PREVIOUS AUDIT SERIAL NUMBER added to DATA SET records. INDEX RANDOM and INDEX SEQUENTIAL entries replaced with INDEXES. Added TABLE CHANGE RECORDS.
			A-4 - A-6	Added BLOCK CONTROL FIELDS. PREVIOUS AUDIT SERIAL NUMBER moved to first entry for LIST records. Deleted NEXT AVAILABLE (BIT(32)) from @4C@. Added LIST HEAD heading. PREVIOUS AUDIT SERIAL NUMBER moved to first entry for LIST HEAD and RECORD ALLOCATION records. Added @62@ to RECORD ALLOCATION.
C	2/1/79	J Hale	A-1	Deleted sentence concerning high-order bit of the second digit.
			A-2 - A-4	BLOCK LOGICAL ADDRESS BIT(24) added to Data Sets and Lists DATA RECORD LOGICAL ADDRESS BIT(32) changed to DATA RECORD NUMBER BIT(8) TABLE ENTRY LOGICAL ADDRESS BIT(36) changed to TABLE ENTRY NUMBER BIT (12) LIST TABLE LOGICAL ADDRESS BIT (32) changed to LIST TABLE NUMBER BIT(8) LIST Records renumbered
			A-4 - A-6	BLOCK LOGICAL ADDRESS BIT(24) added to List Head and Record Allocation Record format changed for List Head entries and Index Splits and Combines Deleted @74@ and @75@ from Index Splits and Combines LIST HEAD records renumbered.
			D-1 - D-2	Appendix D - Program Switches, added.

TABLE OF CONTENTS

INTRODUCTION

This document describes the audit and recovery procedures that are available in the Mark 8.0 and later release of DMSII. It is a general description of these features and a discussion of how they are implemented. The syntax for specifying Audit and Recovery is described in the DASDL product specification (P.S. 2219 0433) and an example of AUDIT/RECOVERY specifications is given in Appendix B of this document.

DMSII audit and recovery procedures enable users to safeguard and to reconstruct the integrity of their data bases by automatically providing a history of all changes to the data base and by allowing a range of recovery procedures.

RELATED DOCUMENTATION

NAME	NUMBER
DASDL	P.S. 2219 0433
DMS Reorganization	P.S. 2219 0540
MCPII	P.S. 2212 5462
Software Operational Guide	1068731

DEFINITIONS OF TERMS

This section contains definitions of the basic terms upon which the following discussion is based.

WORDS — MEANING

AUDIT — Audit is a process of recording within a series of files (the audit trail) a complete history of all the changes made to a data base. Each data base has its own audit trail that cannot be shared with any other data base.

TRANSACTION — A sequence of data management operations required to process one user request. Operations which change the data base can only be done during a transaction.

PROGRAM ABORT RECOVERY — The process of restoring the integrity of a data base when a program is terminated in the middle of a transaction, but the data management system continues to run.

CLEAR/START RECOVERY — The process of restoring the integrity of a data base which was open and being updated when a system failure occurred.

SYNCPOINT — A DMS-controlled time when no programs are updating a data base. No programs are in a transaction.

CONTROLPOINT — A SYNCPOINT that also writes to disk any data base blocks or data base control information that has been updated and that has been in memory since prior to the previous CONTROLPOINT.

RESTART — A process by which the Data Management system provides information to an application program about the last transaction that was completed before a failure which required recovery. The programs that are restarted by the operator after a CLEAR/START can use this information to commence processing transactions that were not completed.

DUMP RECOVERY — The process of using the "after" images in the audit trail and a backup copy of the data base to rebuild the data base when it is found to be unusable because of storage failures.

PARTIAL DUMP RECOVERY — The process of recovering a subset of the files in a data base from a previous dump (Dump Recovery of a subset of the data base).

RECOVERY — The process of restoring the integrity of a data base that may have been corrupted or lost through a hardware or software failure.

FILE DUMP A backup copy of part or all of the data base created while the data base is not being used. The information necessary for DUMP recovery to enter the audit trail will be automatically saved in the <data-base-name>/DICTIONARY which should be included with each dump.

AUDIT

The audit trail is a sequence of "before" and "after" images resulting from changes written to the data base, plus various control records. Pure retrieval requests are not audited. The audit trail is used to recover the data base after a Clear/Start, to provide restart information to object programs, to reconstruct the data base when portions of the data base are lost due to hardware errors, and to back out transactions for aborted programs.

TRANSACTIONS

Restart after recovery is accomplished efficiently by requiring some aid from the application programs. To minimize the coding required in the application programs, transaction processing is the basis for recovery and restart.

A transaction is any sequence of data management operations starting with BEGIN-TRANSACTION and ending with END-TRANSACTION (or END-TRANSACTION with SYNC). A sample program using transactions is included in Appendix C.

When BEGIN-TRANSACTION is executed, the program enters transaction state. When END-TRANSACTION (or END-TRANSACTION with SYNC) is executed, the program leaves transaction state. Also, all records locked by the program are unlocked at the completion of END-TRANSACTION. (The only exception is that the current restart-data-set record is not unlocked.) These transactions may not be nested. That is, BEGIN-TRANSACTION returns an AUDITERROR exception if the program was already in transaction state. Any error on BEGIN-TRANSACTION other than AUDITERROR will leave the program not in transaction state. Any END-TRANSACTION (or END-TRANSACTION with SYNC) returns an AUDITERROR exception if the program was not in transaction state. Any exception other than AUDITERROR will leave the program in transaction state.

END-TRANSACTION with SYNC forces a SYNCPOINT to occur after the transaction is completed but prior to returning control to the user program. This form should be used very sparingly to avoid a significant decrease in system throughput. On data bases which have the audit option set in their DASDL, the only operations which are legal when not in transaction state are OPEN, FIND, MODIFY, FREE, CLOSE, CREATE, RECREATE and BEGIN-TRANSACTION. Any other operation will return an AUDITERROR exception. If AUDIT is specified in BEGIN-TRANSACTION or END-TRANSACTION, the restart record (within the user program) is saved for possible restart. (A STORE on the restart data set is performed). If NO-AUDIT is specified, the restart record area is not saved. AUDIT is true by default for BEGIN-TRANSACTION; NO-AUDIT is the default for END-TRANSACTION.

Prior to any BEGIN or END-TRANSACTION with AUDIT, a CREATE or RECREATE of the restart-data-set must be executed if there is no locked current record for the restart-data-set. If this is not done, a NOTLOCKED exception will occur.

AUDIT should never be set on both BEGIN-TRANSACTION and END-TRANSACTION. Only the last audited restart record is available at restart time, thus the BEGIN-TRANSACTICN will never be returned to the user. If restart information is updated during a transaction, then AUDIT should be set on END-TRANSACTION. Otherwise AUDIT on either but not both. Any transaction which does not set AUDIT on either is not restartable.

RESTART-DATA-SETS

At restart time the data base will be recovered by the RECOVER/DATA.BASE program. It is the user's responsibility to restart the user program and to restore the necessary state for that program. The restart record is provided as a storage area for some of the state information, but it does not do such things as re-position non-data base input and output files, etc.

For each data base that has the audit option specified, a restart-data-set is required. This data set follows the rules of normal disjoint data sets, with a few additions. The restart-data-set may not contain any embedded data sets or subsets. The restart-data-set must be invoked by any COBOL program which contains a BEGIN- or END-TRANSACTION. The restart-data-set record is used for program-dependent restart. The purpose of the restart record area is to hold the minimal amount of information that a program needs to restart after a Clear/Start or when transactions are aborted. Please note that the same size restart-data-set record is used for all programs that run against a data base. The size of the restart data set must be large enough to restart the program which requires the most information to restart.

It is intended that the user supply an item (or items) uniquely identifying the restart areas for each different application program. Typically, the program-id is used as the key to access the restart data set. There often will be a large alpha item which holds the actual restart information. Moving this item to a COBOL work area will allow each different program to access this information in the most convenient format. All COBOL verbs may be used to access the current restart record.

SYNCPOINTS

Transactions for different users may overlap. To allow maximum throughput (minimum record contention), changed records need not be locked until the end of the transaction updating them. Thus, in general, it is not possible to back out a partial transaction without backing out all overlapping transactions (completed or not). In order to limit the amount of backing out required at recovery time, the SYNCPOINT was introduced. SYNCPOINT is a point when no programs are in transaction state.

Transactions are intended to be of roughly equal duration. Otherwise, a disproportionately long delay may be imposed upon programs executing short transactions at syncpoint time. This delay results from having to wait for the longer transactions to complete.

Obvious sources of long waits (such as "NO FILE" conditions) should be avoided when in transaction state. A single program hung while in transaction state means that, eventually, no transactions against the data base will be able to proceed, because a SYNCPOINT must occur.

The user may specify the frequency of syncpoints in terms of transactions. At each END-TRANSACTION, the system checks to see if it is time for a syncpoint. If it is time, then the system waits for all in-process transactions to complete, places a syncpoint in the current audit record, and initiates the I/O on the audit record. Any jobs attempting to execute a BEGIN-TRANSACTION during this period are suspended. When the I/O completes, the syncpoint is complete, and all suspended jobs are reinstated.

The user may force a syncpoint at any time by initiating an

END-TRANSACTION with SYNC. SYNCPOINT must complete before control is returned to the user program when this form of END-TRANSACTION is used.

Close will always force a SYNCPOINT prior to returning to the user program. This eliminates the possibility of having to backout a close due to a program abort or Clear/Start.

CONTROL POINTS

In order to minimize physical I/O, changes to data-base buffers are accumulated in memory. Updated buffers are written to disk only when the memory space they occupy is required for another purpose. Thus, without a special mechanism to force writes on all buffers, Clear/Start recovery would have to go back to the data-base open to insure that all changes are reflected in the data base.

CONTROLPOINTS are the special mechanism to limit the amount of time needed for Clear/Start recovery. The user may specify the frequency of CONTROLPOINTS in terms of SYNCPOINTS. When a syncpoint completes, the system checks to see if this is also a controlpoint. If it is then special code is executed to insure that modified buffers and control information are written to disk at least every two control points. When all I/Os have completed, then a control point is marked in the audit buffers and then normal processing resumes.

When a buffer is updated, it is marked to show that. When the next CONTROLPOINT occurs, if the buffer is still in memory, it is marked to show that. When the second CONTROLPOINT occurs, if the buffer is still in memory, then it will be forced out to disk before the CONTROLPOINT will finish. Thus, Clear/Start recovery need not go back further than two CONTROLPOINTS from the end of the audit trail. If CONTROLPOINTS do not occur frequently, then it is likely that only frequently used records (such as coarse tables) will need to be flushed at CONTROLPOINT time. (Infrequently used records will tend to be overlaid during normal processing against the data base.) Updated available space information (NAHO's) are flushed every other CONTROLPOINT also.

AUDIT-SERIAL-NUMBER

The audit-serial-number performs a number of functions in both auditing and recovery. Each time any block of data is changed because of a data base operation, the change is assigned a unique audit-serial-number. The initial serial number is a function of DASDL compilation. This is used to perform a version check on the first audit trail file. The audit-serial-number is incremented by one for each change to the data base and uniquely identifies an audit record.

One of the primary rules on an audited data base is that no updated data base blocks may be written to disk unless its changes have been written to the audit trail. This guarantees that any change can be backed out if necessary. The audit-serial-number is used to insure that this rule is met. The system keeps track of the highest audit serial numbers in the audit trail. Each block in memory has associated with it an audit serial number reflecting the last update that was done to that block. Any data base block with an audit serial number which is higher than that on the audit trail may not be written to disk.

All tables in the data base (index sequential, index random and lists) contain a space at the end of their block which contains an audit serial number. This audit serial number corresponds to the last change that was made to this table. The audit record which describes the change to the block also has associated with it a serial number. By comparing these serial numbers, the recovery program can tell if the change described by a given audit record has been done, and take the appropriate action. This prevents performing the same operation twice or not performing it at all. Blocks which contain data set records do not contain serial numbers because changes to a data set record do not affect other records; and therefore, a change may be applied more than once with no effect on the data base.

Each audit record concerning tables also explicitly carries with it another audit serial number. This serial number is the audit serial of the block before the change described by this audit record was made. This serial is needed for Clear/Start recovery when it is not known what blocks have been updated on disk. Program-abort recovery also uses this serial number when backing out operations, although it does know that all blocks have been updated on disk.

If the recovery program finds a match between the serial numbers on a block and the audit record, it is sure the change described by the audit has been done. When it backs this change out, it replaces the serial number on the block with the previous serial number. The next time the block is updated the recovery program can again check for the equal condition. After the recovery program has backed out the changes back to the last SYNCPOINT before the Clear/Start, all serial numbers that it will see will then be in a known range. This allows for effective consistency checking which is important in the environment after a Clear/Start. This same type of consistency checking is done for all types of recovery.

AUDIT FILES

The audit trail consists of a sequence of audit files which are named:

<data-base-name>/AUDIT<audit-file-number>

The audit file number starts at one and increases by one every time a new audit file is opened. When it reaches 99,999 it "wraps around" to zero and starts again.

The audit trail should not reside on the system disk but on a user pack or tape, since it will be lost if the system disk fails and a Cold/Start is required. It also should not reside on the same pack as any of the data base files. If that pack is damaged, then both the data base files and the audit files used to protect them will be lost.

Audit files with no data in them may be generated. This may occur when an irrecoverable I/O error is encountered during the write of the first block to an audit file or if a Clear/Start happens prior to the first block being written. These files are not removed since the recovery program automatically skips them, whether it is going forward or backward.

AUDIT FILE FORMAT

The audit file contains fixed-length blocks with variable-length records. The length of a block is set during DASDL compilation and each audit block contains a fixed-size trailer. The trailer contains two audit serial numbers. An audit serial number is needed for every record in the audit file; however, once it is established, it can be calculated from the previous record's serial. After the serial has been established, the serials on the blocks become redundant information which is checked for consistency.

The trailer contains a current block audit serial number, a block number and a full-block bit. The current block audit serial number is that of the first record in the block. The block number is relative to the beginning of the file (0 is first). If the block is completely full, the full-block bit is set to indicate that status.

The trailer also contains the next block's audit serial number and a pointer to the last record. The serial is of the first record in the next block. It matches the current serial in the next block. The pointer is the distance in bits from the start of the block to the end of the last record in this block. If no records end in this block, the pointer takes on the value of @FFFF@. This situation comes up when audit records are long relative to the size of the block.

Several of these fields are used to insure that the end of the audit file is not lost. The end of the audit file can be found by reading the audit trail forward and checking that the block number is correct, and also that the current block audit serial number of this block matches the previous block's next block's audit serial number.

If either of these checks fail, then the end of file has been found. This applies whether auditing to disk or tape. The end-of-file pointer on a disk file is updated on disk every time a new area is opened (it points at the last block in the new area) and when the audit file is closed (it points at the last block used). Thus, in finding the end-of-file on a disk file, the serial search can start at the end of the next to last area.

The length of audit records is determined by the type of the record and the structure which the record references. The audit file must be read forward and backward and so the record type and structure number appear at each end of the audit record.

Some record types are not associated with any particular structure but describe the state of the data base. These only have a record type. They include data base open, close, SYNCPOINT, CONTROLPOINT and PROGRAM ABORT. The rest of the record types are associated with a particular structure and that structure number follows the record type, independent of the direction that the auditfile is being read. (For further information on audit record types and formats, see Appendix A).

Some information in the audit file is redundant at certain times during recovery. An example of this is the record type and structure number at the "other" end of the record. This information is checked as protection against undetected I/O errors.

```
***********************************   1. Audit record area.
*                                 *   2. Last record displacement
*                                 *      from start (16 bits)
*               1                 *   3. First Audit Serial in
*                                 *      this block (32 bits)
*                                 *   4. First audit serial in
*                         *********      next block (32 bits)
*                         *   2   *   5. Full block indicator
***********************************      (1 bit)
*   3   *   4   *   5   *   6     *   6. Block number, first block
***********************************      in file = 0 (32 bits)
         AN AUDIT BLOCK
```

```
              ***********             1. Record type (8 bits).
              *    1    *
              ***********
   AN AUDIT RECORD (DATA BASE STATE)
```

```
*****************************************************
*  1  *  2  *             3            *  2  *  1  *
*****************************************************
```

AN AUDIT RECORD (STRUCTURE OPERATION)

1. Record type (8 bits).
2. Structure number (8 bits).
3. Fields depend on record type and structure number.

AUDIT FILE USAGE

Audit files are of variable length; their size is determined by a number of factors. Initially, an audit file is opened at the first BEGIN-TRANSACTION after the data base is opened. The audit file is closed whenever any one of the following occurs:

1. The audit file fills up.

2. An irrecoverable I/O error occurs.

3. The data base is closed by all programs which have ever gone through BEGIN-TRANSACTION.

4. A PROGRAM ABORT occurs.

Although nothing appears on the audit file to indicate it, the audit file is closed in a sense by a Clear/Start recovery. The exact way that this is handled is explained later in this section.

In cases one and two, there will be no indication why the audit file has been closed. The program will simply reach end of file. In fact, an audit record may be split across a file boundary. In cases three and four, a record is put into the audit file to indicate what caused the file to close.

When an audit file is closed by the MCP, some processing is done, so that the file may be used later. If the file is a disk file, the EOF pointer is adjusted so that it reflects the last block that was written. If the file is a tape file, tapemarks and an ending label which contain, among other things, the block count, are written on to the end of the file. But when a Clear/Start occurs, none of this can be done. To recover the data base immediately after the Clear/Start and make future dump recovery possible, some processing is done after the Clear/Start.

Any tape files that were opened output get a tape mark written on them before they are rewound as part of the Clear/Start. The recovery program requires this tape mark be at the end of all tape files. If the tape mark is not written for some reason (for instance, the tape is dismounted before the Clear/Start) then DMPALL or some other utility must be run to make a new tape with a tape mark.

For disk files, the recovery program must fill in the end-of-file pointer. As the audit file is being written, the end-of-file pointer is always kept at the end of the area being filled up. Because of the way disk space is allocated and buffers are output from memory to disk, the good audit blocks are known. If there is more than one area, recovery is assured for the last block of the next to the last area. That is, when the disk space for a new area is obtained, the last block in the prior area has been successfully written out. If an auditfile has only one area, then there may not be any good audit blocks in it. When the recovery program finds the last good block, it updates the end-of-file pointer to the correct value.

For future recoveries, some way is needed to identify that a Clear/Start recovery has taken place. The way this is done is to make the audit serial number one greater than it would normally be. This is used for both Dump recovery and Clear/Start recoveries which happen immediately after the present one. For Dump recovery, the recovery program opens the next file as it comes to the end of the present one. That a Clear/Start has happened is apparent in the serial-number check. The recovery program then must reopen the prior audit file and back out to the last SYNCPOINT.

If a Clear/Start happens just after the audit file opens, but before the first block gets written out, the recovery program opens the prior file. If this file was closed due to a data base close or it had Clear/Start recovery run on it, there is no recovery that has to be performed. Since all changes to the data base do not go out of memory until the audit block describing them is written out and no audit blocks have gone out, nothing needs to be backed out. However, if, for instance, the prior file was closed because it was full, then the changes described in this prior audit file must be backed out. To tell the difference, the serial number is again used. If the next block serial in the last audit block is the same as that in the dictionary, then a good close has taken place. If the next block serial is one less than that in the dictionary, then a Clear/Start recovery has taken place. Other values for that serial indicate a Clear/Start recovery is needed.

AUDIT EFFICIENCY

Two audit buffers are always used when creating an audit file. The buffers are used alternately. The switch occurs when ever a buffer fills or a syncpoint occurs while auditing to tape. When an audit file fills up, any operations in process are allowed to complete (but no new ones may start), filling the audit buffers and, if necessary, overflowing into temporarily allocated buffers. The audit file is closed and a new audit file is opened, preserving the FIB and audit buffers through close and open. If necessary, the current buffers that are full, and any overflow buffers, are written out to the new audit file. At the completion of these I/Os, the data base operations are allowed to commence.

Audit efficiency is determined by two critical parameters: the frequency of SYNCPOINTS and the file's blocksize. To keep the speed from deteriorating excessively when auditing, the audit trail must have a large effective blocking factor. Too small an audit file blocksize will prevent this, as will frequent SYNCPOINTS. (The audit buffers are flushed at SYNCPOINT time, and the DMS routines are inhibited until the writes complete). On the other hand, if SYNCPOINTS occur infrequently, many transactions may be backed out during Clear/Start recovery or when aborting a transaction. Also, a large audit file blocksize implies that more memory is required for audit buffers and for data buffer space, since updated buffers may not be overlaid (written to disk) until the corresponding audit entries have been written to the audit trail. When auditing to tape, if the blocksize is larger than the audit information recorded during a SYNCPOINT, the extra space does not get used.

For data sets, record images are audited; and for index-sequential, index random and lists, table entry changes are audited. Logical audit records may be split across one or more physical audit blocks. These techniques cause a considerable reduction in the amount of data which might otherwise be written to the audit trail.

I/O ERRORS WHILE AUDITING

During an audit, if an I/O error occurs while attempting to write to the audit trail and the normal MCP retries are not successful, the current audit file is closed and the next one opened. The write is then retried. The data base is locked against update until the audit block is successfully written. If tape is the audit media, it should be copied when the last block of an audit tape had a write parity. The recovery routines need a copy of the tape with a tape mark at the end and no faulty areas.

RECOVERY

When an event has occurred that compromises the integrity of an audited data base, a program is run to make the data base trustworthy again. This section describes that program (RECOVER/DATA.BASE) and some of the resources it uses to accomplish its job.

A data base may lose its integrity in a number of ways. Three possibilities are:

1. The loss of the disk that any part of the data base was stored on.

2. A failure of the hardware or software while the data base is being updated such that a Clear/Start is required. An unknown amount of changes to the data base may be trapped in memory in this case.

3. The failure of a program that was in transaction state (and was potentially capable of changing the data base) to successfully leave transaction state.

These types of failures are handled by the recovery program and are referred to, respectively, as:

1. Dump or Partial-dump recovery

2. Clear/Start recovery

3. Program-abort recovery

Note: The types of failures are ranked in order of their seriousness, i.e., the effort and processor time to recover from the error. The loss of disk (Possibility 1) is the most serious and may include Program-Abort recovery as part of a Dump recovery.

A software failure may cause either a program or system failure. If they occur at the same time, a Clear/Start recovery (the more serious failure) is required.

RECOVERY INITIATION

Recovery begins when the RC control message is entered upon the console printer or CRT. The RC control message executes RECOVER/DATA.BASE, providing the necessary label equation. RECOVER/DATA.BASE must reside on system disk at the time RC is used. The syntax of RC is defined as:

SYNTAX

```
RC --- <data-base-name> -------------------------------->>
                           |                        |
                           |--- ON -- <pack-name> -->|


    >>---------------------------------------------------------->#
         |                      |
         |<-------- , ----------|
         |--- <file-name> ----->|
```

The RC message zips the execution of the data base recovery program and the user-supplied <data-base-name> identifies the name of the data base that is to be recovered. The type of recovery to be done is automatically defined by the information in the data base dictionary on the status of the data base.

RC allows an optional <pack-id> for those data bases where the dictionary resides on a user disk as well as an optional file-list to indicate dump recovery of the specified data base files only. The <file-identifier>s are put into a parameter file similar to a LOAD.DUMP file. The address of this file is passed to the recovery program anc Switch 4 is set to "1" (SW4=1) by the MCP to indicate recovery of a partial data base. There is no file list required for Clear/Start, abort recovery, or full-dump recovery.

TYPES OF RECOVERY

The simplest failure from a recovery point of view is Program-abort recovery. The recovery program need only back out all the changes made to the data base from the last SYNCPOINT or, if there has not been a SYNCPOINT since the data base was opened, then to the data base open.

Clear/Start recovery involves backing out all the changes since the last SYNCPOINT and then going back two CONTROLPOINTS and making sure that all changes to the data base that were done in this interval are reflected in the data base.

Dump recovery requires that the previous copy of the data base, and the audit files written since that copy was taken, be loaded to disk. The recovery program applies the audited changes, i.e., the "after" images, to the previous database. The recovery program processes these images in the same way the DMS program originally processed them, with the exception that when a Clear/Start is encountered on the auditfile, the recovery program only has to back out changes since the last SYNCPOINT. Any changes done before the last SYNCPOINT have been written to disk by the recovery program.

Partial-dump recovery allows the user to perform Dump recovery on a subset of the files in the data base. A backup copy of only those files to be recovered and the old Dictionary are loaded. The old Dictionary is loaded under the name <data-base-name>/OLD.DICT. The current Dictionary must remain under its proper name. Dump recovery is performed on the specified files until they are up to the level of the current Dictionary. At that time Clear/Start recovery or Program-abort recovery will be performed if necessary.

CLEAR/START RECOVERY

When the first update on the data base occurs, a bit (boolean) is set on in the <data-base-name>/DICTIONARY. This bit is turned off when the data base is closed for the last update program. On the first open of the data base, if this bit is on, the data base was not closed properly (e.g., there was a Clear/Start without a successful data base close). Clear/Start recovery is requested. All data management jobs will be suspended by data management OPEN until recovery is successful.

Recovery occurs in four phases:

1. The end of the audit trail is found.

2. The audit trail is read backward to the last SYNCPOINT. The "before" images are used to back out all changes since the last SYNCPOINT.

3. The second CONTROLPOINT from the end of the audit trail is found, and the audit trail is scanned forward from this point. The "after" images are used to insure that all transactions before that last SYNCPOINT are completed. The restart areas for the last completed transactions are saved in the restart data set.

4. The audit file is terminated, the data base is unlocked, and recovery is complete.

Although Clear/Start recovery is requested automatically by the first data base OPEN executed after a Clear/Start, the operator may safely run it at any time. If recovery is not needed, the recovery program will ask for a non-existent audit file. The operator should respond that it is not available and recovery will terminate.

PROGRAM ABORT

The locking of records in DMSII is designed to maximize the potential for multiprogramming against the data base and hence, total throughput. Thus, changed records need not be held locked until the end of the transaction which changed them. A partial transaction cannot, in general, be backed out without invalidating other transactions that were occurring at the same time. One transaction may have updated the data base with data created by the aborted transaction. Thus, an attempt to abort a single transaction may affect all programs currently running against the data base. For this reason, transactions should be aborted only when absolutely necessary. Explicit syntax to invoke the abort transaction process is not provided. The abort transaction function is implicitly invoked when a program terminates and forces a close on a data base while in transaction state. Program termination outside of transaction state does not require recovery, however, the restart record is saved.

The abort transaction process will force a SYNCPOINT before performing the abort. As soon as all programs are out of transaction state, the SYNCPOINT occurs and a "pseudo" close is performed. All updated blocks in memory are written to disk, all control information is written to the <data-base-name>/DICTIONARY and the audit file is closed. Any programs attempting to access the data base will be hung "WAITING DATA BASE RECOVERY". RECOVER/DATA.BASE will be executed by the MCP. For Program-abort recovery, the "before" images in the audit trail will be used to back out all transactions since the last SYNCPOINT. The restart areas for the last completed transaction will be entered in the restart data set.

When the recovery is complete, RECOVER/DATA.BASE issues a special communicate to the MCP. This communicate updates the necessary information in the <data-base-name>/DICTIONARY.

If any other programs were accessing the data base at the time of the Program-abort, then a "pseudo" open is performed. The necessary <data-base-name>/DICTIONARY information is updated in memory and all the affected programs will be restarted. Affected programs will be informed that some of their transactions have been backed out by an ABORT exception returned by their next execution of BEGIN-TRANSACTION or CLOSE following the abort. When this occurs, the affected programs should retrieve their restart record from the restart data set to aid in redoing their backed-out transactions.

The paths of all programs currently using the data base are marked as undefined at the completion of Program-abort recovery.

DUMP RECOVERY

Dump recovery (or RECONSTRUCTION) may be executed any time the data base is not active. Prior to executing the recovery program, a good version of the data base should be loaded to the system. Both the cictionary file and all of the data files which have been updated since this version was created should also be loaded. All audit files created after this version must be available when requested.

The recovery program will apply all of the "after" images on the audit trail to the data base. It will continue until it finishes the last audit file (the operator tells the recovery program that there are no more audit files) or an I/O error is encountered. If an I/O error is encountered or an audit file is unavailable, the data base will be recovered to that point, all further audit files should be destroyed and all processing from that point must be re-run. The RESTART Data Set will contain the normal restart information for programs that were running at that point. If any of the audit files were terminated with a Clear/Start or a Program-abort, the normal Program-abort handling is invoked. Upon completion of Program-abort, dump recovery is resumed with the next audit file.

PARTIAL-DUMP RECOVERY

If a subset of the files in a data base (excluding the dictionary) have been lost or corrupted, the user may request that only those files be recovered from a previous dump through an RC message which specifies a list of the <file-name>s. The dictionary from that dump must be loaded to disk under the name <data-base-name>/OLD.DICT. Recovery will use the old dictionary as the source of the file records of the files to be recovered as well as the audit FPB and the audit serial information (or any other information). Dump recovery is performed on the specified files until they are up to the current dictionary level. At that time, Clear/Start or Program-abort recovery will be performed on the full data base if needed.

UNSUCCESSFUL RECOVERY

RECOVER/DATA.BASE does extensive integrity checking in an attempt to preclude or identify corruption in a data base. A side effect is that recovery is not restartable with this checking. When it is necessary to rerun recovery, the RC message is again entered, but with Switch 3 set to one (SW3=1) in order to reduce the integrity checking. The data base should still be intact, but recovery is not likely to detect previously existing corruption in this mode. If Program-abort or Partial-dump recovery or Clear/Start recovery continues to fail, DUMP recovery may be successful. A backup copy of the data base should be loaded and recovery started.

I/O ERROR DURING RECOVERY

The control information of each audit block contains the audit block number. It increases by one for each audit block written. If it ever increases by more than one for two consecutive blocks, one or more blocks of the audit may have been lost. The recovery routines check for this condition. Any I/O errors while attempting to read the audit trail will cause either incomplete or unsuccessful recovery. Incomplete recovery occurs on dump recovery, where on detecting an I/O error on the audit trail, the recovery program will back out to the last SYNCPOINT. Program-abort recovery, Clear/Start recovery, and Partial-dump recovery terminate unsuccesfully when they encounter an audit trail I/O error.

I/O errors on data base files always result in unsuccessful recovery.

Note: Copying the audit tape will NOT help if there is a faulty area in the middle cf an otherwise good audit tape. (All information after the faulty area will be lost).

APPENDIX A: AUDIT RECORD FORMAT

Audit records are variable length and variable format. The key to the length and format of the record is the type field which is the first item of each record. Each record must be capable of being traversed in both a forward and a backward direction. For this reason each record which is longer than just a type field will also end in the type field. There are two basic types of audit records: Control Records and Data Records.

CONTROL RECORDS

The control records consist only of a type field. On control records, the first digit of the type field is @B@. These records are:

TYPE	MEANING
@B1@	SYNCPOINT
@B2@	CONTROLPOINT
@B3@	DATA BASE CLOSE
@B4@	DATA BASE OPEN
@B5@	PROGRAM ABORT

DATA RECORDS

The data records are grouped into classes. The first digit of the type indicates the class. Immediately following the type field is an 8-bit structure number. The records always end with the 8-bit structure number followed by the 8-bit type field. The general format is:

```
TYPE : STRUCTURE : DATA (VARIABLE) : STRUCTURE : TYPE
```

The specific records and their contents are:

TYPE & MEANING	CONTENTS	LENGTH
DATA SET:		
@10@ -- DATA	PREVIOUS AUDIT SERIAL NUMBER	BIT(32)
RECORD AFTER	BLOCK LOGICAL ADDRESS	BIT(24)
IMAGE	DATA RECORD NUMBER	BIT(8)
	NEW DATA RECORD	BIT(STR.RECORD.SIZE)
@11@ -- DATA	PREVIOUS AUDIT SERIAL NUMBER	BIT(32)
RECORD BEFORE	BLOCK LOGICAL ADDRESS	BIT(24)
IMAGE	DATA RECORD NUMBER	BIT(8)
	OLD DATA RECORD	BIT(STR.RECORD.SIZE)
@12@ -- DATA	PREVIOUS AUDIT SERIAL NUMBER	BIT(32)
RECORD BEFORE	BLOCK LOGICAL ADDRESS	BIT(24)
AND AFTER	DATA RECORD NUMBER	BIT(8)
IMAGE	OLD DATA RECORD	BIT(STR.DATA.SIZE)
	NEW DATA RECORD	BIT(STR.DATA.SIZE)
INDEXES:		
@20@ -- INSERT	PREVIOUS AUDIT SERIAL NUMBER	BIT(32)
TABLE ENTRY	BLOCK LOGICAL ADDRESS	BIT(24)
	TABLE ENTRY NUMBER	BIT(12)
	NEW TABLE ENTRY	BIT(STR.RECORD.SIZE)
@21@ -- REMOVE	PREVIOUS AUDIT SERIAL NUMBER	BIT(32)
TABLE ENTRY	BLOCK LOGICAL ADDRESS	BIT(24)
	TABLE ENTRY NUMBER	BIT(12)
	OLD TABLE ENTRY	BIT(STR.RECORD.SIZE)
@22@ -- INDEX	PREVIOUS AUDIT SERIAL NUMBER	BIT(32)
SEQUENTIAL	BLOCK LOGICAL ADDRESS	BIT(24)
ROOT	OLD ROOT TABLE LOGICAL ADDRESS	BIT(24)
	NEW ROOT TABLE LOGICAL ADDRESS	BIT(24)
@23@ -- INDEX	PREVIOUS AUDIT SERIAL NUMBER	BIT(32)
SEQUENTIAL	BLOCK LOGICAL ADDRESS	BIT(24)
KEY CHANGE	TABLE ENTRY NUMBER	BIT(12)
	OLD KEY	BIT(STR.KEY.SIZE)
	NEW KEY	BIT(STR.KEY.SIZE)

TYPE & MEANING	CONTENTS	LENGTH
BLOCK CONTROL FIELDS:		
@30@ -- SET BLOCK TYPE	PREVIOUS AUDIT SERIAL NUMBER	BIT(32)
	BLOCK LOGICAL ADDRESS	BIT(24)
	OLD BLOCK TYPE	BIT(2)
	NEW BLOCK TYPE	BIT(2)
@31@ -- CHANGE TABLE NEXT	PREVIOUS AUDIT SERIAL NUMBER	BIT(32)
	BLOCK LOGICAL ADDRESS	BIT(24)
	OLD TABLE NEXT	BIT(24)
	NEW TABLE NEXT	BIT(24)
@32@ -- CHANGE TABLE PRIOR	PREVIOUS AUDIT SERIAL NUMBER	BIT(32)
	BLOCK LOGICAL ADDRESS	BIT(24)
	OLD TABLE PRIOR	BIT(24)
	NEW TABLE PRIOR	BIT(24)
@33@ -- SET TABLE NEXT & PRIOR	PREVIOUS AUDIT SERIAL NUMBER	BIT(32)
	BLOCK LOGICAL ADDRESS	BIT(24)
	OLD TABLE NEXT	BIT(24)
	OLD TABLE PRIOR	BIT(24)
	NEW TABLE NEXT	BIT(24)
	NEW TABLE PRIOR	BIT(24)
LIST:		
@40@ -- BEFORE IMAGE OF CONTROLINFO	PREVIOUS AUDIT SERIAL NUMBER	BIT(32)
	BLOCK LOGICAL ADDRESS	BIT(24)
	LIST TABLE NUMBER	BIT(8)
	OLDCONTROLINFO	BIT(72)
@41@ -- AFTER IMAGE OF CONTROLINFO	PREVIOUS AUDIT SERIAL NUMBER	BIT(32)
	BLOCK LOGICAL ADDRESS	BIT(24)
	LIST TABLE NUMBER	BIT(8)
	NEWCONTROLINFO	BIT(72)

TYPE & MEANING	CONTENTS	LENGTH
@42@ -- INSERT LIST RECORD	PREVIOUS AUDIT SERIAL NUMBER	BIT(32)
	BLOCK LOGICAL ADDRESS	BIT(24)
	LIST TABLE NUMBER	BIT(8)
	LIST RECORD NUMBER	BIT(8)
	NEW RECORD	BIT(STR.ENTRY.SIZE)
@43@ -- REMOVE LIST RECORD	PREVIOUS AUDIT SERIAL NUMBER	BIT(32)
	BLOCK LOGICAL ADDRESS	BIT(24)
	LIST TABLE NUMBER	BIT(8)
	LIST RECORD NUMBER	BIT(8)
	OLD RECORD	BIT(STR.ENTRY.SIZE)
@44@ -- REMOVE LIST RECORD AND DELETE	PREVIOUS AUDIT SERIAL NUMBER	BIT(32)
	BLOCK LOGICAL ADDRESS	BIT(24)
	LIST TABLE NUMBER	BIT(8)
	OLD CONTROLINFO & OLD RECORD	BIT(72 + STR.ENTRY.SIZE
@45@ -- STORE LIST TABLE AND INSERT LIST RECORD	PREVIOUS AUDIT SERIAL NUMBER	BIT(32)
	BLOCK LOGICAL ADDRESS	BIT(24)
	LIST TABLE NUMBER	BIT(8)
	NEW CONTROLINFO & NEW RECORD	BIT(72 + STR.ENTRY.SIZE)
@46@ -- CHANGE LIST RECORD	PREVIOUS AUDIT SERIAL NUMBER	BIT(32)
	BLOCK LOGICAL ADDRESS	BIT(24)
	LIST TABLE NUMBER	BIT(12)
	LIST RECORD NUMBER	BIT(8)
	OLD RECORD	BIT(STR.DATA.SIZE)
	NEW RECORD	BIT(STR.DATA.SIZE)

LIST HEAD:
Note: The parent record is being audited

TYPE & MEANING	CONTENTS	LENGTH
@50@ -- LIST HEAD AFTER IMAGE	PREVIOUS AUDIT SERIAL NUMBER	BIT(32)
	BLOCK LOGICAL ADDRESS	BIT(24)
	DATA RECORD NUMBER (PARENT IS A DATA SET) OR LIST TABLE NUMBER (PARENT IS A LIST)	BIT(12)
	EMBEDDED LIST HEAD OFFSET	BIT(16)
	FILLER (VALUE=0) (PARENT IS A DATA SET) OR LIST RECORD NUMBER (PARENT IS A LIST)	BIT(8)
	NEW LIST HEAD	BIT(64)
@51@ -- LIST HEAD BEFORE IMAGE	PREVIOUS AUDIT SERIAL NUMBER	BIT(32)
	BLOCK LOGICAL ADDRESS	BIT(24)
	DATA RECORD NUMBER (PARENT IS A DATA SET) OR LIST TABLE NUMBER (PARENT IS A LIST)	BIT(12)
	EMBEDDED LIST HEAD OFFSET	BIT(16)
	FILLER (VALUE=0) (PARENT IS DATA SET) OR LIST RECORD NUMBER (PARENT IS A LIST)	BIT(8)
	OLD LIST HEAD	BIT(64)

RECORD ALLOCATION:

TYPE & MEANING	CONTENTS	LENGTH
@60@ -- UPDATE NEXT AVAILABLE AND HIGHEST OPENED	PREVIOUS AUDIT SERIAL NUMBER	BIT(32)
	BLOCK LOGICAL ADDRESS	BIT(24)
	OLD NEXT AVAILABLE	BIT(32)
	NEW NEXT AVAILABLE	BIT(32)
@61@ -- UPDATE NEXT AVAILABLE	PREVIOUS AUDIT SERIAL NUMBER	BIT(32)
	BLOCK LOGICAL ADDRESS	BIT(24)
	OLD NEXT AVAILABLE	BIT(32)
	NEW NEXT AVAILABLE	BIT(32)

TYPE & MEANING	CONTENTS	LENGTH
@62@ -- RETURN NEXT AVAILABLE	PREVIOUS AUDIT SERIAL NUMBER	BIT(32)
	BLOCK LOGICAL ADDRESS	BIT(24)
	NEW NEXT AVAILABLE	BIT(32)
	OLD NEXT AVAILABLE	BIT(32)
@63@ -- OPEN NEW AREA	AREA NUMBER	BIT(8)

INDEX SPLITS & COMBINES:

TYPE & MEANING	CONTENTS	LENGTH
@70@ -- INSERT INTO FRONT OF TABLE	PREVIOUS AUDIT SERIAL NUMBER	BIT(32)
	BLOCK LOGICAL ADDRESS	BIT(24)
	NUMBER OF ENTRIES TO MOVE	BIT(12)
	SPLIT ENTRIES	BIT(# ENTRIES TO MOVE * STR.RECORD.SIZE)
	NUMBER OF ENTRIES TO MOVE	BIT(12)
@71@ -- INSERT INTO BACK OF TABLE	PREVIOUS AUDIT SERIAL NUMBER	BIT(32)
	BLOCK LOGICAL ADDRESS	BIT(24)
	NUMBER OF ENTRIES TO MOVE	BIT(12)
	SPLIT ENTRIES	BIT(# ENTRIES TO MOVE * STR.RECORD.SIZE)
	NUMBER CF ENTRIES TO MOVE	BIT(12)
@72@ -- REMOVE FROM FRONT OF TABLE	PREVIOUS AUDIT SERIAL NUMBER	BIT(32)
	BLOCK LOGICAL ADDRESS	BIT(24)
	NUMBER OF ENTRIES TO MOVE	BIT(12)
	SPLIT ENTRIES	BIT(# ENTRIES TO MOVE * STR.RECORD.SIZE)
	NUMBER OF ENTRIES TO MOVE	BIT(12)
@73@ -- REMOVE FROM BACK OF TABLE	PREVIOUS AUDIT SERIAL NUMBER	BIT(32)
	BLOCK LOGICAL ADDRESS	BIT(24)
	NUMBER OF ENTRIES TO MOVE	BIT(12)
	COMBINE ENTRIES	BIT(# ENTRIES TO MOVE * STR.RECORD.SIZE)
	NUMBER OF ENTRIES TO MOVE	BIT(12)

APPENDIX B: SAMPLE DASDL DECLARATIONS

The following is a skeleton of a sample DASDL for a data base using Audit and Recovery:

```
PARAMETERS (SYNCPOINT      = 100 TRANSACTIONS,
            CONTROLPOINT   = 100 SYNCPOINTS);
OPTIONS (AUDIT);


RESTARTAREAS restart-data-set
     (
     TC "TRANSACTION COUNTER BUMPED BY ONE DURING EACH
       TRANSACTION BY THE USER PROGRAM"
     NUMBER (6)

     ALPHAID "UNIQUE ID FOR EACH PROGRAM"

             ALPHA (18);

USERINFO
     "INFO PROGS NEED TO RESTART"
     ALPHA (240);

RESTARTSET SET OF RESTARTAREAS
     KEY IS (ALPHAID);
          •
          •
          •
AUDIT TRAIL(
      FAMILYNAME  =  AUDITPACK,
      AREAS  =  100, AREALENGTH  =  50 BLOCKS
      BLOCKSIZE  =  3000 BYTES);
```

APPENDIX C: SAMPLE PROGRAM

The following is a skeleton of a restartable data management COBOL program. The program reads an input tape. Each block contains enough information to update one member of the data management data set PERSONNEL of data base COMPANYX. The updating of one member is a transaction. The point of restart code is to skip the blocks that have already been processed.

NOTE: The cards with an asterisk (*) in the first column are generated from the data base description automatically by the COBOL compiler.

```
IDENTIFICATION DIVISION.
      .
      .
      .
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
      SELECT TF ASSIGN TO TAPE.
      SELECT PRINT ASSIGN TO PRINTER.
      .
      .
      .
DATA DIVISION.
FILE SECTION.
FD TF.
01 BLOCKBUF.
      .
      .
      .
DATA-BASE SECTION.
DB COMPANYX.
01 PERSONNEL INVOKE PERSONNEL.

 *    .
 *    .
 *    .
01 RESTARTAREAS INVOKE RESTARTAREAS.
 * 01 RESTARTAREAS DATA SET.
 *    RESTARTSET SET OF RESTARTAREAS KEY IS ALPHAID.
 *       02  TC              PIC 9(6).
 *       02  ALPHAID         PIC X(18).
 *       02  USERINFO        PIC X(240).
WORKING-STORAGE SECTION.
```

```
77 BLOCKSIN         PIC 9(5) COMP-1 VALUE 0.
77 SKIPNUM          PIC 9(5) COMP-1 VALUE 0.
77 BLOCKNUM         PIC 9(5) COMP-1 VALUE 0.
77 I                PIC 9(5) COMP-1 VALUE 0.
77 ABORTING         PIC 9(5) COMP-1 VALUE 0.

      .
      .
      .
PROCEDURE DIVISION.
THEONLY SECTION.
DUMMYLABEL.
    OPEN      INPUT      TF.
    OPEN      OUTPUT     PRINT.
    OPEN      UPDATE     COMPANYX
    ON EXCEPTION GO DIE.
    MODIFY RESTARTAREAS VIA RESTARTSET AT RESTARTALPHAID = "MYID"
    ON EXCEPTION CREATE RESTARTAREAS
    MOVE "MYID" TO RESTARTALPHAID ELSE GO RESTARTED.
MAINLOOP.
    PERFORM READIT.
    GO TO A-TRANSACTION.

 *
DIE.  STOP RUN.
 *
READIT.

     READ TF AT END GO CLOSE-DB.
     ADD 1 TO BLOCKSIN.

 *

A-TRANSACTION.
    BEGIN-TRANSACTION RESTARTAREAS NO-AUDIT
         ON EXCEPTION IF DMSTATUS (ABORT)
         GO ABORTED ELSE
         GO DIE.
     <data base operations to update a member of PERSONNEL using
     the information in BLOCKBUF.>
    ADD 1 TO TC.
    END-TRANSACTION RESTARTAREAS AUDIT
    ON EXCEPTION GO DIE.
    GO TO MAINLOOP.
RESTARTED.
    MOVE TC TO I.
    MOVE 0 TO BLOCKSIN.
    PERFORM READIT I TIMES.
    GO MAINLOOP.
```

```
ABORTED.
   MOVE 1 TO ABORTING.
   MODIFY RESTARTSET AT
       RESTARTALPHAID = "MYID"
       ON EXCEPTION IF DMSTATUS (NOTFOUND) CREATE RESTARTAREAS
       GO MAINLOOP ELSE GO DIE.

 *        BACKED OUT TO START OF PROGRAM.


    CLOSE TF RELEASE.
    OPEN INPUT TF.
    GO TO RESTARTED.
CLOSE-DB.
    BEGIN-TRANSACTION RESTARTAREAS NO-AUDIT
         ON EXCEPTION IF DMSTATUS(ABORT) GO ABORTED
         ELSE GO DIE.
         DELETE RESTARTAREAS
         ON EXCEPTION GO DIE.
    END-TRANSACTION RESTARTAREAS NO-AUDIT SYNC
         ON EXCEPTION GO DIE.
    CLOSE COMPANYX ON EXCEPTION GO DIE.
    <don't have to worry about ABORT EXEPTION because sync
     on previous END-TRANSACTION prevents basing out any
     transactions>
    STOP RUN.
```

APPENDIX D: PROGRAM SWITCHES

The recovery program uses the program switches for several functions. The use of each switch is described below.

SWITCH 1: Any nonzero value causes the recovery program to produce a printed trace of its operation. This trace contains a formatted copy of each record processed, the address and data of each data base block read or written, and a notation of each major decision that is made or step that is started.

SWITCH 2: Any nonzero value causes the recovery program to stop and await user action whenever a major step is started. There are four major steps, some of which are used in each type of recovery. The four steps are:

1. Process the audit trail forward to the end
2. Process the audit trail forward a specified number of syncpoints
3. Process the audit trail backward to the first syncpoint or beginning of this audit file
4. Skip backward on this audit file until two controlpoints have been found or the beginning of the file is reached.

SWITCH 3: Any nonzero value tells the recovery program to ignore the flag that indicates recovery has already been tried. It also restricts the integrity checking that can be done on dump recovery, partial dump recovery and program abort recovery. It should be used only when recovery must be restarted. This switch is interrogated only at the start of recovery.

SWITCH 4: Any nonzero value tells the recovery program that it is doing partial dump recovery. This switch should be set only by the MCP. This switch is interrogated only at the start of recovery.

SWITCH 5: This switch is used in conjunction with Switch 1 to control the printed trace of the recovery program. Switch 1 controls the invocation of the trace, and Switch 5 controls the contents of the trace. If Switch 5=0, then the trace will contain the history of the changes to all structures; if the switch is nonzero, then the user will be requested to supply (via the SPO) a list of the structures to be traced. This switch is interrogated only at the start of recovery.

All other switches are not used by the recovery program.

INDEX