northgate
INFORMATION SOLUTIONS

DRAFT COPY

RealityX FailSafe

V3.1

Reference Manual

Document No. UM70006220AX5
September 1992

Northgate Information Solutions UK Limited
Peoplebuilding 2
Peoplebuilding Estate
Maylands Avenue
Hemel Hempstead
Herts
HP2 4NW

Tel: +44 (0)1442 232424
Fax: +44 (0)1442 256454

www.northgate-is.com

Chapter 1 About this Manual

Chapter 2 Outline of Operation

Chapter 3 Guidelines for Managing Logs

Chapter 4 Setting Up Procedures

Chapter 5 Operating Procedures

Chapter 6 Recovery Procedures

Chapter 7 UNIX Tools

Chapter 8 TCL Commands

Chapter 9 Log Files

Chapter 10 Applications Interface

Chapter 1
About this Manual

This chapter describes the purpose and use of this manual, including:

- The intended readership and the knowledge they are assumed to have in order to use the manual.
- A list of associated manuals.
- A brief introduction to FailSafe operation.
- A summary of chapter contents,.
- A list of conventions used in the manual.

Purpose

This manual contains the information necessary to administer and operate FailSafe for a Reality X database.

Intended Readership

The manual is aimed at:

- The System Administrator responsible for setting up and maintaining the FailSafe pair.
- Operators responsible for the day-to-day operation of the FailSafe database.
- Programmers responsible for creating or modifying applications to include transactions.

Assumed Knowledge

It is assumed that the reader understands the basics of UNIX and Reality X administration and has appropriate knowledge of TCL, ENGLISH, DATA/BASIC and PROC. Detailed information on these subjects can be obtained from the manuals listed later in this chapter.

References

The following manuals provide reference information for the commands and procedures described in this manual.

Administrator's Guide to Reality X

Reality X Differences Supplement

You also need a set of REALITY Release 7.0 manuals which are used in conjunction with the *Reality X Differences Supplement* to provide information on the RealityX applications environment. A list of Release 7.0 manuals is given in the supplement.

For information on the UNIX environment, refer to the set of reference manuals supplied with your UNIX system

Comment Sheet

A Comment Sheet is included at the front of this manual. If you find any errors or have any suggestions for improvements in the manual please complete and return the form. If it has already been used then send your comments to the Technical Publications Manager at the address on the title page.

Introducing FailSafe Operation

FailSafe operation is an optional facility supported by RealityX which maintains a two identical databases for one set of users. One database is the live one to which users can log on and carry out update operations. The other database is maintained as a standby to which users can switch if the live database fails. The live database is designated the 'primary' and the standby duplicate database is designated the 'secondary'.

On a machine containing multiple databases it is not necessary for all live databases to operate in FailSafe mode, nor is it necessary to maintain all primary or all secondary databases on the same machine. RealityX FailSafe software supports complete flexibility in the setting up and location of FailSafe databases pairs across two or more machines. FailSafe operation is an extension of the RealityX Transaction Logging software.

The consistency and integrity of a FailSafe database can be maintained by the Transaction Handling facility; a standard feature of RealityX.

Transaction Handling

Transaction Handling maintains a sequence of updates to a database as a single 'transaction'. The contents of the transaction are application-defined using two commands, 'transaction start' and 'transaction end'. If the transaction is not completed, all updates since its start are deleted and the original data items restored (rolled back) to maintain the consistency of the database.

Transaction Logging

Transaction Logging saves updates to the primary database and logs them on disk. Initially all updates are logged in raw partitions, called 'raw logs', one locally on the machine containing the primary database and the other, remotely via a communications link to the machine containing the secondary database. Normally there is one raw log per system

Independent updates and completed transactions logged in the local and remote raw logs are then written to a log file called a clean log where they are stored for back-up purposes. One clean log must be provided for each database, primary and secondary.

The FailSafe logging mechanism maintains the secondary as a real-time duplicate of the primary by applying all updates logged in the secondary clean log to the secondary database

This comprises:

Chapter 1 (this chapter).

Chapter 2, Introduction, contains an overview of FailSafe and a description of Transaction Logging and how it is used to implement FailSafe operation. It also describes the Transaction Handling mechanisms, used to maintain a consistent database.

Chapter 3, Guidelines for Managing Logs, contains recommendations to the System Administrator on estimating the size of the raw log and clean log partitions, and administering their use to facilitate maximum operating efficiency of Transaction Logging.

Chapter 4, Setting Up Procedures, describes the procedures used by the System Administrator to configure the systems and databases for FailSafe operation.

Chapter 5, Operating Procedures, describes procedures and facilities used by the System Operator to enable efficient day-to-day operation of a FailSafe database.

Chapter 6, Recovery Procedures, describes the facilities available to the System Administrator and/or Operator for recovering a FailSafe database and restoring it to its most current, consistent and predictable state after a system failure.

Chapter 7, UNIX Tools, describes the utilities available to administer a FailSafe pair.

Chapter 8, TCL Commands, details in alphabetical order the TCL commands available to run a FailSafe configuration.

Chapter 9, Examining Log Files, describes the purpose and contents of various log and history files supported by Reality X and explains how ENGLISH can be used to examine them.

Chapter 10, Applications Interface, is directed at programmers and describes the use of transactions within DATA/BASIC, PROC, TCL and ALL applications.

Appendix A, Error Messages, lists of error messages that may be generated in a FailSafe configuration and suggests actions that should be taken in response.

Appendix B, Installation of Transaction Handling and Logging, details the procedure to follow in order to install transaction processing on a system.

Glossary and Index are included at the end of the manual.

Conventions

This manual uses the following conventions:

`Text` — Bold text shown in this typeface is used to indicate input which must be typed at the terminal.

`Text` — Text shown in this typeface is used to show text that is output to the screen.

Bold text — Bold text in syntax descriptions represents characters typed exactly as shown. For example

WHO

text — Characters or words in italics indicate parameters which must be supplied by the user. For example in

LIST *file-name*

the parameter *file-name* is italicized to indicate that you must supply the name of the actual file defined on your system.

Italic text is also used for titles of documents referred to by this document.

{Braces} — Braces enclose options and optional parameters. For example in

BLIST {**DICT**} *file-name item-id* {**(***options*}

- the word **DICT** can optionally be typed to specify the dictionary of the file.
- *file-name* and *item-id* must be supplied
- one or more single-letter options can be included, as defined for the command; these must be preceded by an open parenthesis, can be given in any order, and are not separated by spaces. Any number of options can be used except where specified in text.

[param | param]	Parameters shown separated by vertical lines within square brackets in syntax descriptions indicate that at least one of these parameters must be selected. For instance, [**THEN** *statements* | **ELSE** *statements*] indicates that either a THEN clause or an ELSE clause must be included (or both).
...	In syntax descriptions, indicates that the parameters preceding can be repeated as many times as necessary.
SMALL CAPITALS	Small capitals are used for the names of keys such as RETURN.
CTRL+X	Two (or more) key names joined by a plus sign (+) indicate a combination of keys, where the first key(s) must be held down while the second (or last) is pressed. For example, CTRL+X indicates that the CTRL key must be held down while the X key is pressed.
Enter	To enter means to type text then press RETURN. For instance, 'Enter the WHO command' means type **`WHO`**, then press return. In general, the RETURN key (shown as ENTER or ¿ on some keyboards) must be used to complete all terminal input unless otherwise specified.
Press	Press single key or key combination but do not press RETURN afterwards.
X'nn'	This denotes a hexadecimal value.

Chapter 2
Outline of Operation

This chapter provides an elementary introduction to the operation of Reality X FailSafe and the associated Transaction Logging and Handling facilities. The description is given in the following order:

- Transaction Handling. The concept of a transaction is explained with a simple example, and the facilities available to applications programmers are described.

- Transaction Logging. The basic elements of logging to the raw log and logging to clean logs are discussed.

- FailSafe operation. The purpose and operation of logging in a FailSafe pair is described. The description is extended to multiple databases.

- Database Recovery. Only a brief overview of recovery methods and procedures is provided here. For detailed procedures you must refer to Chapter 6.

Reality X FailSafe software maintains two identical databases; one as the live database to which users log on and one as a duplicate of the live database which is used as a standby. Users can then switch to the standby if the live database fails. The two databases are normally located on different machines so that one remains functional if a machine crashes. The live database is designated the 'primary' and the standby is designated the 'secondary'

The Reality X FailSafe software logs updates on the primary database to two log files. One is associated with the primary database and one is associated with the secondary. The logging mechanism which saves the updates in the secondary's log, also applies them to the secondary database to maintain it as a real-time duplicate of the primary. The FailSafe logging software is an extension of the Transaction Logging software available for a stand-alone database.

The consistency and integrity of the primary and secondary databases when performing a set of interdependent updates (a transaction) are maintained by Transaction Handling, if transactions have been incorporated into the application being run. Transaction Handling is a standard facility of Reality X.

Transaction Handling

Transaction Handling is a standard software facility on Reality X by which a database is maintained in a consistent and predictable state when performing transactions. This section contains a description of what it is and how it works.

What is a Transaction

A transaction is a set of related updates made to a database which can be logically grouped together by Transaction Handling 'start' and 'end' commands. Each update is a single change made to the database, from DATA/BASIC, PROC, TCL or ALL, by creating, altering or deleting an item. Individual updates not grouped into a transaction are defined as being 'independent'.

The relationship between updates belonging to a transaction and logically grouped by transaction start and end commands may be defined as follows:

- if one update within the transaction is applied to the database, then all of the remaining updates within the transaction must be applied in order to maintain a consistent database.

The following example may make the concept clearer:

Example of a Transaction

Consider a stock control program which generates a set of updates from a single input; first to an Orders file, then to a Customer file, and then finally to a Stock file. The program can be considered in four stages.

1. Details of an order are entered.

2. The Orders file is updated with the name and address of the customer, the goods ordered, the price and the date of order.

3. The Customer file is updated with the date of order, the goods ordered and the price, so that an invoice can be produced

4 The company Stock file is updated, subtracting the quantity of goods ordered from the current stock.

If the program were aborted after stage 2, this would mean an order would be sent out, but the customer would not receive the invoice produced by stage 3 (Customer file) and the Stock file would not be amended, causing 'out of stock' problems in the future.

To maintain a consistent database, Stages 1 to 4 must all be completed, that is, they must be maintained as a single transaction. Transaction Handling facilities are provided to do this.

The creation of transactions by applications programmers is discussed in Chapter 10.

What is Transaction Handling?

Transaction Handling ensures that the updates defined as belonging within a transaction are maintained together as a set, so that, if a transaction is not completed, the updates made since the start of the transaction are deleted from the database and the pre-updated items are restored. This maintains the database in a predictable and consistent state.

Transaction Handling also suspends the release of item locks set within transactions. These remain locked until the end of a transaction. This prevents inconsistencies in data due to attempted simultaneous update of one or more items by processes which are not involved in the transaction.

Transaction Handling supports three transaction boundary commands.

TRANSTART	which marks the start of a transaction.
TRANSEND	which marks the end of a successful transaction, i.e. the transaction is committed.
TRANSABORT	which marks the end of an unsuccessful transaction, i.e. the transaction is rolled back.

These boundary markers are implemented as TCL commands, DATA/BASIC statements and ALL functions. Refer to Chapter 10 for a more detailed description.

A fourth transaction command, TRANSQUERY, can be used to find out the transaction status of the current port. This can be executed, either by a TCL command or a DATA/BASIC function.

The transaction boundary markers, TRANSTART, TRANSEND and TRANSABORT, can be used to update existing application code to incorporate transactions. This may (in some cases) require some restructuring of the application in order to collect related updates together, so that they are performed in sequence and can therefore be defined as a transaction. When designing applications to incorporate transactions, the definition of transactions should be an integral part of the design.

It is important that transactions are made as short as possible in order to minimise the effect of the Transaction Handling mechanism on the overall efficiency of the system.

Executing a Transaction

A transaction is started by executing a TRANSTART operation from ALL, DATA/BASIC, TCL, or PROC and completed by executing a TRANSEND.

During each transaction the following events occur:

- Whenever a database update occurs within the transaction, a 'Before' image is copied into a central log on disk called the 'raw log' (See Figure 2-1). Each 'Before' image contains enough information to reverse the effect of the change to the database brought about by the update. For example, if you update an existing item, the 'Before' image that is logged contains a copy of the item before it was changed. If you create an item, the 'Before' image that islogged is 'delete item'. 'Before' images are only held for the duration of the transaction.

- Items locked in a transaction are kept locked until a TRANSEND or TRANSABORT is issued. This prevents other transactions or processes reading updated items while the current transaction is still in progress, thus preventing dirty and unrepeatable reads (see Glossary). All item locks set during a transaction remain locked until the 'transaction commit' or 'transaction abort', after which they are released.

Note: It is important that all processes use item locking to prevent dirty reads. Items that are updated without having been locked previously are not guaranteed to be recovered correctly by a TRANSABORT

- TRANSEND generates a 'transaction precommit', followed by a 'transaction commit'. Transaction commit indicates that the transaction is completed. 'Before' images are only held in the raw log for the duration of the transaction. Once a transaction is committed, i.e. the 'transaction commit' is logged in the raw log, the 'Before' images are discarded and all the item locks held by the transaction are released.

Aborting a Transaction

If a transaction is aborted, either deliberately by TRANSABORT or by a forced log off occurring mid-transaction, the transaction is 'rolled back' by applying the 'Before' images to the database in reverse order. The 'Before images are then discarded.

Forced Abort and Logoff of a Transaction

Two conditions will force a transaction to abort automatically and logoff.

1. If the raw log becomes excessively full (>85%), then the oldest transaction, and therefore the longest, is forcibly aborted and logged off.

2. A transaction longer than a pre-defined timeout period will be forcibly aborted. The timeout period is specified in minutes in the environmental variable REALTXNTIMEOUT which should be set up by the system administrator in /etc/realityrc. The default is 8 hours (480 minutes). If REALTXNTIMEOUT is changed, the new value becomes effective the next time the central daemon is started.

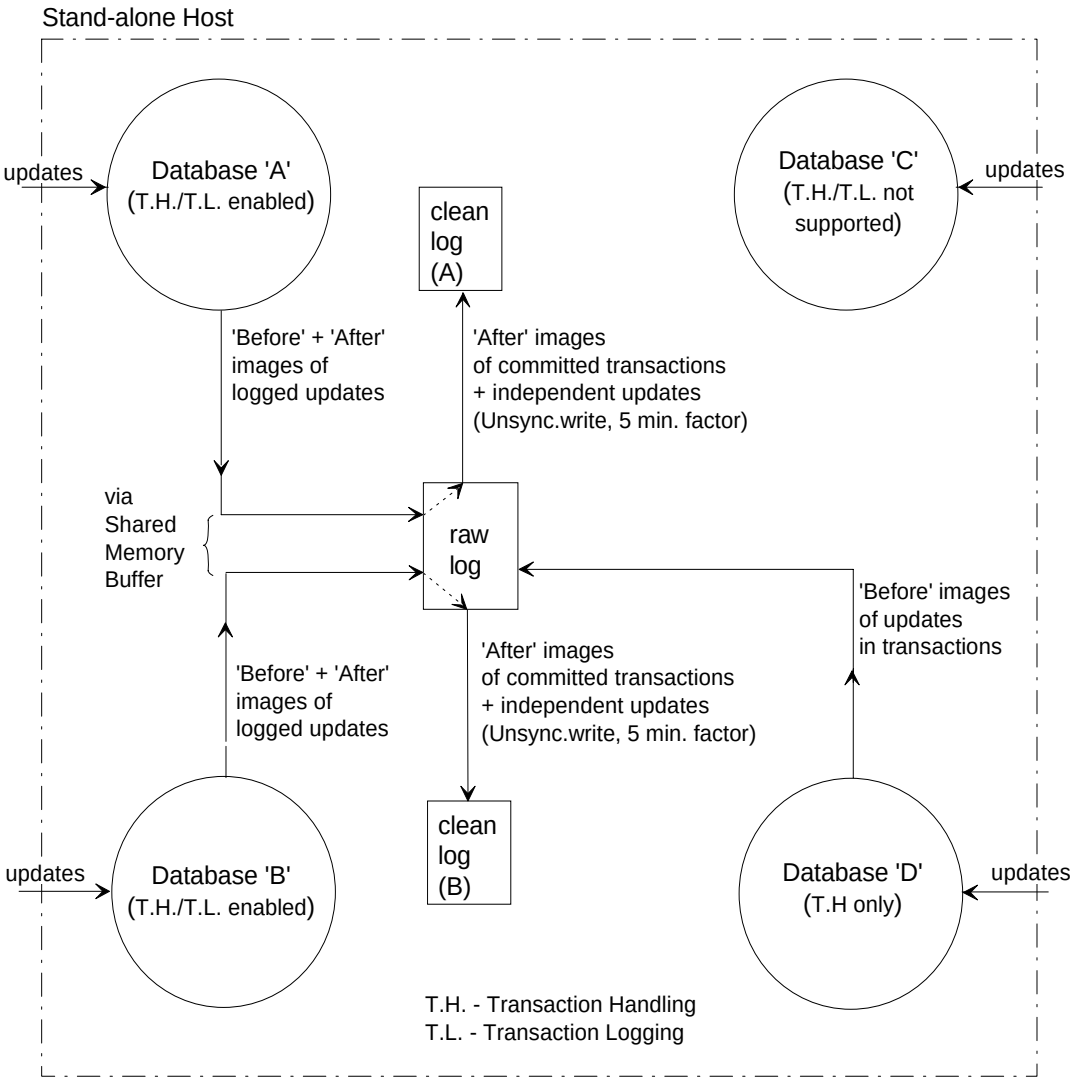


Figure 2-1 Transaction Handling/Logging Paths

Transaction Logging

What is Transaction Logging?

Transaction Logging is a software facility which permanently saves completed transactions and independent updates to disk, so that, in the event of a database failure, the logged updates can be restored so as to recover the most recent version of the database.

With transactions defined, Transaction Handling and Transaction Logging operate together to ensure the consistency of the database in the event of a failure. Only committed transactions are logged. Aborted transactions, initiated by a TRANSABORT or program failure, are rolled-back to restore the database to the state it was in before the current transaction was started.

If a database failure, the database can be recovered by restoring the most recent back-up tape, then restoring all committed transactions and independent updates logged since the back-up was made.

How Transaction Logging Works

Figure 2-1 illustrates the logging paths for Transaction Handling/Logging on a system containing multiple databases. Note that not all databases on a system have support Transaction Logging or Logging. In the example, only Databases 'A' and 'B' support Transaction Handling and Transaction Logging. Database 'C' does not support either, and Database 'D' supports Transaction Handling only, so that only 'Before' images are written to the raw log.

Images Written to the Raw Log

Transaction Handling/Logging writes two images to the raw log for each logged update, a 'Before' image and an 'After' image. These are defined, as follows:

'Before' image	This defines how the updated item is restored to its original value. This is supported by the Transaction Handling mechanism to roll-back the updated item to its original value if the system or process fails in mid-transaction or in the event of a 'transaction abort'.
'After' image	This defines the item update. This is saved to enable recovery of the updated item in the event of a system/program failure.

For a committed transaction three transaction boundary images are written to the raw log, Start, Precommit and Commit. TRANSEND generates a Precommit, then a Commit.

For an aborted transaction two transaction boundary images are written to the raw log, Start and Abort.

The Raw Log

The raw log is a central log located on disk (normally one per system) to which is written 'Before' and 'After' images for all recently logged updates for all databases on a system. It is located in a raw partition, normally on a disk dedicated to logging (the log disk), and operates as a cyclic buffer. The raw partition is created by the administrator when setting up Transaction Logging. Chapter 3 discusses the sizing of the raw partition.

The raw log is the key component of the Reality X Transaction Handling and Logging system. The complex cyclic queue structure ensures efficient and secure storage of 'Before' and 'After' images for all active and recently committed transactions and independent updates. It supports the following facilities:

- Roll-back of transactions in the event of an abort or program failure.
- Buffering for 'After' Images before they are written to a clean log.

Writing to the Raw log

'Before' and 'After' images are written to the raw log via a cyclic buffer in shared memory. This buffer is common to all databases. The shared memory buffer is maintained as an image of the current write point on the raw log. Each user copies images to the shared memory buffer and the buffer is 'flushed' to raw disk, either periodically or by a transaction being completed. The transfer to disk from the shared memory buffer to the raw log is carried out by a synchronised write.

The two operations of copying a committed transaction to the shared memory buffer and logging it to disk can be synchronised or unsynchronised. Synchronised logging is referred to a FULL logging mode and unsynchronised logging is referred to as BRISK logging. The mode is set as part of the Transaction Logging setting up procedure using the **mklog** utility. See Chapter 7 for details.

In FULL logging mode, the shared memory buffer is flushed each time a 'transaction commit' image is copied to it. This ensures that all committed transactions are written safely to disk immediately so that they cannot be lost in a system crash. However, this has a performance overhead, in that the user process waits until the synchronised write to disk is completed before continuing. Note that, on receiving the 'commit' image all previous images in the shared memory buffer are flushed to disk. If transactions are not in use, i.e. no 'transaction commits' copied to shared memory, the buffer is flushed periodically only, or when full.

In BRISK logging mode, the shared memory is not flushed by a 'transaction commit', but is flushed periodically or when full. This improves up the performance of the database, but does mean there is a potential for losing committed transactions from the memory buffer during the period between flushes.

Writing to Clean Log

When a transaction is committed, that is, the 'transaction commit' image is written to the raw log, all 'After' images and 'transaction boundary' images for that transaction are written to a clean log file which is assigned to database on which the transaction is performed. All images relating to that transaction are then cleared from the raw log, i.e. 'Before' images are discarded.

'Before' and 'After' images are maintained in the raw log, after being written to a clean log, for just under 5 minutes. This is because the writing to the clean log is via memory buffers and is unsynchronised. It is, therefore, necessary to allow 5 minutes to ensure that the images have reached the safety of the clean log disk and will not be lost if the system crashes. If the system does crash, the images can then be recovered from the raw log and written to the clean log. After 5 minutes all committed transactions and independent updates in the raw log are cleared.

Clean log

A clean log is a serial file (one per database) which is created to hold committed transactions and independent updates for one associated database. Transactions are written to it in the order in which they are committed. Each transaction on a clean log file consists of a copy of the sequence of logged 'After' images and 'transaction boundary' images for that transaction. The contents of an inactive clean log can be used to restore updates to its associated database in the event of a system/program failure. Refer to the section on recovery at the end of this chapter. Operations on logs are not logged as this [redacted]leads to obvious conflict (e.g. CLEAR-FILE on a clean log).

FailSafe Operation

FailSafe operation maintains two identical databases for one set of users; the live database that is, the one to which users log on and a duplicate of the live database which is used as a standby and is not normally accessed by users. The standby is normally on a different machine from the live database to guard against machine failure. If the live database fails, then users can switch to the standby and continue operating with only a short break in service and minimal loss of data. The live database is referred to as the 'primary' and the standby database is referred to as the 'secondary '.

FailSafe is an extension of Transaction Logging on a stand-alone machine. Figure 2-2 illustrates the logging path in a FailSafe configuration.

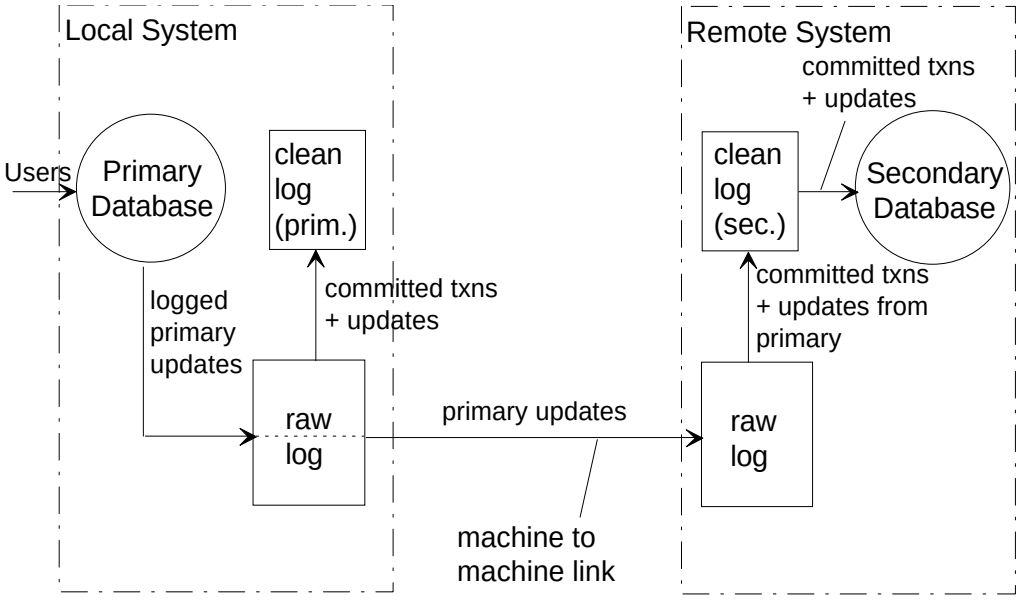


Figure 2-2 Logging Path of a FailSafe Pair

'Before' and 'After' images for all primary database updates are written to two raw logs; one on the local machine containing the primary database and one on the remote machine containing the secondary database, via a machine to machine communications link.

Committed transactions and independent updates in the local and remote raw logs, respectively, are then written to clean logs for the primary and secondary databases, respectively. Transactions and independent updates logged to the secondary's clean log are also applied to the secondary database to shadow the primary.

Reality X supports multiple databases on one system. It is not necessary for all databases on a system to operate in FailSafe mode. Some may operate in FailSafe mode while others may be stand-alone unresilient databases, with or without Transaction Logging. There are also no technical limitations on where the primary and secondary databases in a FailSafe pair are located. However, it is necessary that each half of a pair should be on a different system, so that in the event of a system failure, one database remains in service, otherwise the purpose of FailSafe operation is negated. Unrelated primary and secondary databases may be located on the same system.

Figure 2-3 illustrates FailSafe operation in a multiple database configuration, showing two FailSafe pairs (Databases A and B) and an unresilient database not using transaction boundaries (Database C). Note that FailSafe operation can take place in both directions across the machine to machine link. Updates from local and remote primaries are stored in the same raw log.

While operating in FailSafe mode, the secondary database is closed to all users except the system super-user (root). Even the super-user should exercise extreme caution as update operations on the secondary may lead to loss of synchronisation between the two databases.

If a primary database becomes unavailable, for example, due to a system crash, the secondary database can be converted to be the primary, without loss of transaction integrity and with minimum loss of data and service. The transfer of users to the secondary is a manual operation. If a secondary database becomes unavailable, the primary continues unaffected as a stand-alone database.

The failed database, whether primary or secondary, can be recovered by restoring the most recent file-save and clean log(s). The restored database can then be re-introduced as a secondary and synchronised with the primary without affecting the users. FailSafe operation is then resumed. In the case of primary failure, primary/secondary roles will be reversed after recovery. Refer to Chapter 6 for details.

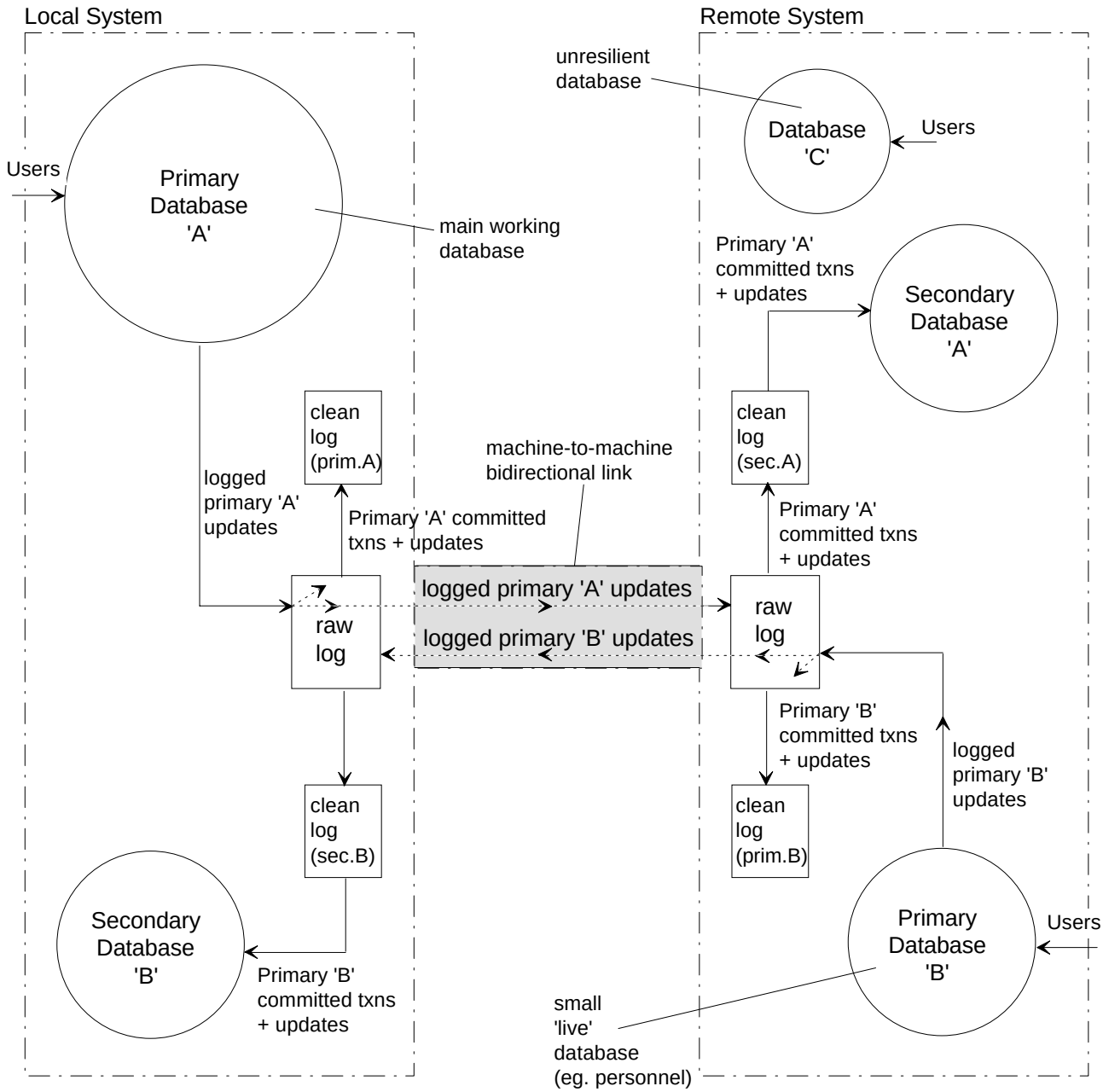


Figure 2-3 FailSafe Operation with Multiple Databases

Database Recovery

Recovery Methods

Currently, Transaction Logging supports three methods to recover the most recent, consistent and predictable version of a database. They are:

- Hit Process Recovery
- Full Recovery
- Selective Recovery

Hit Process Recovery

Hit Process Recovery is an automatic recovery mode which restores a database to a consistent state, after a process has crashed or been killed.

This recovery method is supported by Transaction Handling with or without Transaction Logging. Whem a failure occurs, it rolls back all updates in incomplete transactions, restoring their 'Before' images, so that the database(s) on the failed machine are restored to the consistent state that they were in before the incomplete transactions started. This uses Transaction Handling facilities only.

Full Recovery

Full Recovery is initiated manually when a system crashes and a database is corrupted. Either a TL-RESTORE or a TL-REDUAL command is used to initiate the restore all updates from clean logs to a database. The appropriate command is executed after the most recent version of the database has been restored from a FILE-SAVE. All After images held in the clean log(s) are applied to the partially restored database. TL-REDUAL not only restores the database, but re-establishes FailSafe operation, re-synchronising the primary and secondary databases.

Selective Recovery

Selective Recovery is a special case of the Full Recovery Method, used when only certain areas of a database need to be recovered. Like Full Recovery, it is initiated using the TL-RESTORE command, but instead of restoring updates to the whole database, it restores selected accounts and files only. A selection list is generated using the ENGLISH query language to operate on the clean log file. All selected 'After' images from the clean log(s) are then TL-RESTORE'd on the database.

Procedure to Recover a Database After a System Failure

In the event of a system/database failure causing data loss or corruption, the steps to recovery are as follows:

For a primary database/system failure:

1. Reverse the primary and secondary roles of the FailSafe pair using **fsadm** with the **-R** option on both machines.

2. Unlock the new primary using **unlockdbase** and tell users to switch to it.

3. If necessary, repair and restart the failed primary.

4. Restore the damaged database from the last back-up tape(s).

5. Restore updates logged in clean logs since the last back-up and up to, but not including the currently active clean log on the live database.

   A number of different procedures are available to do this. These involve the use of TL-RESTORE and/or TL-REDUAL. The method used depends partly on system limitations and partly personal preference.

6. Use TL-REDUAL to re-establish FailSafe operation, and re-synchronise the primary and secondary databases.

For a secondary database/system failure, the steps to recovery are similar to the primary, except that the failure does not affect users directly and therefore it is not necessary to carry out step 1. Also, there is no reversal of roles. After the secondary is repaired, it is restarted and resynchronised with the primary to restore FailSafe operation.

Refer to Chapter 6 for a detailed description of recovery procedures.

Chapter 3
Guidelines for Managing Logs

This chapter contains recommendations on the sizing and managing of the raw log and clean logs to ensure optimum operating efficiency of Transaction Logging on a system. Recommendations are given on:

- where the raw log and clean log directory should be located.
- What size the raw log should be.
- How to estimate the rate of clean log growth.
- How the clean log growth rate affects clean log switching and archiving.
- How many times to change clean logs each day.
- When to archive clean logs and why.
- What to name the clean log files.

Refer to Appendix B for details on setting up the raw log and clean log partitions

Locating of the Raw Log and Clean Logs

This section provides recommendations on where the raw log and clean logs should be located on the system disks.

Raw Log

The raw log, as its name implies, is created in a raw partition.

Clean Logs

Clean logs are maintained within a mountable file system which is created in a disk partition dedicated to clean logs.

> CAUTION
>
> It is mandatory that the raw log partition and the clean log file system partition are located on a disk which is separate from all standard UNIX partitions, swap partitions and logged databases.

Clean Log File System

The clean log file system consists of a three-level hierarchy, as follows:

- A 'clean log directory' which is the mount point for the file system.
- Clean log sub-directories; one for each logging database. Sub-directories are contained in the clean log directory.
- Clean log files; logs for a particular database are contained in the clean log sub-directory for that database

A typical clean log file system structure is illustrated below:

Typically, the clean log file system is mounted below root, but this is not mandatory. If the partition contains other file structures, the clean log directory may be mounted further down the disk file system.

Use of a Log Disk

Normally a disk, designated the Log Disk, is dedicated to the raw and clean logs. However, in small systems other user partitions with low utilisation may also be located on the Log Disk. Remember, though, that the accessing of non-logging related data/partitions on the Log Disk will impair the performance of Transaction Logging, particularly when using transaction boundaries.

Estimating Raw Log Size

The raw log must be large enough to retain all roll-forward information ('After' and transaction boundary images) and all roll-back information ('Before' images) until a transaction is ended or aborted and any associated 'After' images and transaction boundary images are securely in the clean log.

Note also that only about 85% of available raw log space may actually be used. If the raw log becomes excessively full (>85%), then the longest, and therefore the oldest, transaction is forcibly aborted and the process logged off from the database. This should be taken into account in the final specification of minimum raw log size.

The 5 Minute Factor

'After' images and transaction boundary images transferred to a clean log are maintained in the raw log for 5 minutes after being written, to ensure that they have been flushed from the UNIX system buffer and have reached the clean log. Writing to a clean log is not synchronised.

To meet this requirement the raw log should be made large enough to hold all roll-forward and roll-back images generated during the run period of the longest transaction expected to be on the system, plus 5 minutes.

Allowing for Longest Transaction

The minimum partition size required to operate as the raw log is determined by the longest transaction expected to be input to the raw log. This is because the raw log operates as a circular buffer and all buffer space between the transaction start and current update of an open transaction remains locked until the transaction is committed. Hence, although, shorter transactions and updates within the time span of a longer transaction may be successfully written to a clean log, the buffer space used by them is locked until the longest transaction is committed.

How Update Data is Stored

To calculate raw log size you need to appreciate how the update data is stored. For each item update within a transaction, Transaction Logging generates two images; a 'Before' image containing the update item before the update was made plus a header, and an 'After' image containing the update item after the update was made plus a header. An update outside a transaction only generates an 'After' image.

Size of Image Header

The image header contains information about the update, for example, user, account, file, time of update, port, etc. On average there are about 100 bytes of information in the header.

Transaction Boundary Images

For each completed transaction, Transaction Logging generates transaction boundary images TRANSTART, PRE-COMMIT and COMMIT. Each of these contains a header of approximately 50 bytes and a text string which is application determined. A total image size of 100 bytes is assumed in the calculations in this chapter.

Parameters Affecting Raw Log Size

To work out the minimum size of raw log, you need to estimate the following parameters during the peak work period of your databases:

U the maximum number of item updates per hour performed on your system.

I the average size of an updated item in bytes.

T the maximum number of transactions per hour.

t the estimated duration in minutes of the longest transaction.

These parameters are used to calculate the amount of disk space required for the raw log, as follows,

Maximum Number of Updates (A)

Maximum number of updates (A) to be held on the raw log is

$A = (5 + t) \cdot U/60$

U is divided by 60 to calculate the number of updates per minute. Five minutes is added to the estimated duration of the longest transaction, as the transaction is maintained in the raw log for an additional 5 minutes after it is written to the clean log to ensure that the write is successful.

Bytes Per Update (B)

Number of bytes (B) created by a single update is

$B = 2 \cdot (I+100)$

The figure of 100 bytes is added to the average item size (I) to allow for the header of an image in the raw log. Two images, 'Before' and 'After', are stored in the raw log for each update, hence the multiplication by two.

Bytes For Transaction Boundaries (C)

Growth in bytes (C) due to transaction boundary images generated during the longest transaction anticipated is

$C = (5 + t) \cdot (T \cdot 3 \cdot 100)/60$

The number of transactions (T) is multiplied by three, as there are three images per transaction (start, precommit and commit), then by 100 as each image is assumed to consist of 100 bytes, maximum. This total is divided by 60 to calculate the number of bytes generated per minute. This figure is then multiplied by (5 + t) to allocate space for the longest transaction anticipated.

Calculation of Raw Log Size (R)

Using the above calculations the minimum number of bytes of disk space required for the raw log partition is

$$R = (A \cdot B) + C$$

Without Transactions

Note that in this calculation we have assumed that all updates are made within transactions. If Transaction Handling is not used at all, the minimum raw log size can be half that of the above calculation, as only one image is logged for an independent transaction and there will be no transaction boundary images.

Minimum Size of Raw Log

The size of raw log (R) calculated here should be considered as the absolute minimum. It is recommended that where possible, the raw log should be double the calculated value to allow for worst case conditions. On systems where disk space is at a premium, a margin of at least 25% is strongly advised. Remember that automatic aborting and log off of transactions occurs when the raw log is >85% full.

> CAUTION
>
> A "raw log full" condition can lead to serious system performance problems and potential lock-up conditions. It is most important that the raw log be configured large enough in the first place as any resizing will require the Reality X system to be shut-down and the log disk re-partitioned.

Managing Clean Logs

A policy for managing clean logs on your system will depend on a number of factors. These include:

- Size of the clean log partition
- Growth rate of each clean log.
- Number of databases on the system
- Requirements for keeping old clean logs

The following sections deals with each of these factors. By considering these factors together you can establish a clean log cycling policy.

Clean Log Partition Size

The size of the clean log partition on the log disk is the primary factor in limiting the maximum size and number of clean logs which can be maintained on your system. This may be equal to the total storage space on the log disk minus the raw log partition. It is recommended that the log disk is dedicated to logging and there are only two partitions on it, raw log and clean log.

Esimating Clean Log Growth Rate (G)

The rate of growth of data in the clean log for a single database in bytes per hour is:

$G = ((I+130) \cdot u/d) + (3 \cdot t \cdot 100)$

where,

u is the estimated number of updates on the database per working day. This is divided by the number of hours in the working day to calculate the hourly rate.

d is the length of the working day for the database in hours.

I is the average size of an updated item in bytes. The figure of 130 is the overhead allowed for the header of a clean log image.

t is an estimate of the average number of transactions per hour on the database.

The expression $3 \cdot t$ calculates the number of transaction boundary images (3 per transaction, start, pre-commit and commit) stored in the clean log per hour. This is multiplied by the average size of a transaction boundary image (100 bytes) to calculate the total amount of transaction boundary information held in the clean log. The average size (100 bytes) of a transaction boundary includes a 50 byte overhead for the header.

Avoiding a Clean Log Full Condition

When the clean log partition becomes 70% full, warning messages are displayed at the system console and the raw log is locked. Clean log disk space must be released immediately otherwise logging will grind to a halt. The procedure to be carried out when a 'clean log 70% full condition' is received, is described in Chapter 5.

> CAUTION
>
> A "clean log partition full" condition will lead to serious performance problems and potential lock-up conditions on your system. It is very important that large clean logs are cleared from disk long before this condition becomes a possibility.

Multiple Databases

Where there are a number of databases on your system the growth of each clean log will contribute to the total 'rate of filling' of the clean log partition. The clean log cycling policy for all databases on the systems should be defined so as to avoid filling up the clean log partition to 70% full. It may be necessary to switch the clean logs more often on the database(s) with the largest growth rate to ensure that the total amount of clean log data on the system does not approach 70% of the partition size. Clean logs will also have to be cleared from the log partition more often.

Naming Clean Logs

In order to make it easy to identify a clean log and the date it was used, it is recommended that you establish a naming convention for clean log files used on your system.

Note: Clean log file names are limited to 13 characters and have the usual UNIX constraints. The slash '/' is a reserved character.

For example, the first part of a clean log file name might be a standard string such as CLOG, short for clean log. The latter part might be some form of alphanumeric identifier. What this will be depends on the clean log cycling policy for your system and whether you wish to archive files for security or audit purposes.

It is up to the Database Administrator to choose the naming convention most suited to the way Transaction Logging is used on the database.

No Archiving

If archiving of clean logs is not required, it is recommended you use one of two naming conventions. This depends on whether it is necessary to change logs more than once each working day.

The following naming conventions are recommended.

- CLOG-MON, CLOG-TUES, CLOG-WED and so on, one for each working day of the week.
- CLOG-MON-A, CLOG-MON-B, CLOG-TUES-A, CLOG-TUES-B, CLOG-WED-A CLOG-WED-B and so on, two, or more, for each working day of the week.

Archived Clean Logs

If archiving is required, it is recommended you include the date in the log file name as follows:

- CLOG11.02.91, CLOG12.02.91 and so on for a one log per working day database
- CLOGA11.02.91, CLOGB11.02.91, CLOGA12.02.91, CLOGB12.02.91, and so on where more than one log per working day is used.

Archived Logs from Multiple Databases

If you want to archive clean logs from more than one database on a system or from a FailSafe pair, it is recommended that you include an identifier in the clean log file names which associates each clean log with a database. It is recommended that you a clean log naming convention that does not require them to change the name of the file when it is archived.

As a filename in UNIX is restricted to 13 characters, it is improbable that a full database name can be included. Filenames of the following type are suggested.

LA11.02.91D1, LB11.02.91D1, LA11.02.91D2, LB11.02.91D2, where,

L	This prefix identifies that its a clean log (CLOG) file
A and B	specifies clean log A and clean log B, respectively.
11.02.91,	is the date when the clean log was filled.
D1, D2	specifies the associated database. This suffix identifier can be cross-referred to a database name.

Alternatively, it may be easier to manage clean logs if you archive all clean logs for different databases on different tapes,.

Viewing Clean Logs

The contents of a clean log can be viewed from the Reality X environment and can be accessed using standard ENGLISH verbs such as LIST. Details are later in this manual in the 'Log Files' chapter.

Log Archiving Policy

Database back-up and data security procedures will vary according to user requirements. Once a database has been backed up, the earlier clean log(s) are effectively redundant, however, some users may wish to keep clean logs for an extended time period to provide additional security or for auditing purposes. Larger systems may require logs to be archived during each working day to make space on the clean log partition. The policy is user determined.

The procedures to archive and retrieve clean logs are discussed further in the 'Operating Procedures' chapter

Chapter 4
Setting Up Procedures

This chapter outlines the procedures used to set up Transaction Logging on two separate systems/databases and then configure them to operate as a FailSafe pair. These are:

• Setting up Transaction Handling/Logging initially on each system.

• Defining the files to be logged

• Saving the live database to tape.

• Creating an identical database

• Configuring the secondary

• Configuring the primary

Appendix B is referenced. This details the procedure for installing Transaction Handling/Logging on your system

Commands and Utilities used for Setting Up FailSafe Operation

TCL Commands

The following TCL commands are used in this chapter as part of the setting up procedures.

- TL-CREATE-FILE
- TL-SET-LOG-STATUS

Full descriptions of these commands, including syntax and restrictions, are given in Chapter 8.

UNIX Tools

Also the following special UNIX tools are used.

- **fsadm**
- **mklog**
- **mkdbase**
- **killreal**

Full descriptions of these utilities, including syntax, are given in Chapter 7.

Setting Up a FailSafe Database

This section describes the procedures carried out to set up a pair of identical databases to operate as a FailSafe pair. For the purpose of this description, the two databases are identified as pdbase on the system phost, and sdbase on the system shost. Refer to Figure 4-1.

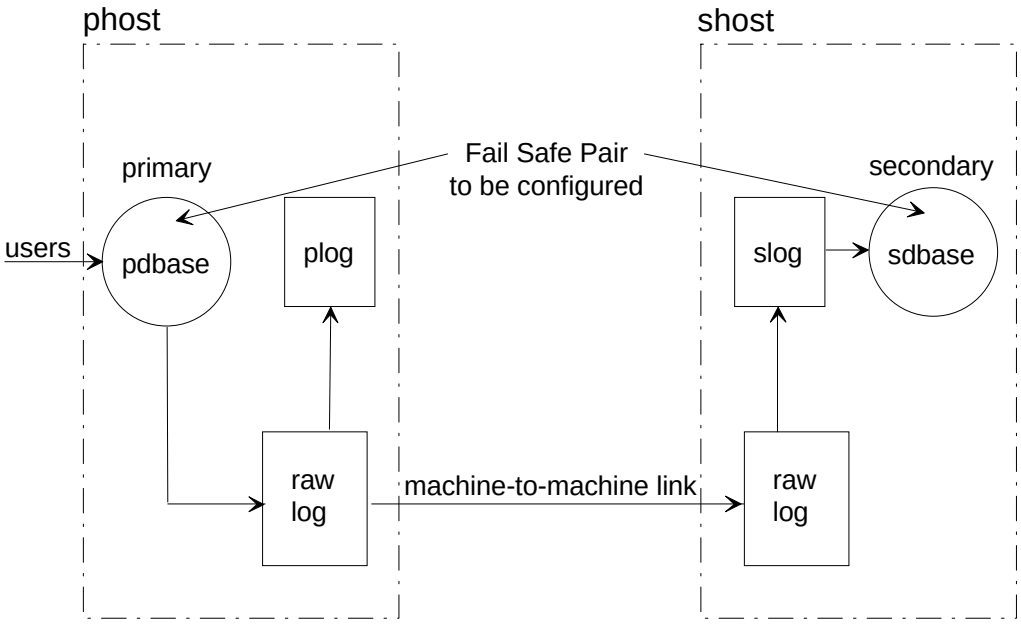


Figure 4-1 FailSafe Setup

It is assumed that pdbase is the live database which is to be configured as the primary. The other database, sdbase, is created as a duplicate of pdbase and will be configured as the secondary database. For the purpose of this example it is assumed that pdbase and sdbase are located in a directory /usr/dbases on their respective systems.

Note: Ensure that shost and phost are defined in the /etc/hosts file.

For the sake of clarity, these procedures identify the primary and secondary databases by different names (pdbase and sdbase). However, for optimum FailSafe configuration, it is recommended that you give them the same name. Differences in setting up databases with the same name and different names are highlighted in the following procedures.

The procedures consists of the following:

1. Set up Transaction Logging on each system .
2. Define the files to be logged on the unresilient database.
3. Save the unresilient database to tape.
4. Create an identical database.
5. Configure the secondary database.
6. Configure the primary database.

These are detailed in the following sub-sections.

Setting Up Transaction Handling/Logging

First ensure that both systems (phost and shost) are configured for Transaction Logging, with a raw log, a clean log partition and clean log file system. Systems will normally be configured by McDonnell Douglas support personnel when they are installed. However, in case re-configuration is necessary, the procedure is detailed in Appendix B.

If necessary configure the live database (pdbase) with a clean log sub-directory using **mklog**. Refer to Chapter 7.

Defining the Files to be Logged

Before creating a duplicate of the live database (pdbase) on the second system (shost), it is recommended that you define the log status of the files on pdbase. This is done using the TL-SET-LOG-STATUS command (See Chapter 8). Then when the database (pdbase) is copied to the shost, the log status is also duplicated.

Saving the Database to Tape

Having configured both systems to support Transaction Logging, you must now save the current live database (pdbase) to tape, as follows:

1. Log in to the local system (phost).
2. Run **reality** to enter pdbase and logon to SYSMAN.
3. Enter `INHIBIT-LOGONS *` to prevent any more users from logging on to the database.
4. Send a message asking users to log off. You can check who is logged on using LISTU.
5. After a reasonable period of time, log off any remaining users using the LOGOFF command.

6. Enter TL-STOP to disable logging.

7. Run the FILE-SAVE PROC to save pdbase onto tape and VERIFY-SAVE.

Creating an Identical Database

Having saved pdbase to tape, now create a second database (sdbase) on another system (shost) and restore the FILE-SAVE of pdbase onto sdbase, so that the two databases have identical user data. The procedure is as follows:

1. Log in to the second system (shost).

2. Change to the directory in which you wish to create the database.

3. Make a new Reality X database by entering

```
mkdbase sdbase
```

4. Lock the database using lockdbase. This prevents others from logging on, allowing only the database owner or super-user to access it.

5. Run **reality** to enter sdbase and logon to SYSMAN.

6. Mount the tape containing the FILE-SAVE of pdbase onto a tape unit and ensure that the unit is on-line.

7. Attach the tape unit to the system using T-ATT or ASSIGN.

8. Position the tape at the beginning of the files section by entering **`T-FWD`**, followed by **`T-RDLBL`**, then **`T-FWD`** again.

9. Restore the FILE-SAVE onto the newly created sdbase by entering

```
ACCOUNT-RESTORE * (O
```

10. Return to the UNIX shell and use **mklog** to create the clean log subdirectory for sdbase. See Chapter 7 for a description of **mklog**.

Note: Ignore the next step if the two databases have been given the same name

11. If the two databases have different names then you must now ensure that file items referencing sdbase are changed after the FILE-SAVE of pdbase has overwritten them. These include: the Reality X ROUTE-FILE items, the CUSTOMER-SYSTEM-IDENT item in the SYSMAN MD and the LOGON item in SYSTEM.

CAUTION

Any attempt to unlock and operate on the secondary database may lead to loss of synchronisation with the primary.

Configuring the Secondary

The next step is to mark sdbase as the secondary database and link it with the live database (pdbase). This is done using the UNIX utility **fsadm**.

Note: This sample procedure assumes that the databases are located in the directory /usr/dbases.

1. Log in to the system (shost) containing the secondary database.

2. Enter **`fsadm -s -h phost -d /usr/dbases/pdbase sdbase`**. See Chapter 7 for details.

   The **-h** (host) option identifies the remote host as phost.

   The **-s** option marks sdbase as the secondary, inserting an entry in the raw log header

   The **-d** (database) option identifies the remote database as pdbase. The absolute path name must be specified. This must be the same for the primary and secondary. You can omit the **-d** option if the primary (pdbase) and secondary (sdbase) are to have the same name.

   If the primary and secondary database names are the same, for example,

   pdbase = sdbase = dbase

   then you can omit the **-d** option and enter the following:

   ```
   fsadm -s -h phost dbase
   ```

   If $REALDBASE is defined dbase can be omitted as well, that is,

   ```
   fsadm -s -h phost
   ```

After configuring the secondary, config file parameters are displayed, similar to the following.

```
   Failsafe Pair1:
   Database'/usr/dbases/sdbase'
   TCP Host'shost' (Local)
Failsafe Pair2:
   Database'/usr/dbases/pdbase'
   TCP Host'phost' (Remote)
Mode:
   Logging inactive
   Failsafe enabled,secondary,inactive
```

Note: The numbering of the FailSafe Pair variables is not significant. The config file entries on the two hosts may or may not be identical.

Configuring the Primary

Now to complete the FailSafe configuration you must configure the live database pdbase as the primary and link it with the secondary (sdbase). Again, this is done using **fsadm**, as follows:;

Note: This sample procedure assumes that the databases are located in the directory /usr/dbases.

1. Log on to the system containing the primary.

2. Enter **`fsadm -p -h shost -d /usr/dbases/sdbase pdbase`**. See Chapter 7 for details

   The **-h** (host) option identifies the remote host as shost.

   The **-p** option marks pdbase as the primary, inserting an entry in the raw log header

   The **-d** (database) option identifies the remote database as sdbase. The absolute path name must be specified. This must be the same for the primary and secondary. You can omit the **-d** option if the primary (pdbase) and secondary (sdbase) are to have the same name

If the names of the primary and secondary databases are the same, for example,

pdbase = sdbase = dbase

then you can omit the **-d** option with **fsadm** and enter the following:

```
fsadm -p -h shost dbase
```

If $REALDBASE is defined dbase can be omitted as well, that is,

```
fsadm -p -h shost
```

On entering the above **fsadm** command, config file parameters are displayed similar to the following.

```
Failsafe Pair1:
   Database'/usr/dbases/pdbase'
   TCP Host'phost' (Local)
Failsafe Pair2:
   Database'/usr/dbases/sdbase'
   TCP Host'shost' (Remote)
Mode:
   Logging inactive
   Failsafe enabled,primary,inactive
```

Note: The numbering of the FailSafe Pair variables is not significant. The config file entries on the two hosts may or may not be identical.

The two databases pdbase and sdbase are now configured together as a FailSafe pair. Only the primary (pdbase) can be logged to and used as a live database. The secondary (sdbase), is locked at TL-START time.

3. Log on to the primary and create a clean log with a suitable name using TL-CREATE-FILE. See Chapter 3 for naming conventions.

Chapter 5
Operating Procedures

This chapter describes operations and facilities which are used during the routine operation of a FailSafe database. The following procedures are described:

- Initial startup of FailSafe operation
- Switching to a new clean log
- Synchronising primary and secondary databases
- Reversing the roles of a FailSafe pair
- Shutdown procedures
- Archiving clean logs to tape
- Retrieving clean logs from tape
- Monitoring logging

Commands and Utilities Referenced in this Chapter

The following TCL commands are used in the operating procedures described in this chapter.

TCL Commands

- CLEAR-FILE
- ENABLE-LOGONS
- INHIBIT-LOGONS
- TL-CONTINUE
- TL-CREATE-FILE
- TL-DUMP
- TL-LISTFILES
- TL-LOAD
- TL-REDUAL
- TL-START
- TL-STOP
- TL-SWITCH
- TL-STATUS
- TL-TRANSACTIONS

Detailed descriptions of TL-commands are given in Chapter 8. The other TCL commands are described in the RealityX reference manuals

UNIX Tools

The following UNIX tools are used.

- **killreal**. Refer to Chapter 7 for details.
- **cpio**. Refer to the user reference manuals supplied with your system for details

Notes:

1. It is assumed that the FailSafe software is fully installed and configured. If not, see Chapter 4.

2. It is the responsibility of the system administrator to ensure that the primary database is locked so that no other users are able to log on until logging is enabled. Failure to do this may result in loss of synchronisation between primary and secondary databases.

The procedure to start logging is as follows:

1. Check that a FILE-SAVE p tape exists which reflects the current state of the primary and which can be used as a base for restoring future logged updates onto the primary or secondary. If not, make one, or you can use a UNIX back-up utility, such as **cpio**.

2. Run **reality** to enter the primary database and log on to SYSMAN.

3. If necessary, create the clean log(s) on the primary database, with appropriate names which adhere to the naming conventions recommended in Chapter 3. For example:

   ```
   TL-CREATE-FILE CLOGA-MON

   TL-CREATE-FILE CLOGB-MON
   ```

   If the required files already exist, ensure that they are empty. Use CLEAR-FILE, if necessary, to clear them.

4. Now start FailSafe logging using the TL-START command. For example:

```
TL-START CLOGA-MON
```

This starts logging to the primary clean log CLOG-MON, creates a clean log of the same name on the secondary and starts logging to it. It also starts restoring the logged updates to the secondary database.

5. Finally, enter **`ENABLE-LOGONS *`** at TCL to permit user access to the primary database.

FailSafe operation is now active and the primary database is fully operational with the secondary database operating as the standby.

CAUTION

The secondary database is locked to all users, except the super-user and database owner. Unlocking and accessing of the secondary database may result in loss of synchronisation with the primary.

Switching to a New Clean Log

TL-SWITCH, described in Chapter 8, is used to switch from one clean log to another while Transaction Logging is enabled. It can be entered on the primary only, but switches both primary and secondary clean logs at the same time. When and how often you switch the clean log depends on the clean log cycling policy appropriate to your system and database.

Refer to Chapter 3 for detailed advice on establishing a clean log cycling policy, clean log naming conventions, archiving clean logs etc.

> CAUTION
>
> A "clean log partition full" condition can lead to serious system performance problems and potential lock-up conditions. A clean log cycling procedure must be chosen to avoid this happening. See Chapter 3 for advice.

The procedure is as follows:

1. Ensure that appropriate empty clean logs exist on the primary database, as determined by the clean log cycling policy. For example:

   - To switch the clean log once a day with no archiving, you need clean logs such as, CLOG-MON, CLOG-TUES, CLOG-WED etc., one for each day of the working week.

   - To switch the clean log once a day, then archive it to tape, you need clean logs, such as, CLOG-09.03.92, CLOG-10.03.92, CLOG-11.03.92 etc, one for each day and dated appropriately.

   - To switch the clean log more than once a day and archive to tape, you need clean logs, such as, CLOGA09.03.92, CLOGB09.03.92, CLOGA10.03.92, CLOGB10.03.92 etc. or CLOGA-MON, CLOGB-MON, etc., two or more for each day, and dated appropriately.

   If necessary, create the required clean log(s) on the primary using TL-CREATE-FILE or clear them using CLEAR-FILE.

   Associated clean logs on the secondary database are created and cleared automatically by TL-START or TL-SWITCH.

2. Switch to the new clean log using TL-SWITCH. It is recommended that you use TL-SWITCH with the H option to switch logs just before the FILE-SAVE. This switches logs, but suspends the secondary database, allowing the FILE-SAVE to be performed on the secondary while maintaining a fully operational primary. Refer to the section in this chapter, 'Shutting Down the Secondary Temporarily'.

   For example, enter one of the following:

   - **`TL-SWITCH CLOG-TUES (H)`** at the end of Monday's working day.
   - **`TL-SWITCH CLOGB-MON`** during Monday, followed by **`TL-SWITCH CLOGA-TUES (H)`** at the end of Monday's working day. Where archiving is not required, CLOGA-MON should at be kept at least until after the FILE-SAVE.
   - **`TL-SWITCH CLOG-10.03.92 (H)`** at the end of the working day dated 9th March 1992.
   - **`TL-SWITCH CLOGB09.03.92`** during the day, followed by **`TL-SWITCH CLOGA09.03.92 (H)`** at the end of Monday's working day. The full clean log CLOGA-09.03.92 can then be archived. This should be done before the clean log partition becomes full.

     After completing the TL-SWITCH with the H option, you can then execute FILE-SAVE and VERIFY-SAVE on the secondary database, before resuming normal FailSafe operation by entering the TL-CONTINUE command on the primary.

3. If archiving is required, wait until switching is completed before archiving the most recent full clean log. You can check this using the TL-STATUS command. Logging status shold have changed from SWITCH IN PROGRESS to ACTIVE.

   To archive the clean log(s), copy to tape using either TL-DUMP at TCL or a UNIX utility, such as **cpio**. Refer to the section 'Archiving Clean Logs' for more details.

4. As necessary, clear or delete the old log(s) from disk to release space in the clean log partition. Use CLEAR-FILE if you wish to clear old clean logs and re-use them. Use DELETE-FILE if you have archived them and wish to remove the log names from the system.

   If you clear a primary log and re-use it, the secondary log is cleared automatically at TL-START or TL-SWITCH. However, if you delete a primary log, then the secondary log remains full and it is necessary to delete it, to release clean log partition space on the secondary's system.

Notes:

1. This procedure is particularly useful when the roles of primary and secondary have been reversed after database recovery and where the new roles do not provide for optimum operating efficiency.

2. The changeover is only made when all recent primary updates have been applied to the secondary and the two databases are synchronised.

3. Any active transactions will be rolled back.

4. It is not necessary to stop logging.

To reverse the roles of the primary and secondary databases in a FailSafe pair, proceed as follows:

1. Log in to the machine containing the primary database, and log on to the primary database For the sake of this procedure lets call it 'dbase'.

2. On dbase, enter

   ```
   INHIBIT-LOGONS *
   ```

   then send a message to all users to log off.

3. Wait a reasonable period to allow users time to log off, then LOG OFF all remaining users.

4. Now log off dbase and return to the UNIX shell by entering

   ```
   OFF
   ```

5. At the UNIX shell, enter

   ```
   fsadm -T dbase
   ```

   This converts the dbase to be a secondary and locks it. If dbase is defined in the environment variable £REALDBASE, then you can omit the database name from **fsadm**.

6. If necessary log in to the system containing the secondary. Lets assume the secondary database is also called dbase. Ensure no one is logged on to the secondary database.

7. At the UNIX shell, enter

   `fsadm -T dbase`

   This converts the secondary dbase to be a primary and unlocks it for users. If dbase is defined in the environment variable £REALDBASE, then you can omit the database name from **fsadm**.

   Users can now log on to the new primary database and continue working.

Shut-down Procedures

A number of shut-down options are supported in a Reality X FailSafe configuration. They are:

- To shut down both primary and secondary databases together so that they remain synchronised.
- To suspend the secondary only, maintaining synchronisation.
- To shut down the secondary only permanently, leaving the primary as a stand-alone database.
- To shut down all databases on a complete system.

Shutting Down a FailSafe Pair

To shut-down FailSafe on your database carry out the following steps.

1. Enter the primary database and log on to SYSMAN.
2. Enter **`INHIBIT-LOGONS *`** to prevent further user access.
3. Send a message to ask all users to log off the database. Use LISTU to check that all users have logged off.
4. After a reasonable period of time, log off all remaining users from the database.
5. Stop logging using the TL-STOP command.

   You can check that logging has completed using the TL-STATUS command. This should show the logging status as INACTIVE. The FailSafe database is now in the logging disabled state. It can be restarted using TL-START as described in the initial start up procedure.
6. Maintain the clean log on disk according to your back-up policy.

The current clean log, together with any other clean logs filled since the last back-up, can be used together with the last back-up tape to recover the most recent and consistent database. However, remember that, if the system crashes, all updates made to the database after shut-down of logging and before the next TL-START are not recoverable.

It is the system administrator's responsibility to ensure that users cannot log on to the primary while FailSafe logging is disabled. Failure to do this may result in loss of synchronisation between primary and secondary databases.

Note also that the execution of a TL-STOP followed by a TL-START will terminates one chain of clean logs and start a new one. Therefore in the event of a failure, two restore operations will be required in order to complete the recovery of all clean logs. Recovery procedures are described in Chapter 6.

Suspending the Secondary Database

TL-SWITCH with the H option suspends update operations on the secondary database, until TL-CONTINUE is executed, while maintaining FailSafe logging to the secondary's clean log. Primary updates continue to be logged to the secondary clean log. This allows you to back-up the secondary without shutting the secondary down completely and losing synchronisation. The procedure is as follows:

1. Create a empty clean log on the primary, for example, CLOG-FRI.

2. On the primary database, enter **`TL-SWITCH CLOG-FRI (H`**.

   This creates a CLOG-FRI clean log on the secondary. Logging is switched to CLOG-FRI on the primary and secondary, and the restore process which applies the updates to the secondary database is disabled.

3. Use TL-STATUS on the secondary to ascertain when the restore process has completed, indicated by a ACTIVE-SECONDARY PAUSED status.

4. You can now save the secondary database onto a back-up tape. Primary users may continue operating on the database unaffected.

Note: A TL-STOP and TL-SWITCH cannot be performed until the databases have been returned to FailSafe synchronised mode using the TL-CONTINUE verb.

5. Re-synchronise the secondary with the primary by entering **`TL-CONTINUE`** on the primary.

   TL-CONTINUE re-enables the updating of the secondary database. Firstly, all outstanding updates, logged in the secondary clean log, are restored on the secondary database, after which, all current primary updates are applied to it.

   This process updates the secondary database with all outstanding updates from the secondary log, made since the secondary was suspended. This continues until the databases are synchronised and normal FailSafe operation is resumed.

Shutting Down the Secondary Database Only

TL-SWITCH with the K option is used to shut down the secondary database while maintaining the primary as a stand-alone database. Primary users are unaffected by the secondary shut-down. The procedure is as follows:

1. Create a empty clean log on the primary, for example, CLOG-FRI.

2. On the primary database, enter

   ```
   TL-SWITCH CLOG-FRI (K
   ```

   This switches logging to the new primary log CLOG-FRI and disconnects the FailSafe link to the secondary causing the secondary to become idle. The databases are marked as 'Failsafe failed'. Synchronisation between the primary and secondary databases is therefore lost and you have to execute a TL-REDUAL in order to restart FailSafe operation.

Shutting Down a System

The **killreal** command (without options) shuts down the RealityX central daemon and all database daemons on a system. killreal with the **-d** option shuts down a specified database. It is recommended that you shut-down Transaction Logging on each database before shutting down the complete system. The procedure is as follows.

1. Log on to each primary database on the system to be shut down and enter **`TL-STOP`**.

2. If there are any secondary databases on the system, log on to their associated primaries and in each case enter TL-STOP.

3. Wait until the last TL-STOP is completed (This will just under 5 minutes.), then enter **`killreal`** on the system to be shut down. This will terminate the Reality X central daemon and all associated processes.

Note: Failure to execute a TL-STOP before executing **killreal** may result in loss of synchronisation between the associated databases.

Archiving Clean Logs

To archive clean logs to tape the following commands are available:

- TL-DUMP. This is entered at TCL and archives clean logs one at a time onto tape.
- **cpio**. This is entered at the UNIX shell, either as part of the standard daily back-up of the whole database or to copy multiple clean logs onto a separate clean log tape.

Using TL-DUMP

For example, to archive the clean logs CLOGA21.03.91, CLOGB21.03.91 and CLOGC21.03.91 using TL-DUMP, proceed as follows:

1. Enter the database and log on to SYSMAN.
2. Load and attach the first tape, then enter the following at TCL:

   ```
   TL-DUMP CLOGA21.03.91
   ```

3. Load and attach another tape, then, enter the following at TCL:

   ```
   TL-DUMP CLOGB21.03.91
   ```

4. Load and attach another tape, then enter the following at TCL:

   ```
   TL-DUMP CLOGC21.03.91
   ```

Note: Each clean log is copied onto a separately attached tape. It is necessary to archive one clean log per tape, as TL-LOAD cannot read multiple logs from a single tape.

Using cpio

For example, using the UNIX copying utility **cpio** you can archive clean logs CLOGA21.03.91, CLOGB21.03.91 and CLOGC21.03.91, as follows:

1. Change to the clean log sub-directory. For example, enter

   ```
   cd /clean-logs/dbase1-clogs
   ```

   where 'clean-logs' is the main clean log directory and 'dbase1-clogs', the sub-directory for the database 'dbase1'

2. Use the **-o** (output) option of **cpio** to archive the day's clean logs to tape. For example, type

```
ls CLOG?21.03.91|cpio -ocvB > /dev/rmt/1bm
```

Using this statement, the clean logs, filled during the 21 March 1991, are listed and piped to the standard input of the **cpio** utility which copies the listed logs to the tape device /dev/rmt/1bm and archives them with relative path names.

Having archived the clean logs, they may be cleared and, if appropriate, deleted from the database using DELETE-FILE to release clean log partition space. Typically they would be deleted if their file names were date-specific.

Note: You should use DELETE-FILE and not the UNIX command **rm**, as rm will not delete the D pointers.

Retrieving Clean Logs

To retrieve clean logs from tape the following commands are available.

- TL-LOAD. This is entered at TCL within the Reality X environment and loads clean logs one at a time from tape.
- **cpio**. This is entered at the UNIX shell, either as part of a selective restore of clean logs from a standard daily back-up tape or to copy multiple clean logs from a separate clean log tape.

Using TL-LOAD

For example, to load clean logs back into your database from tape, you can use TL-LOAD, as follows:

1. Enter the database and log on to SYSMAN.
2. Load and attach the first tape containing CLOGA21.03.91, then enter the following:

   ```
   TL-LOAD CLOGA21.03.91
   ```

3. Load and attach the tape containing CLOGB21.03.91, then enter the following at TCL

   ```
   TL-LOAD CLOGB21.03.91
   ```

4. Load and attach the tape containing CLOGC21.03.91, then enter the following:

   ```
   TL-LOAD CLOGC21.03.91
   ```

Each command takes the specified clean log and loads it back into the clean log sub-directory for your database. This operation will fail if the file names already exist in the sub-directory.

Using cpio

For example, to selectively restore the files archived on 21 March 1991, proceed as follows:

1. Change to the clean log sub-directory in which you want to restore them. For example, enter

   ```
   cd /clean-logs/dbase1-clogs
   ```

   where 'clean-logs' is the main clean log directory and 'dbase1-clogs', the sub-directory for the database 'dbase1'

2. Use the **-i** (input) option of **cpio** to retrieve all 21 March '91 clean logs to tape. For example, type

```
cpio  -icvB "*21.03.91"</dev/rmt/1bm
```

This statement copies all files with path names ending in 21.03.91 from tape device /dev/rmt/1bm into the clean log sub-directory in which you are currently working.

Finally log on to the database and recreate the clean log D-pointers, previously deleted from the database, using TL-CREATE-FILE with the E option. For example:

```
TL-CREATE-FILE CLOGA21.03.91 (E)
```

Facilities to Monitor Transaction Logging

Three TCL commands are available to monitor Transaction Logging:

- TL-LISTFILES to list information about the log files on the database.
- TL-STATUS to monitor the current status of logging on the database.
- TL-TRANSACTIONS to display information about transactions currently active on the database.

For a description of these commands refer to Chapter 8.

Chapter 6
Recovery Procedures

This chapter introduces the data recovery methods supported by Transaction Logging in a FailSafe configuration. It then describes the first steps in the procedure to recover a database and four optional procedures to complete the restoration of clean logs and the resynchronisation of FailSafe operation. These are:

1. A procedure using TL-REDUAL to restore a chain of clean logs in one sequence.

2. A procedure using TL-REDUAL to restore clean logs one at a time.

3. A procedure using TL-RESTORE to restore chained clean logs in one sequence, followed by the use of TL-REDUAL to resynchronise databases.

4. A procedure using TL-RESTORE to restore clean logs one at a time, followed by the use of TL-REDUAL to resynchronise databases.

Finally it describes the facilities available for copying clean logs from one database to another.

Commands and Utilities Referenced in this Chapter

TCL Commands

The following TCL commands are used in the recovery procedures described in this chapter.

- TL-RESTORE
- TL-REDUAL
- TL-DUMP
- TL-LOAD
- SET-FILE

> **CAUTION**
>
> TL-RESTORE and TL-REDUAL require all clean logs to have been TL-SWITCH'ed in order to restore clean logs in one chained sequence. If a TL-STOP/TL-START operation has been carried out, then the linkage between logs will be broken, in which case the restore will terminate at the clean log which was active when the TL-STOP occurred. It is essential that TL-SWITCH is used to change logs to maintain linkage between clean logs.

UNIX Tools

Also, the following UNIX utilities are used.

- **ftp**
- **cpio**
- **fsadm**

Introducing Recovery Methods

Full Recovery

Full Recovery means restoring a database from all logged updates. This is carried out by first restoring the most recent back-up of the database, then restoring all clean logs since that back-up was taken, onto the now partially-restored database. Facilities are also supported to re-establish the FailSafe link and resynchronise the databases.

There are a number of ways in which you can to recover and resynchronise a FailSafe database. These use TL-RESTORE and/or TL-REDUAL. Which one you choose will depend partly on system limitations and partly on personal preference. Four ways are summarised below. Guidance on which way to choose is given in a flowchart in Figure 6-1. Each method is described step by step in subsequent sections of this chapter.

1. Execute a TL-REDUAL after copying all necessary clean logs to the corrupted database. TL-REDUAL then initiates one automatic restore sequence which restores all clean logs onto the corrupted database in a verified chronological order, resynchronises the databases and resumes FailSafe operation. This is probably the most efficient method, but in order to do this it is necessary for the clean log disk partition to be large enough to hold all the necessary clean logs and the machine-to-machine link to be up and reliable.

2. Execute a TL-REDUAL, copying clean logs to the partially-restored database one at a time. The restore process, initiated by TL-REDUAL, restores clean logs in chronological order, if present on the database. If the next clean log in chronological order is not present, the restore process prompts for it. Once loaded onto the database the chained restore continues. If the next clean log is detected as being missing then the prompt sequence is repeated. This interactive method is useful if the disk partition is not large enough to hold all the clean logs. It allows you to load and delete clean logs one at a time, but still verifies the order in which they are restored. FailSafe operation is re-established in parallel with the restoring of clean logs and on completing the restore, the primary and secondary databases are re-synchronised,

3. Execute TL-RESTORE with the AE option and with all clean logs to be restored present on the database, then resynchronise using TL-REDUAL. TL-RESTORE with the AE option restores all clean logs in a verified chronological order. Before executing TL-RESTORE, all necessary logs need to be loaded onto the database.

4. Execute TL-RESTORE with the AE option, but only load one clean log at a time onto the database. This method is useful when disk space is limited. Having loaded and restored a clean log you then delete or clear it to release space in the clean log partition. If a clean log is not present, TL-RESTORE prompts and wait for it to be copied to the database. This method is particularly useful if you do not want to resume FailSafe operation, but you want to commence restoring the database, for example, if the machine-to-machine link is down.

Selective Recovery

Selective Recovery is a procedure in which only selected items from a clean log are restored.

Selection of the update items to be restored from the clean log is made using the ENGLISH SELECT verb, then TL-RESTORE is applied to the SELECTed list.

Chapter 9 gives details on the use of ENGLISH to manipulate clean log items. Refer to the *ENGLISH Reference Manual* for details on the use of the SELECT verb.

First Steps to Recovery

This section outlines first steps to recovering a FailSafe database after a system failure causes a primary or secondary database to be corrupted.

Procedure after a Primary Failure

If the primary database fails, proceed as follows:

1. On the system containing the corrupted primary, enter

   ```
   fsadm -R pdbase
   ```

   where pdbase is the corrupted primary. This re-configures the database as a secondary and locks it.

2. Similarly, on the system containing the associated secondary database, enter

   ```
   fsadm -R sdbase
   ```

   where sdbase is the secondary database name. This re-configures the database as a promary and unlocks it.

3. Now inform all users to access the new stand-alone primary (previously the secondary).

4. Repair and re-boot the failed system

5. Restore the most recent FILE-SAVE or UNIX back-up onto the corrupted database. Once the back-up is restored, the next stage is to restore clean logs and resynchronise FailSafe operation. See below the sections on Recovering Clean Logs.

Procedure after a Secondary Failure

When a secondary database system fails, primary users are unaffected and the primary database continues to operate normally as the live database. They are also unaware that the secondary has failed unless the FailSafe failed flag is set. If the secondary fails while the primary is IDLE (no one is logged on), the FailSafe failed flag will not be set, so that when the first primary user attempts to log on, the logon will fail. Hence the followin procedure ensures that the FailSafe failed flag for the primary is set.

1. Enter

   ```
   fsadm -f pdbase
   ```

   on the system containing the primary. This sets the FailSafe failed flag for pdbase.

2. Repair and re-boot the failed system

3. Restore the most recent FILE-SAVE or UNIX back-up onto the corrupted database. Once the back-up is restored, the next stage is to restore clean logs and resynchronise FailSafe operation. See below the sections on Recovering Clean Logs.

Recovering SWITCH'ed Clean Logs

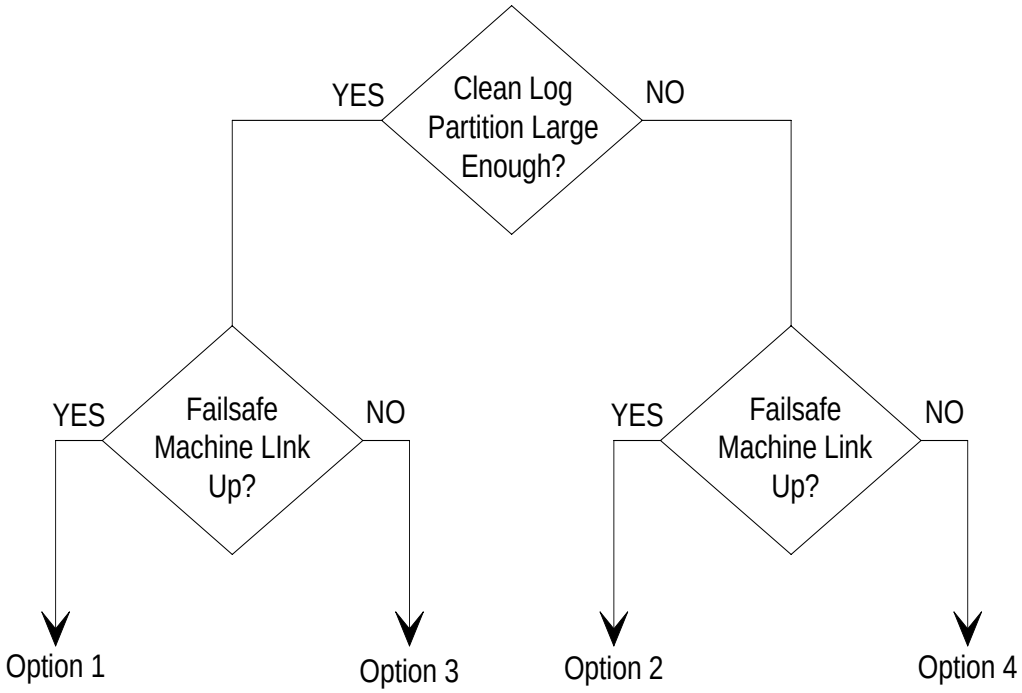
If all the clean logs to be restored have been TL-SWITCH'ed so that they are linked in a single chronological chain, then you can use one of four options to restore them depending on system limitations as follows:

Option 1	If the clean log disk space is large enough to hold all clean logs made since the last back-up [redacted] the machine-to-machine link is up and reliable, carry out Option 1 - Using TL-REDUAL to Restore a Chain of Clean Logs in One Sequence.
Option 2	If the clean log partition space is too small to hold all clean logs, but the machine-to-machine link is up and you wish to restore and resynchronise FailSafe operation, carry out Option 2- Using TL-REDUAL to Restore Clean Logs One at a Time.
Option 3	If the clean log disk space is large enough to hold all clean logs made since the last back-up, [redacted] the machine-to machine-link is down, or for some other reason you wish to commence the restore, but [redacted] re-establish FailSafe operation, carry out Option 3 - Using TL-RESTORE to Restore Chained Clean Logs in One Sequence, then TL-REDUAL.
Option 4	If the clean log disk space is too small to hold all clean logs made since the last back-up, and the machine-to machine-link is down, or for some other reason you wish to start to restore, but not re-establish FailSafe operation, carry out Option 4 - Using TL-RESTORE to Restore Clean Logs One at a Time, then TL-REDUAL.

A flowchart to help in this decision process is shown in Figure 6-1 below.

Recovering TL-STOP/ TL-START'ed Log Sequences

If logging has been stopped and started again when changing to a new clean log, this breaks the link between clean logs. Restore of clean logs cannot be executed as one complete chain. It is therefore necessary to use TL-RESTORE with the AE options, as in Options 3 and 4, for each sub-chain of clean logs created by a TL-START/STOP sequence except for the last sub-chain of logs linked to the active log when you use TL-REDUAL.



Notes:

1. Options 1 to 4 are detailed later in this chapter.

2. Options 1 to 4 are described with reference to a typical sequence of SWITCH'ed clean logs logged on a FailSafe database since the last back-up. This sequence is illustrated in Figure 6-2.

Figure 6-1 Flowchart to Choose Clean Log Restore Procedure

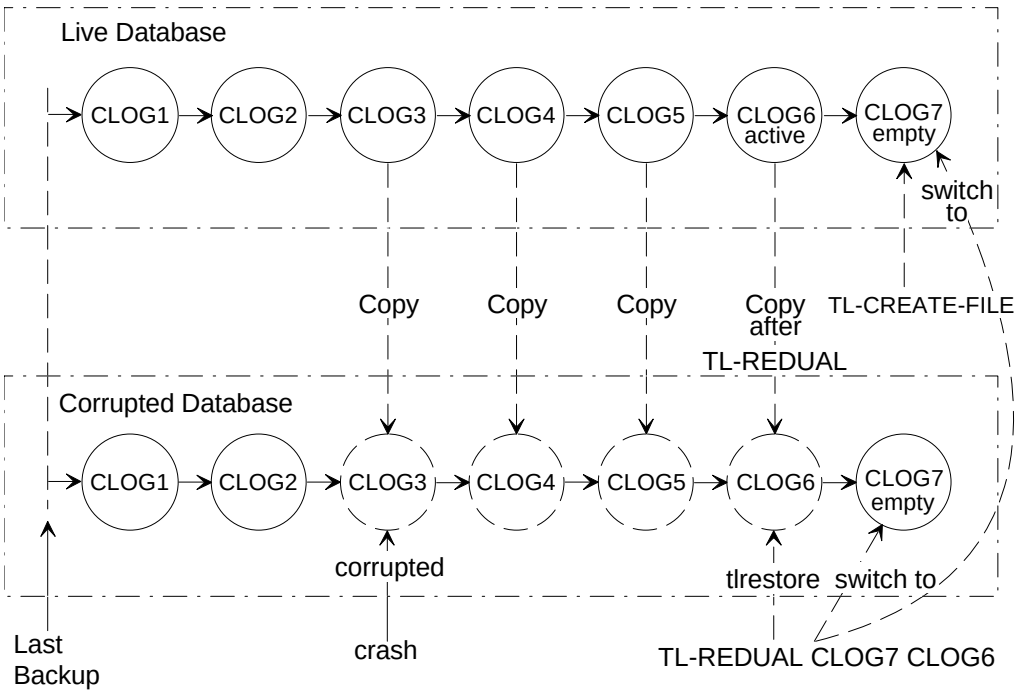


Figure 6-2 Example of Clean Log Restore Sequence

Notes:

1. The chain of clean logs illustrated below assumes that changing from one clean log to the next has been achieved by a TL-SWITCH or TL-REDUAL.

2. The clean log restore and resynchronisation procedures described next in this chapter use the scenario illustrated in this diagram as a basis for the descriptions.

Option 1 - Using TL-REDUAL to Restore a Chain of Clean Logs in One Sequence

The procedure is as follows:

1. Delete, from the failed system, the clean log (CLOG3) active at the time that the system crashed.

Note: This is necessary as CLOG3 on the corrupted database may be out of synchronisation with the corresponding clean log (CLOG3) on the now live database. Updates to the live database may have been added to the CLOG3 while the failed system was down.

To do this, enter

```
TL-CREATE-FILE CLOG3 (E)
```

then

```
DELETE-FILE CLOG3
```

This recreates the D-pointers, lost when the database was restored from the last back-up tape, then deletes the clean log.

Alternatively, enter

```
rm /clean/dbase/CLOG3

rm /clean/dbase/CLOG3v
```

to remove both visible and binary files.

2. Copy all necessary clean logs, CLOG1 to CLOG5, to the failed system. Facilities to copy clean logs between databases or from tape are described at the end of this chapter. The necessary clean logs may be

   - already on the failed system, in which case, recreate their D-pointers using TL-CREATE-FILE with the E option to make them available on the partially-restored database.
   - on the live database, in which case copy them over to the failed system.
   - archived, in which case retrieve them from tape and load onto the failed system.

If you use a UNIX utility to copy a clean log across you will need to use the TL-CREATE-FILE verb with the E option to create a D-pointer for the clean log, before it can be used on the database. Refer to the section on Copying Clean Logs between Databases at the end of this chapter.

Note: CLOG6 cannot be copied over yet as it is still the active log.

3. Create an empty clean log (CLOG7) on the live database using TL-CREATE-FILE.

4. Execute TL-REDUAL on the live database. For example, enter

```
TL-REDUAL CLOG7 CLOG1
```

This switches logging to CLOG7 on both databases. If an empty clean log (CLOG7) does not exist on the secondary, TL-REDUAL creates one.

TL-REDUAL also re-establishes the FailSafe link. Updates on the live database (the primary) are once more logged to the secondary clean log, but are not yet applied to the secondary (partially-restored database). Instead, the secondary is restored from the clean logs starting with CLOG1 and carrying on in sequence through to CLOG7 (the active log), assuming all appropriate clean logs were TL-SWITCH'ed during logging run time.

TL-REDUAL informs you that CLOG6 does not exist by displaying a prompt at the system console similar to the following:

```
Jul 09 16:32:03 #7309 tlrestore WARNING Log/cleanlog failsafe/LOG6 empty
        Please load new log file
```

The message is repeated every 5 minutes. You have to wait approximately 5 minutes to allow switching of clean logs to be completed before you can load CLOG6 onto the secondary database.

5. Execute TL-STATUS with the L option on the primary to monitor the state of switching. You must wait until the Status field on the TL-STATUS screen changes from SWITCH IN PROGRESS to ACTIVE before you copy CLOG6 across. This should take just under 5 minutes.

6. Once the Status on the TL-STATUS screen has changed, copy CLOG6 across from the primary. With CLOG6 copied across the restore process continues on through to the current active clean log (CLOG7), until the backlog of updates in CLOG7 are restored and recovery is complete. The recovered secondary database is now synchronised with the live primary database and normal FailSafe operation is re-established.

Option 2- Using TL-REDUAL to Restore Clean Logs One at a Time

If clean log partition space is a problem then you can still use TL-REDUAL, but copy clean logs onto the failed system one at a time and restore them singly. The procedure is as follows:

1. Delete, from the failed system, the clean log (CLOG3) active at the time that the system crashed.

Note: This is necessary as CLOG3 on the corrupted database may be out of synchronisation with the corresponding clean log (CLOG3) on the now live database. Updates to the live database may have been added to the CLOG3 while the failed system was down.

To do this, enter

```
TL-CREATE-FILE CLOG3 (E)
```

then

```
DELETE-FILE CLOG3
```

This recreates the D-pointers, lost when the database was restored from the last back-up tape, then deletes the clean log.

Alternatively, enter

```
rm /clean/dbase/CLOG3
```

```
rm /clean/dbase/CLOG3v
```

to remove both visible and binary files.

2. Copy the first clean log CLOG1 to the failed system. Facilities to copy clean logs between databases or from tape are described at the end of this chapter. The required clean log may be

   - already on the failed system, in which case, recreate their D-pointers using TL-CREATE-FILE with the E option to make them available on the partially-restored database.
   - on the live database, in which case, copy it over to the failed system.
   - archived, in which case, retrieve it from tape and load onto the failed system.

If you use a UNIX utility to copy a clean log across you will need to use the TL-CREATE-FILE verb with the E option to create a D-pointer for the clean log, before it can be used on the database. Refer to the section on Copying Clean Logs between Databases at the end of this chapter.

3. Create an empty clean log (CLOG7) on the live database using TL-CREATE-FILE.

4. Execute TL-REDUAL on the live database. For example, enter

   TL-REDUAL CLOG7 CLOG1

   This switches logging to CLOG7 on both databases. If an empty clean log (CLOG7) does not exist on the secondary, TL-REDUAL creates one.

   TL-REDUAL also re-establishes the FailSafe link. Updates on the live database (the primary) are once more logged to the secondary clean log, but are not yet applied to the secondary (partially-restored database). Instead, the secondary is restored from the clean logs starting with CLOG1 and carrying on to CLOG7 (the active log).

   After restoring CLOG1, TL-REDUAL looks for CLOG2 and if it does not find it, it displays a prompt at the system console similar to the following:

   ```
   Jul 09 16:32:03 #7309 btlrestore WARNING Log/cleanlog failsafe/LOG2 empty
           Please load new log file
   ```

   The message is repeated every 5 minutes.

5. Now delete the previously restored clean log from the failed system to recover clean log partition space. You can use DELETE-FILE

6. Copy the requested log (CLOG2) on the failed system. TL-RESTORE will then continue restoring CLOG2.

7. Repeat steps 5. and 6. for CLOG2 through to CLOG5, deleting each clean log after the restore is complete and copying across the next consecutive clean log, as requested by the message prompt.

   You have to wait approximately 5 minutes to allow switching of clean logs to be completed. before you can load CLOG6 onto the secondary database.

8. Execute TL-STATUS with the L option on the primary to monitor the state of switching. You must wait until the Status field on the TL-STATUS screen changes from SWITCH

IN PROGRESS to ACTIVE before you copy CLOG6 across. This should take just under 5 minutes.

9. Once the Status on the TL-STATUS screen has changed, copy CLOG6 across from the primary.

   With CLOG6 copied across the restore process continues on through to the current active clean log (CLOG7), until the backlog of updates in CLOG7 are restored and recovery is complete. The recovered secondary database is now synchronised with the live primary database and normal FailSafe operation is re-established.

Option 3 - Using TL-RESTORE to Restore Chained Clean Logs in One Sequence, then TL-REDUAL

To restore all clean logs in one chain, proceed as follows:

1. Delete, from the failed system, the clean log (CLOG3) active at the time that the system crashed.

Note: This is necessary as CLOG3 on the corrupted database may be out of synchronisation with the corresponding clean log (CLOG3) on the now live database. Updates to the live database may have been added to the CLOG3 while the failed system was down.

To do this, enter

```
TL-CREATE-FILE CLOG3 (E)
```

then

```
DELETE-FILE CLOG3
```

This recreates the D-pointers, lost when the database was restored from the last back-up tape, then deletes the clean log.

Alternatively, enter

```
rm /clean/dbase/CLOG3
```

```
rm /clean/dbase/CLOG3v
```

to remove both visible and binary files, where /clean/dbase is the clean log sub-directory path-name.

2. Ensure that CLOG1 to CLOG5 are present on the failed system. These may be

   - already on the failed system, in which case, recreate their D-pointers using TL-CREATE-FILE with the E option.
   - on the live database, in which case, copy them over to the failed system.
   - archived, in which case, retrieve them from tape and load onto the failed system.

   If you use a UNIX utility to copy a clean log across you will need to use the TL-CREATE-FILE verb with the E option to create a D-pointer for the clean log, before it

can be used on the database. Refer to the section on Copying Clean Logs between Databases at the end of this chapter.

3. Execute TL-RESTORE with the AE option on the partially-restored database. For example,

```
TL-RESTORE CLOG1 (AE
```

The restore commences at CLOG1, and continues with CLOG2 through to CLOG5 in chronological order until all inactive clean logs are restored. The restore process will then prompt for CLOG6, as follows:

```
Log CLOG6 empty.    Please load new log file.
  Hit A to Abort or C to continue.
```

4. Enter A to abort the restore.

5. Create an empty clean log (CLOG7) on the live database using TL-CREATE-FILE.

6. Ensure that the network connection is up, then execute TL-REDUAL on the live database (now the primary). For example, enter

```
TL-REDUAL CLOG7 CLOG6
```

This switches logging to CLOG7 on both databases. If an empty clean log (CLOG7) does not exist on the secondary, TL-REDUAL creates one.

TL-REDUAL also re-establishes the FailSafe link. Updates on the live database (the primary) are once more logged to the secondary clean log, but are not yet applied to the secondary (partially-restored database). Instead, the secondary is restored from the clean logs starting with CLOG6 and carrying on to CLOG7 (the active log).

TL-REDUAL informs you that CLOG6 does not exist by displaying a prompt at the system console similar to the following:

```
Jul 09 16:32:03 #7308tlrestore WARNING Log/cleanlog failsafe/LOG6 empty
        Please load new log file
```

The message is repeated every 5 minutes. You have to wait approximately 5 minutes to allow switching of clean logs to be completed. before you can load CLOG6 onto the secondary database.

7. Execute TL-STATUS with the L option on the primary to monitor the state of switching. You must wait until the Status field on the TL-STATUS screen changes from SWITCH IN PROGRESS to ACTIVE before you copy CLOG6 across. This should take just under 5 minutes.

8. Once the Status on the TL-STATUS screen has changed, copy CLOG6 across from the primary.

   With CLOG6 copied across the restore process continues on through to the current active clean log (CLOG7), until the backlog of updates in CLOG7 are restored and recovery is complete. The recovered secondary database is now synchronised with the live primary database and normal FailSafe operation is re-established.

Option 4 - Using TL-RESTORE to Restore Clean Logs One at a Time, then TL-REDUAL

In this procedure clean logs are copied onto the failed system one at a time because of restrictions in clean log partition space and restored separately.

This procedure uses TL-RESTORE with the AE option which restores a chain of clean logs in chronological order and prompts for the correct log in the chain if it is not present on the database.

> **CAUTION**
>
> It is important that you use TL-RESTORE with the AE options. Using TL-RESTORE without the AE options does not verify the order in which logs are restored.

The procedure is as follows:

Note: This procedure requires you to copy clean logs between databases or from tape. Facilities to do this are described at the end of this chapter.

1. Delete, from the failed system, the clean log (CLOG3) active at the time that the system crashed.

Note: This is necessary as CLOG3 on the corrupted database may be out of synchronisation with the corresponding clean log (CLOG3) on the now live database. Updates to the live database may have been added to the CLOG3 while the failed system was down.

To do this, enter

```
TL-CREATE-FILE CLOG3 (E)
```

then

```
DELETE-FILE CLOG3
```

This recreates the D-pointers, lost when the database was restored from the last back-up tape, then deletes the clean log.

Alternatively, enter

```
rm /clean/dbase/CLOG3
```

```
rm /clean/dbase/CLOG3v
```

to remove both visible and binary files.

2. Copy the first clean log CLOG1 to the failed system. Facilities to copy clean logs between databases or from tape are described at the end of this chapter.

   The required clean log may be

   - already on the failed system, in which case, recreate their D-pointers using TL-CREATE-FILE with the E option.
   - on the live database, in which case, copy them over to the failed system.
   - archived, in which case, retrieve them from tape and load onto the failed system.

   If you use a UNIX utility to copy a clean log across you will need to use the TL-CREATE-FILE verb with the E option to create a D-pointer for the clean log, before it can be used on the database.

3. Enter

   ```
   TL-RESTORE CLOG1 (AE
   ```

   After restoring CLOG1, TL-RESTORE looks for CLOG2 and if it does not find it, it displays the following prompt at the system console and waits:

   ```
   Log CLOG2 empty.    Please load new log file.
     Hit A to Abort or C to continue.
   ```

4. Now delete the previously restored clean log from the failed system to recover clean log partition space. You can use DELETE-FILE

5. Copy the requested log (CLOG2) on the failed system and type C to continue. TL-RESTORE will then continue restoring CLOG2.

6. Repeat steps 4. and 5. for CLOG2 through to CLOG5, deleting each clean log after the restore is complete and copying across the next consecutive clean log, as requested by the message prompt.

7. When CLOG6 is prompted for type **A** to abort the restore.

8. Create an empty clean log (CLOG7) on the live database using TL-CREATE-FILE.

9. Execute TL-REDUAL on the live database. For example, enter

```
TL-REDUAL CLOG7 CLOG6
```

This switches logging to CLOG7 on both databases. If an empty clean log (CLOG7) does not exist on the secondary, TL-REDUAL creates one.

TL-REDUAL also re-establishes the FailSafe link. Updates on the live database (the primary) are once more logged to the secondary clean log, but are not yet applied to the secondary (partially-restored database). Instead, the secondary is restored from the clean logs starting with CLOG6 and carrying on to CLOG7 (the active log).

TL-REDUAL informs you that CLOG6 does not exist by displaying prompt at the system console similar to the following:

```
Jul 09 16:32:03 #7308 tlrestore WARNING Log/cleanlog failsafe/CLOG6 empty
        Please load new log file
```

You have to wait approximately 5 minutes to allow switching of clean logs to be completed. before you can load CLOG6 onto the secondary database.

10. Execute TL-STATUS with the L option on the primary to monitor the state of switching. You must wait until the Status field on the TL-STATUS screen changes from SWITCH IN PROGRESS to ACTIVE before you copy CLOG6 across. This should take just under 5 minutes.

11. Once the Status on the TL-STATUS screen has changed, copy CLOG6 across from the primary.

With CLOG6 copied across the restore process continues on through to the current active clean log (CLOG7), until the backlog of updates in CLOG7 are restored and recovery is complete. The recovered secondary database is now synchronised with the live primary database and normal FailSafe operation is re-established.

Copying Clean Logs between Databases

Clean logs need to be copied from one database to another, as part of the recovery procedure. This section discusses the methods available. Two main media for transferring copies are available:

- Magnetic tape
- Communications network

Copying Clean Logs via Tape

Dumping a clean log onto tape and reloading it onto another database can be performed in either the Reality X database or in UNIX. Reality X supports the TCL verbs TL-DUMP and TL-LOAD, and UNIX supports the **cpio** command.

TL-DUMP and TL-LOAD

TL-DUMP is used to copy a clean log to tape and TL-LOAD to reload it from tape onto a database. Both commands are detailed in Chapter 8. Archiving and retrieval procedures, which are very similar, are described and illustrated in Chapter 5.

cpio Utility

cpio is a UNIX utility that enables you to copy a set of files to tape and recover them individually. The command is detailed in the UNIX user manuals supplied with your system Archiving and retrieval procedures using **cpio** are described and illustrated in Chapter 5.

Copying Clean Logs via a Network

Clean logs can be copied across a network using remote Q pointers or a UNIX file transfer facility, such as **ftp** (ARPANET file transfer program). These options are discussed below.

Using Remote Q Pointers

The following procedure is an example of how you can use a remote Q pointer to copy a clean log between databases.

1. Create a remote Q pointer to a clean log on the remote database using SET-FILE.

Note: A clean log must already exist on the remote database. If not, it must be created before using SET-FILE.

For example,

```
SET-FILE failsafe-a.SYSMAN CLOG1
```

The screen displays

```
QFILE and QF*port updated
```

2. If necessary, clear the binary data section of the remote clean log via the Q pointer. For example,

```
CLEAR-FILE QFILE,BINARY
```

3. Copy all clean log items from CLOG1 on the live database to the empty clean log on the remote database. For example,

```
COPY CLOG1,BINARY *
TO:(QFILE,BINARY
```

Using ftp

Clean logs can also be copied across a network from UNIX using **ftp**. The following is an example of a procedure for transferring from a remote active system to a local corrupted system.

1. Change to the appropriate clean log sub-directory on the local database. For example,

```
cd /clean-logs/dbase1
```

2. Enter **ftp** at the shell prompt. The ftp prompt is now displayed.

```
ftp>
```

3. Open a connection to the remote system. For example, enter

```
open 192.67.50.36
```

where 192.67.50.36 is the network address of the remote system.

The system responds with

```
Connected to 192.67.50.36
Host1 FTP server (Version ### ready)
Name:
```

4. Enter the UNIX user id for the remote system, for example,

   ```
   Name:realman
   ```

   The system responds with

   ```
   password required for realman
   Password:
   ```

5. Enter the password. The system responds with

   ```
   User realman logged in
   ```

6. Change to the appropriate clean log sub-directory. For example, enter

   ```
   cd /clean-logs/dbase1
   ```

   The system responds with

   ```
   CWD command successful
   ```

7. Set the file transfer type to support binary images. Enter

   ```
   type binary
   ```

   The system responds with

   ```
   Type set to I
   ```

8. Turn off interactive prompting. Enter

   ```
   prompt
   ```

   The system responds with

   ```
   Interactive mode off
   ```

   Multiple files are now transferred by 'mget' in one sequence without user intervention.

9. Transfer all CLOG files from the clean log sub-directory for the remote database. For example, enter

   ```
   mget CLOG?
   ```

The system responds with

```
Using binary mode to transfer files
```

followed by a sequence of messages similar to the following.

```
Opening data connection for CLOG1
Transfer complete
Local: CLOG1  Remote: CLOG1
6293696 bytes received in 34.4 seconds

Opening data connection for CLOG2
Transfer complete
Local: CLOG2  Remote: CLOG2
141592 bytes received in 0.78 seconds

Opening data connection for CLOG3
Transfer complete
Local: CLOG3  Remote: CLOG3
82328 bytes received in 0.44 seconds
```

10. Run **reality** and enter dbase1. If the files do not already exist on the database, create the clean log D-pointers and link the visible and binary file names. To do this, enter

```
TL-CREATE-FILE CLOG1 (E)
TL-CREATE-FILE CLOG2 (E)
TL-CREATE-FILE CLOG3 (E)
```

Chapter 7
UNIX Tools

This chapter details the special UNIX utilities available to administer a FailSafe system. They include:

- **fsadm**
- **lockdbase**
- **killreal**
- **mklog**
- **runrealcd**
- **unlockdbase**

fsadm

Purpose

Used to configure and administer databases in FailSafe mode

Syntax

fsadm {*options*} {*local-dbase*}

Parameters

options These are defined below. Enter fsadm with no options to show usage.

local-dbase The path name of the database on the local host. If not specified, the default is the environment variable £REALDBASE. How **reality** processes £REALDBASE to find the database is described in the *Administrator's Guide to Reality X*.

Options

-c Clears failed flag.

-d *remote-dbase* Used when the database name on the remote host is different from the database name on the local host with which it is paired. The database name *remote-dbase* must be specified as an absolute path name.

-f Sets failed flag

-H Shows the local system name. No changes made.

-h *remote-host* Edits config file entries to pair the local host with the system called *remote host*.

-L Switch the config file entry FailsafeAllowLogons to off

-l Switch the config file entry FailsafeAllowLogons to on. With FailSafeAllowLogons set, users can log on to the database as a standalone primary when the secondary is unavailable.

-p Marks *local-dbase* on the local host as a primary FailSafe database. This is flagged in the local raw log header.

-q Query option which shows the current set up of the FailSafe configuration. No changes made.

-R This option is similar to the **-T** option, but is used to swap primary and secondary roles in a FailSafe pair when the primary database fails. When applied to both the primary and associated secondary it reverses the roles, leaving an active stand-alone primary and a failed secondary database.

When applied to the failed primary, it logs off current primary users, locks the database and re-configured it as a failed secondary, that is, the FailSafe failed flag is maintained set. Any active transactions are rolled back.

When applied to the secondary, it unlocks the database and re-configures it as a primary, but with the FailSafe failed flag still set and hence, the database still operates in stand-alone mode.

-r Removes all FailSafe entries from config file.

-s Marks *local-dbase* on the local host as a secondary FailSafe database. This is flagged in the local raw log header.

-T Used to swap primary and secondary roles in an active FailSafe configuration and maintain active FailSafe operation with primary and secondary roles reversed.

When applied to a primary database, it logs off current users, locks the database and re-configured it as a secondary. Any active transactions are rolled back.

When applied to a secondary database, it unlocks the database and re-configures it as a primary.

-t *transport* Specifies network transport protocol. *transport* may be specified as 'TCP' or 'X25'. Without the **-t** option, the default is TCP.

-u Marks *local-dbase* on the local host as unpaired, removing the FailSafe flag from the local raw log header.

Restrictions

Can be used by the super-user or the database owner only. The central daemon must be running and the database must be configured for logging using **mklog**.

Comments

Some **fsadm** facilities can also be executed from Reality X TCL using the FSADM command and associated menu commands described in Chapter 8.

killreal

Purpose

Used to terminate the Reality X daemon process(es).

Syntax

killreal {options}

Options

-y suppresses the WARNING.

-d {*database*} kills daemon for the named database only. The default is £REALDBASE

Restrictions

Can be used by the super-user. The **-d** option can be executed by the database owner as well.

Comments

This command without the **-d** option affects all databases, sending a termination message to the central daemon which in turn sends messages to terminate the database daemons. Each database daemon then broadcasts requests to all associated active reality processes to initiate a controlled and orderly log off. If a **reality** process fails to respond after a period of approximately 30 seconds, the database daemon initiates a forced termination. Reality X is made unavailable on the system.

Using the **-d** option, killreal kills the database daemon for a specified database only. The central daemon and other database daemons and processes are maintained.

Reality X is terminated in an orderly and controlled manner so that the affected database(s) are left in a consistent and predictable state. Note, however, that use of **killreal** in a FailSafe configuration may cause loss of synchronisation. It is therefore recommended that Transaction Logging be shut down using TL-STOP on the primary before **killreal** is executed.

Example

On entering **killreal** the following is displayed:

```
WARNING

This will cause all Reality X databases running on message
queue-id to be killed, and the daemons to exit.

Type 'y' if you are sure you want this:
```

See **runrealcd** to start up the central daemon again.

lockdbase

Purpose

Used to disable all connections to a database, preventing all users, except the database owner and super-user, from logging on. This exception is modified by the **-a** option.

Syntax

lockdbase {*database-name*}

Options

-a	prevents all users, except super-user from logging on.

Parameters

database-name	The name of the database on the local host which is to be locked. The default is £REALDBASE.

Restrictions

Can only be used by the super-user or owner of the database.

Comments

lockdbase only prevents users logging on. it does not log off users that are currently logged on.

A LOCK.FILE in the database directory with zero permissions is used to maintain the lock on the database, hence, the lock is maintained across a system re-boot or shut-down of the Reality X daemons.

lockdbase provides an alternative to the INHIBIT-LOGONS TCL command.

The **-a** option performs the same function as INHIBIT-LOGONS with the A option at TCL.

See **unlockdbase** for unlocking a database.

mklog - Making a Raw Log

Purpose

Used to create a raw log.

Syntax

mklog -r {**-o**} {**-e**} {**-s** *size*}{**-b** size}{**-t**} *partition bin-path*

Parameters

-r	specifies that a raw log is to be created
-o	enables any existing link to the raw log in $REALROOT/bin to be overwritten with a new raw partition link.
-e	empties the raw log
-s *size*	enables a raw log smaller than the specified raw log partition to be created. The default is the size of the partition.
-b *size*	specifies a central buffer cache. Without this option the default is 128 Kbyte.
partition	is the path name of the raw log.
bin-path	is the path name of Reality X binaries (normally £REALROOT/bin).

Examples

For example,

```
mklog -r /dev/rdsk/0s4  £REALROOT/bin
```

initialises the raw log, where 0s4 is the allocated raw partition. This will fail if the raw log already exists.

If a raw log already exists, but you are certain you want to create a new one, for example allocated to a different raw partition (0s3), use the **-o** option, as follows:

```
mklog -or /dev/rdsk/0s3   £REALROOT/bin
```

The **-o** option re-initialises the raw log with the new raw partition path and overrides the current link.

mklog - Making a Clean Log Sub-directory

Purpose

Used to create a clean log sub-directory and set the logging mode for the specified database.

Syntax

mklog {**-o**}{**-c** *sub-dir-name*}{**-m** *log-mod*e} {*clog-dir*} *dbase-path*

Parameters

-o overrides the current clean log sub-directory entry in the database config file to enable the creation of a new clean log sub-directory.

-c *sub-dir-name* enables you to give the clean log sub-directory a different name from that of the database. The default is the database name.

-m *log-mode* enables the logging mode to be set for a specified databases. *log-mode* can be:

F(ULL) Committed transactions are written synchronously to the raw log. A reality process waits until the write to disk is completed before continuing. This ensures that all committed transactions are guaranteed saved. This is at the expense of a performance overhead due to the synchronised write at each transaction commit..

B(RISK) Committed transactions are not synchronised and are written to the raw log periodically, or when the raw log input buffer is full. This means that committed transactions may be lost if a system fail. However, the performance of RealityX is faster.

The choice between FULL and BRISK modes is made according to whether transaction security or database performance is most important.

(N)ONE Transaction Handling only. Transaction Logging is disabled. Only before images are logged.

(O)FF Disables Transaction Handling and Logging.

clog-dir is the full UNIX path-name of the clean log directory on the system. One directory is created per release of RealityX.

dbase-path is the UNIX path-name of the database you wish to configure.

Examples

For example, if you enter

```
% mklog  /clean-logs  /usr/jones/dbase1
```

a clean log sub-directory called dbase1 is created in the clean log directory clean-logs for the database dbase1. This will fail if a sub-directory already exists. The **-o** option must be used to overwrite an existing directory.

If you want, you can specify a different name from that of the database for the clean log sub-directory by using the **-c** option, for example, you may call it dbase1-clogs. To do this, enter:

```
% mklog -c dbase1-clogs  /clean-logs  /usr/jones/dbase1
```

This creates the clean log sub-directory dbase1-clogs in the directory clean-logs for dbase1.

Note: It is recommended that the clean log sub-directory name is the same as the database name.

If only Transaction Handling is required without logging, then enter

```
% mklog -m N /usr/jones/dbase1
```

runrealcd

Purpose Used to start up the Reality X central daemon.

Syntax **runrealcd**

Restrictions Can be used by the super-user only.

Man Pages Information on **runrealcd** is also available on your system. Enter

```
man runrealcd
```

at the UNIX shell prompt to display this information.

Comments The central daemon exercises overall control of the Reality X applications environment. Until the central daemon is started, Reality X is unavailable on the system. This command is normally run automatically at boot time.

unlockdbase

Purpose Used to re-enable all connections to a database, previously locked by **lockdbase** or INHIBIT-LOGGINGS (A).

Syntax **unlockdbase** *database-name*

Parameters database-name The name of the database on the local host which is to be unlocked.

Restrictions Can only be used by the super-user or the owner of the database.

CAUTION

Unlocking a secondary database makes it available for users to log on. Updates performed by users on a secondary database may lead to loss of sychronisation.

Chapter 8
TCL Commands

This chapter details, in alphabetical order, the TCL commands supported by Reality X to administer and operate a FailSafe configuration. This is a reference resource for the rest of the manual, as many of the operational and administrative procedures, described in the manual, require one of these commands. Refer to Chapter 10 for a description of the Transaction Handling commands.

TCL Commands Described in this Chapter

Special TL/FS Commands

FSADM	TL-LOAD
FSADM-PRIMARY	TL-REDUAL
FSADM-SECONDARY	TL-RESTORE
FSADM-STATUS	TL-SET-LOG-STATUS
FSADM-UNPAIR	TL-START
TL-CONTINUE	TL-STOP
TL-CREATE-FILE	TL-STATUS
TL-DUMP	TL-SWITCH
TL-LISTFILES	TL-TRANSACTIONS

Modified Standard Commands

ACCOUNT-RESTORE	CREATE-FILE
CREATE-ACCOUNT	SEL-RESTORE

ACCOUNT-RESTORE

Purpose

To restore one or more accounts from tape.

Syntax

ACCOUNT-RESTORE [*accounts-names*|*] {**(***options*

Special Option for Transaction Logging

L	Specifies that, if Transaction Logging is running on the database, the restored items are to be logged as updates to the database, otherwise the restored items are not logged.

Restrictions

This command cannot be executed inside a transaction. Use is restricted to SYSMAN or SYSPROG.

Comments

For a complete description of this command, with examples, refer to the standard RealityX reference manuals.

CREATE-FILE

Purpose

To create a new file and define its log status

Syntax

CREATE-FILE {**DICT**} *file-name* {**,***data-sect*} *m1*{**,***s1*} {*m2*{**,***s2*}
{**(***options*}

Special Options for Transaction Logging

L	Indicates that the file should not be logged, but that the CREATE-FILE operation itself should be logged.
X	Indicates that the file should not be logged and that the CREATE-FILE operation should not be logged.

Logged by Default

When a file is created by the CREATE-FILE command, it is logged by default. 'DL' is placed in attribute 1 of the file's definition item(s). If only a single level file is created, then only that level is logged.

The special options described above enable the suppression of logging of the file's items or the creation of the file itself.

Comment

For a complete description of CREATE-FILE, with examples, refer to the standard RealityX reference manuals.

CREATE-ACCOUNT

Purpose To create a new account and define the log status of its MD.

Syntax **CREATE-ACCOUNT**

Operation When the account is created, its MD is automatically set up as a file which should be logged.

Files within the account can be individually set as logged or not logged.

Comments For a complete description of this command with examples, refer to the standard RealityX reference manuals.

FSADM

Purpose To configure and administer databases in FailSafe mode. It provides some, but not all, of the functionality of **fsadm**.

Command Class Cataloged DATA/BASIC program.

Syntax **FSADM**

Restrictions Use is restricted to SYSMAN.

Menu Screen The following screen is displayed when you enter FSADM.

```
                    FAILSAFE ADMINISTRATION
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

  1. Show current settings

  2. Mark as primary

  3. Mark as secondary

  4. Remove primary/secondary mark

  Enter option:
```

Menu Options You can select one of the four menu options by entering the appropriate number at the screen prompt. The options and their equivalent TCL/UNIX commands are:

Show current settings	Used to display the current status of the FailSafe configuration. This is equivalent to entering FSADM-STATUS at TCL or **fsadm** with the **-q** option at the UNIX shell.
Mark as primary	Used to mark the database currently logged to as a primary in a FailSafe pair. This is equivalent to entering FSADM-PRIMARY at TCL or **fsadm** with the **-p** option at the UNIX shell.

Mark as secondary	Used to mark the database currently logged to as a secondary in a FailSafe pair. This is equivalent to entering FSADM-SECONDARY at TCL or **fsadm** with the **-s** option at the UNIX shell.
Remove primary/secondary mark	Used to mark the local database, primary or secondary, as unpaired. This is equivalent to entering FSADM-UNPAIR at TCL or **fsadm** with the **-u** option at the UNIX shell.

Comments

Only partially fsadm functionality is supported by FSADM. This is detailed in the description of the menu options given above. Refer to Chapter 7 for a description of full **fsadm** functionality, executed from the UNIX environment.

The equivalent FSADM TCL verbs for each menu option are described in this chapter.

FSADM-PRIMARY

Purpose To mark a database as a primary in a FailSafe pair.

Command Class Cataloged DATA/BASIC program.

Syntax **FSADM-PRIMARY**

Restrictions Use is restricted to SYSMAN.

Comments The primary mark is entered into the config file. Refer to the description of FSADM-STATUS.

The primary can also be marked from the UNIX environment using **fsadm** with the **-p** option.. Refer to Chapter 7.

FSADM-SECONDARY

Purpose To mark a database as a secondary in a FailSafe pair

Command Class Cataloged DATA/BASIC program.

Syntax **FSADM-SECONDARY**

Restrictions Use is restricted to SYSMAN.

Comments The secondary mark is entered into the config file. Refer to the description of FSADM-STATUS.

The secondary can also be marked from the UNIX environment using **fsadm** with the **-p** option. Refer to Chapter 7.

FSADM-STATUS

Purpose To display the current status of the FailSafe configuration.

Command Class Cataloged DATA/BASIC program.

Syntax **FSADM-STATUS**

Restrictions Use is restricted to SYSMAN.

Example The following report is an example of the status information displayed by FSADM-STATUS.

```
FailSafe Pair1:
   Database'/usr/databases/dbase1'
   TCP Host'host1' (Local)
FailSafe Pair2:
   Database'/usr/databases/dbase1'
   TCP Host'host2' (Remote)
Mode:
   Logging inactive
   FailSafe enabled,primary,inactive
```

Explanation of Example This example shows the following information.

- The name of the local system and database in the FailSafe pair.
- The name of the remote system and database in the FailSafe pair.
- The status of logging in the local database, that is, active or inactive.
- The status of FailSafe configuration in the local database, that is, enabled (configured), disabled (not configured).
- The primary, or secondary, marker, if set.
- The status of FailSafe operation in the local database, that is, active, inactive, etc

Comments This status information can also be queried in the UNIX environment using **fsadm** with the **-q** option. Refer to Chapter 7.

FSADM-UNPAIR

Purpose To mark the local database as unpaired.

Command Class Cataloged DATA/BASIC program

Syntax **FSADM-UNPAIR**

Restrictions Use is restricted to SYSMAN

Comments A Failsafe pair can also be unpaired from the UNIX environment using **fsadm** with the **-u** option. Refer to Chapter 7.

SEL-RESTORE

Purpose

To restore one or selected items from a file stored on tape using an FILE-SAVE, ACCOUNT-SAVE, or equivalent.

Syntax

SEL-RESTORE *file*{,*data-sect*} *item-list* **(**options

Special Option for Transaction Logging

L	Specifies that, if Transaction Logging is running on the database, the restored items are to be logged as updates to the database; otherwise the restored items are not logged.

Restrictions

This command cannot be executed inside a transaction.

Comments

For a complete description of this command with examples, refer to the standard RealityX reference manuals.

TL-CONTINUE

Purpose To resume secondary FailSafe operation after the secondary has been suspended by a TL-SWITCH with the H option.

Command Class Cataloged DATA/BASIC program.

Syntax **TL-CONTINUE**

Restrictions Use is restricted to SYSMAN on the primary database.

TL-CONTINUE can be used only after secondary database operation has been suspended by TL-SWITCH with the H option.

Comments All outstanding updates from the secondary's clean log are restored until the databases are synchronised and full FailSafe operation resumed.

TL-CREATE-FILE

Purpose

To create a new log file, that is, clean log, reject log or error log. See Chapter 9.

Command Class

Cataloged DATA/BASIC program.

Syntax

TL-CREATE-FILE *log-file* {**E**}

Syntax Elements

log-file	The name assigned to the new log file.

Options

E	allows the creation of a log file when the UNIX file already exists in the clean log sub-directory. This option re-creates the log dictionary and the D-pointer to it in the Master Dictionary. It also creates a UNIX link to the visible file.

Restrictions

Use is restricted to SYSMAN. The file name must be unique in the SYSMAN Master Dictionary.

Comments

Three types of log files are supported by Reality X Transaction Logging.

- Clean log
- Reject log (TL-REJECT)
- Error log (TL-ERRORS)

TL-CREATE-FILE can be used to create each of these, although TL-REJECT and TL-ERRORS are normally created automatically. See Chapter 9. TL-CREATE-FILE only needs to be executed on the primary database. An identical secondary clean log is created automatically at TL-START.

With logging enabled, log files are created in the clean log sub-directory. However, if Transaction Handling only is specified, then the TL-ERRORS log file is created in the database directory.

TL-LISTFILES can be used to display a list of log files for the database and DELETE-FILE can be used to remove a log from the database.

Example

```
TL-CREATE-FILE DBASE-MON

[CTL] Logfile DBASE-MON created
```

TL-DUMP

Purpose To dump clean log files to a tape device or other system file.

Command Class Cataloged DATA/BASIC program.

Syntax **TL-DUMP** *log-file* {*device*}

Syntax Elements

log-file is the name of the clean log file to be dumped.

device the name of the tape device or system file to dump the clean log to. The default is the device attached by ASSIGN or T-ATT.

Restrictions Use is restricted to SYSMAN on the primary database.

Multiple clean logs should not be archived to one tape as it is not possible to retrieve multiple files using the TL-LOAD verb.

An active clean log should not be dumped.

TL-LISTFILES

Purpose To list information on all clean logs and error logs on the database.

Command Class Cataloged DATA/BASIC program.

Syntax **TL-LISTFILES**

Restrictions Can only be executed from SYSMAN.

Log File Information Displayed

```
Clean log files in SYSMAN at 15:40:02 on 20 JUN 1991

File name             Bytes     Items     Created

TL-ERRORS                 0               11:05  15/06/91
CLOG3                     0         0     22:05  15/06/91
CLOG2             6,093,330    35,100     22:10  16/06/91
CLOG1             4,119,456    25,267     22:09  17/06/91
TL-REJECT                 0               1035      19/06/91
```

The information provided in each column is defined as follows:

`File name` The name of the log.

`Bytes` The size in bytes of the log.

`Items` The number of update items held in the log.

`Created` The creation date and time of the log file.

TL-LOAD

Purpose To load a clean log file from tape

Command Class Cataloged DATA/BASIC program.

Syntax **TL-LOAD** *log-file {device***}**

Syntax Elements

log-file is the name of the clean log file to be loaded.

device the name of the tape device or system file from which the clean log is to be loaded. The default is the device attached by T-ATT.

Restrictions

Use is restricted to SYSMAN on the primary database.

TL-REDUAL

Purpose

To resynchronise databases for normal FailSafe operation.

Command Class

Cataloged DATA/BASIC program

Syntax

TL-REDUAL *new_clog* {*first_clog*}

Syntax Elements

new_clog is the name of empty clean log to which logging on the primary and secondary databases is switched to.

first_clog is the name of the first clean log to be restored onto the secondary database to bring it into line with the primary. The default is the clean log which is active on the primary database prior to switching.

Restrictions

Use is restricted to SYSMAN on the primary database. Logging must be enabled. new_clog on the primary must be empty. On the secondary it will be created or cleared, if necessary. *first_clog* and all chained clean logs, since the last back-up, must be available on the secondary database. If a clean log is unavailable, or empty, TL-REDUAL prompts for it and waits.

CAUTION

TL-REDUAL requires a continuous chain of clean logs between *first_clog* and *new_clog*. Use of TL-STOP followed by a TL-START divides the clean logs into two separate chains.

Operation

TL-REDUAL is entered on the current live database (the primary). Its purpose is to restart logging on the secondary database and to bring the partially-restored secondary database up-to-date with the primary, until the state of both databases is identical (synchronised) and normal FailSafe operation is resumed. To do this:

1. It switches logging to a new empty clean log (*new_clog*) on both the live database (primary) and the partially-restored database (secondary). *new_clog* need only be present on the primary. It is created on the secondary automatically by TL-REDUAL.

   Having switched to *new_clog*, updates on the primary database are logged in both the primary and secondary clean logs. However, they are not applied to the secondary database until it has been restored with earlier updates, as described below.

Note that the clean log switching initiated by TL-REDUAL takes about 5 minutes, you cannot copy the previously active clean log across to the secondary until switching is complete, that is, the TL-STATUS screen displays logging ACTIVE.

2. It also initiates, in parallel with the clean log switching, a restore process on the secondary which restores the updates from the clean logs, required to recover the database, commencing with the *first_clog* specified in the TL-REDUAL command. On completing the restore from *first-clog*, it continue with one or more subsequent clean logs if a chain of logs exists. See below.

Chaining of Clean Logs

A chain of clean logs is created where TL-SWITCH or TL-REDUAL has been used to switch from one clean log to the next. This creates a pointer to the next log. Thus having completed restoring the first clean log, the restore process continues with the next log. Assuming the chain is continuous, the restore process restores all clean logs in chronological order right through to the current log (*new_clog*).

Having completed the restore and synchronised the databases, updates logged in the *new_clog* are once again applied to the secondary database and normal FailSafe operation is resumed.

If the next clean log is not available on the secondary, the restore process displays a message prompt at the system console of the form:

```
Jul 09 16:32:03 #7308 tlrestore WARNING Log /cleanlog failsafe/LOG empty
        Please load new log file
```

This message is repeated every 5 minutes until the required clean log becomes available.

If the clean logs are divided into two separate chains by a TL-STOP/START operation, it will be necessary to restore the first chain using TL-RESTORE with the A option, before continuing with a TL-REDUAL to restore the second chain and resynchronise the FailSafe databases. The earliest log to TL-REDUAL will be the first log used after the restart.

TL-RESTORE

Purpose

initiates a restores all updates, or a selected list of updates, from a specified clean log onto a the database

Command Class

Cataloged DATA/BASIC program.

Syntax

TL-RESTORE *first_clog* {(*options*)}

Syntax Elements

first_clog The name of the first clean log to be restored.

Options

A causes the restore process to sequence through clean logs in chronological order until no more logs exist. Using this option all clean logs in the chain must be present on the database.

AE causes the restore process to sequence though all existing logs in chronological order, then wait at the EOF mark of the last log until the next log is available. If a log is missing or invalid it is prompted for.

Note: The A and AE options require clean logs to be chained together during logging run time in order for them to work.. See the description in this in the section on the chaining of clean logs.

C displays a count of sets of 500 updates applied to the database and information about the images applied.

H{n} specifies the maximum size (n) of the history file item listing the last *n* image ids applied to the database. The default is 2000. The CLOG.PORTS item size is always 20. *n* set to 0 inhibits the history function.

L Prompts you for alternative file names to TL-REJECT and TL-LIST , as follows:

```
ERROR LOG:
HISTORY FILE:
```

When you enter a name, the file is created if it does not already exist. Rejected images and history information for the restore are then entered into the named files.

If you enter RETURN, TL-RESTORE uses files with the default file names, TL-REJECT and TL-LIST, creating them if necessary. This is equivalent to executing TL-RESTORE without the L option, when TL-REJECT and TL-LIST are created automatically without prompts.

R Prevents the error log (TL-REJECT) and history file (TL-LIST) from being cleared by a TL-RESTORE. Rejected images and history information for the current restore are then appended to the files.

If this option is not specified, TL-RESTORE clears these files before using them for the current restore.

S generates an asterisk for each update restored.

Restrictions

Use is restricted to SYSMAN.

TL-RESTORE does not allow you to specify image selection criteria. To initiate selective recovery you must carry out a SELECT operation first; then use TL-RESTORE on the select list.

TL-REJECT and TL-LIST

Initially when TL-RESTORE is executed, it creates two files:

TL-REJECT	Used to log After images that fail to be restored on the database by TL-RESTORE.
TL-LIST	Used to contain history information about the last 2000 images successfully restored and the last 20 images successfully restored per port.

Subsequent execution of TL-RESTORE clears these files before they are used, unless the R option is invoked, in which case the files are not cleared and data is appended. This may be particularly useful during a multi-file restore, to retain previous history information.

Chaining of Clean Logs

TL-SWITCH and TL-REDUAL create a pointer from the current clean log to the new clean log to which logging is switched. Hence this creates a chain of clean logs which can be restored by TL-RESTORE without manual intervention using the A and E options.

TL-RESTORE with the A option commences by restoring the first_clog on the database then continues with the next clean log in the chain Assuming the chain is continuous, the restore process restores all clean logs in chronological order right through to the clean log before the current active one.

TL-RESTORE with the AE option sequences through a chain of clean logs in the same way as the A option, however, if the next clean log is not available, TL-RESTORE prompts for the next log and waits, as follows:

```
Log CLOG2 empty.    Please load new log file.
  Hit A to Abort or C to continue.
```

The chain of clean logs will be interrupted by a TL-STOP/TL-START operation, in which case the restore will terminate at the clean log active when the TL-STOP occurred. If this is the case, it will be necessary to execute a TL-RESTORE with the A option, before continuing a TL-REDUAL. The earliest log to TL-REDUAL will be the first log used after the restart.

Selective Recovery

To carry out a selective recovery you must first execute the SELECT command to compile a select list of the After images and transaction boundary images to be recovered from the clean log, then execute TL-RESTORE. TL-RESTORE will then only restore the images in the select list. For a list of attribute definitions which can be used as selection criteria, refer to Chapter 9. For general information on the use of the SELECT verb, refer to the *ENGLISH Reference Manual*.

CAUTION

You must not use the SORT verb to manipulate items in a clean log as this will cause incorrect sequencing of images during a TL-RESTORE.

TL-SET-LOG-STATUS

Purpose To define or modify the set of files to be logged by Transaction Logging.

Command Class Cataloged DATA/BASIC program.

Syntax **TL-SET-LOG-STATUS**

Restrictions Use is restricted to SYSMAN.

Options 4 and 5 only can be executed when Transaction Logging is active.

Menu Screen On entering **TL-SET-LOG-STATUS** (except for the first time after installation, see note below), Reality X displays a menu screen, as follows:

```
                    SET-LOG-STATUS

  [0]  Exit
  [1]  All system and user accounts
  [2]  All system accounts
  [3]  All user accounts
  [4]  Selected accounts
  [5]  Selected files

Enter option :
```

To select an option, type the associated number, and press RETURN.

Note: When you first run TL-SET-LOG-STATUS after initial installation of Transaction Logging, Option 1 is run automatically.

Explanation of Menu Options

The purpose of the menu options is as follows:

[0] Exits from TL-SET-LOG-STATUS to the TCL prompt.

[1] Allows you to define the logging status of some or all data sections in all accounts on your database (except those with compulsory logging status). You can define whether they are to be all 'Logged', all 'Not Logged' or selectively 'Logged'.

[2] Allows you to define the logging status of some or all system files on your database (except those with compulsory logging status). You can define whether they are to be all 'Logged', all 'Not Logged' or selectively 'Logged'.

[3] Allows you to define the logging status of some or all data sections in all user accounts on your database. You can define whether they are to be all 'Logged', all 'Not Logged' or selectively 'Logged'.

[4] Allows you to select a specific account and define the logging status of some or all data sections in it. You can define whether they are to be all 'Logged', all 'Not Logged' or selectively 'Logged'.

[5] Allows you to select a individual data section/system file and define its logging status as 'Logged' or 'Not Logged'.

Selecting the L, N or S Options

On selecting one of options [1] to [5], messages are displayed prompting you to select the logging status of the accounts and/or files selected by that option. The messages are unique to each option. but the responses asked for are the same, that is, L, N or S. A typical message prompt is:

```
Globally Log all data sections
Enter option (L,N,S) :
```

The three options are

L which sets all specified files/data sections, to be 'Logged', except for those which have the compulsory status of 'Not Logged'.

N which omits all files/data sections from being logged, except for those which have the compulsory status of 'Logged'.

S which allows you to specify the logging status of files/data sections individually, except for those which are preset as 'Compulsory Logged' or 'Compulsory Not Logged'. Entering **S** is followed by further prompts. For example for each account:

```
Log all data sections
Enter Option: (L,N,S):
```

If you enter **S**, the prompt (L, N): is returned for each data section. This enables you to select the logging status of each system file or data section individually.

Completion of Menu Option

When you have finished defining the logging status of all files/data sections allowed by a particular option, you are returned to the main menu. You can then update the logging status of the database again or exit to TCL by entering 0.

Comments

Selecting the Logging Status option

At initial installation, unless there are some special considerations for your installation, it is recommended that you specify L at the `(L, N, S)`: prompt to log the whole database. It is normally easier to set all files/data sections to be logged, and then, if necessary, de-select the files which you do not wish to log.

Compulsory Logging Status

For some files the logging status is preset to 'Compulsory Logged' or 'Compulsory Unlogged', as appropriate, and cannot be changed using TL-SET-LOG-STATUS.

SYSTEM and Master Dictionaries 'Compulsory Logged'

The SYSTEM dictionary is 'Compulsory Logged', as are all account master dictionaries. Making all master dictionaries 'Compulsory Logged' ensures that the logging facility records each CREATE-FILE and MOVE-FILE operation, where a D-pointer is placed in or removed from the MD.

'Compulsory Not Logged' Files

Also, certain files are defined as 'Compulsory Not Logged'. For these files, restoration of updates is unnecessary or may even be undesirable.

The following files are 'Compulsory Not Logged'.

In DENAT account:

BP
ENGLISH
UTILITY

In SYSFILES account:

USER.LOG
SESSION-LOG
ROUTE-FILE
PH-HISTORY
SYSTEM-LOG
LANG.PTRS

Scrolling Through Compulsory Log Status Information

When you select menu options [1], [2] and [3] followed by the **S** option, accounts and files with 'Compulsory Logged' or 'Compulsory Not Logged' status are displayed, but scrolled by automatically. The scrolling stops at the next account or file for which the logging status must be selected.

Example 1 The following is an example of the TL-SET-LOG-STATUS report when you select menu option [1] followed by the L (log everything that can be logged) option.

```
                  SET-LOG-STATUS


     [0]  Exit
     [1]  All system and user accounts
     [2]  All system accounts
     [3]  All user accounts
     [4]  Selected accounts
     [5]  Selected files

    Enter option :  1

Globally Log all data sections

Enter option (L,N,S):  L


SYSTEM

  SYSMAN                                                M/DICT   Compulsory Logged
  SYSTEM                                                M/DICT   Compulsory Logged
  UPGRADE.ACCOUNT                                       M/DICT   Compulsory Logged
  SYSPROG                                               M/DICT   Compulsory Logged
  ENGLISH-TUTORIAL                                      M/DICT   Compulsory Logged
  HOTEL                                                 M/DICT   Compulsory Logged
  DENAT                                                 M/DICT   Compulsory Logged
  SYSFILES                                              M/DICT   Compulsory Logged


SYSMAN

  NETWORK
     NETWORK                                                                 Logged
  SYSPL
     SYSPL                                                                   Logged
  SECURITY
     SECURITY                                                                Logged
  SYSBP.MSGS
     SYSBP.MSGS                                                              Logged

[and so on, listing all files in SYSMAN]
```

Example 1
(Continued)

```
  UPGRADE.ACCOUNT

SYSPROG

  BP
     BP                                                              Logged
  SYSPROG-PL
     SYSPROG-PL                                                      Logged


ENGLISH-TUTORIAL                                                     Logged

HOTEL

  SAVE.BED-CODES
     SAVE.BED-CODES                                                  Logged
  SAVE.ROOMS
     SAVE.ROOMS                                                      Logged
  GUESTS
     GUESTS                                                          Logged
     FEB                                                             Logged

[and so on, listing all files in HOTEL files]


DENAT

  BP
     BP                                                 Compulsory Not Logged
  ENGLISH
     ENGLISH                                            Compulsory Not Logged
  UTILITY.MSGS
     UTILITY.MSGS                                       Compulsory Not Logged


SYSFILES

  BASIC-COMPILERS
     BASIC-COMPILERS                                                 Logged
  CURSOR-DEFS
     CURSOR-DEFS                                                     Logged
  USER.LOG
     USER.LOG                                           Compulsory Not Logged

  [and so on listing all files in SYSFILES, followed by list of all user accounts on the database]
```

Example 2 The following is an example of the TL-SET-LOG-STATUS report when you select menu option [1] followed by the S (Selected files) option.

```
                  SET-LOG-STATUS


     [0]  Exit
     [1]  All system and user accounts
     [2]  All system accounts
     [3]  All user accounts
     [4]  Selected accounts
     [5]  Selected files

    Enter option :  1

Globally Log all data sections
Enter option (L,N,S):  S


SYSMAN

Log all data sections
Enter Option:  (L,N,S): S

 NETWORK
    NETWORK                                                  (L,N) L Logged
 SYSPL
    SYSPL                                                    (L,N) L Logged
 SECURITY
    SECURITY                                                 (L,N) L Logged
 SYSBP.MSGS
    SYSBP.MSGS                                               (L,N) L Logged
```

[and so on, listing all files in SYSMAN]

```
SYSTEM

 SYSMAN                                              M/DICT   Compulsory Logged
 SYSTEM                                              M/DICT   Compulsory Logged
 UPGRADE.ACCOUNT  M/DICT   Compulsory Logged
 SYSPROG                                             M/DICT   Compulsory Logged
 ENGLISH-TUTORIAL M/DICT   Compulsory Logged
 HOTEL                                               M/DICT   Compulsory Logged
 DENAT                                               M/DICT   Compulsory Logged
 SYSFILES                                            M/DICT   Compulsory Logged


UPGRADE.ACCOUNT

Log all data sections
Enter Option:  (L,N,S): L

SYSPROG
```

Example 2
(Continued)

```
Log all data sections
Enter Option:  (L,N,S): S

 BP
    BP                                                        (L,N) L Logged
 SYSPROG-PL
    SYSPROG-PL                                                (L,N) L Logged
ENGLISH-TUTORIAL

Log all data sections
Enter Option:  (L,N,S): L


HOTEL

Log all data sections
Enter Option:  (L,N,S): N

 SAVE.BED-CODES
    SAVE.BED-CODES                                                  Not Logged
 SAVE.ROOMS
    SAVE.ROOMS                                                      Not Logged
GUESTS
    GUESTS                                                          Not Logged
    FEB                                                             Not Logged
```

[and so on, listing all files in HOTEL files]

```
DENAT

 BP
    BP                                                 Compulsory Not Logged
 ENGLISH
    ENGLISH                                            Compulsory Not Logged
 UTILITY.MSGS
    UTILITY.MSGS                                       Compulsory Not Logged


SYSFILES

Log all data sections
Enter Option:  (L,N,S): L

 BASIC-COMPILERS
    BASIC-COMPILERS                                                 Logged
 CURSOR-DEFS
    CURSOR-DEFS                                                     Logged
```

[and so on, listing all files in SYSFILES, followed by list of all user accounts on the database]

TL-START

Purpose Used to start logging and activate FailSafe operation initially.

Command Class Cataloged DATA/BASIC program.

Syntax **TL-START** {*log-file*}

Syntax Elements

log-file The name of an empty clean log to log to. This is omitted to enable Transaction Handling without logging.

Restrictions Use is restricted to SYSMAN on the primary database. TL-START can be used only when all other primary users are logged off. The clean logs used for both primary and secondary must be empty.

Transaction Handling without Logging If a database is configured for 'no logging', TL-START is used without a clean log name to enable Transaction Handling.

Error Messages If logging is configured (using mklog) and you do not specify a clean log, TL-START displays:

```
[CTL2206] You must specify a clean log file name with this command
```

If you specify the name of a clean log which already has data in it, TL-START displays:

`[2108] Logfile` *logfile* `is not empty, use CLEAR-FILE`

The clean log used must be empty. You can either clear a current log using CLEAR-FILE or create a new one using TL-CREATE-FILE.

If you attempt to start logging within 5 minutes of executing a TL-STOP, TL-START responds with the error message

```
You can't do this when logging is in a state of switching/stopping
```

This is because all committed transactions and independent update images sent to the 'old' clean log are retained in the raw log for 5 minutes after being transferred, to ensure that they have reached the clean log.

Refer to Appendix A for a list of TL error messages.

TL-STATUS

Purpose Displays the current status of logging on either the primary or secondary database (S option).

Command Class Cataloged DATA/BASIC program

Syntax **TL-STATUS** {**(***options*}

Options

L *{n}*	Repeats (Loops) the TL-STATUS command every *n* seconds, where n is decimal. Type CTRL+E or X to terminate the loop and return to TCL. If you do not specify the looping period *(n* seconds) , the default is 3 seconds
S	Shows the secondary status only
T	Displays the status of the primary and a list of active transactions.
W	Shows the staus of the database after waiting for clean log switching to complete, at which point it also rings a bell.

Status Information Displayed

```
***Transaction Logging Status at 20:05:01 on 21 AUG 1992***

Status . . . . . . . . . . . . . . . . . . . . . . ACTIVE
Clean Log file  . . . . . . . . . . . . . . . . . .CLOG3
Recovery file . . . . . . . . . . . . . . . . . . .CLOG2
Raw log items waiting . . . . . . . . . . . . . . 89
Clean log items logged . . . . . . . . . . . . . .3130
Clean log in use . . . . . . . . . . . . . . . . . 560,253
Transactions open . . . . . . . . . . . . . . . . .2
Database Recovery mode . . . . . . . . . . . . . .FULL RECOVERY
Time of last status change . . . . . . . . . . . . 11:41 14 JAN 1992

Size of raw log . . . . . . . . . . . . . . . . . .1,994,624 bytes
Raw log usage . . . . . . . . . . . . . . . . . . . 6%
Maximum raw log usage . . . . . . . . . . . . . . 16%
Post processor status . . . . . . . . . . . . . . PRESENT

TL-RESTORE file and status . . . . . . . . . . . . CLOG  Image   3250   Time  10:35

FailSafe configured as . . . . . . . . . . . . . .PRIMARY
FailSafe status . . . . . . . . . . . . . . . . . .ACTIVE
Restoring from . . . . . . . . . . . . . . . . . . .CLOG2
Sequence/time so far restored . . . . . . . . . . .3250, 14:15  MAY 1992
```

The status information given on each report line is:

`Status`

Current state of the logging. This may be one of the following:

ACTIVE	In progress, initiated by TL-START.
INACTIVE	Supported, but not in progress.
SWITCH REQUESTED	A switch to a new clean log has been requested, but not yet actioned.
SWITCH IN PROGRESS	In the process of changing to a new clean log. It will remain in this state until the old clean log is no longer required.
STOP IN PROGRESS	Writing of updates to raw log is stopped, but committed transactions and independent updates are still being flushed to the clean log.
STOPPED	Inactive, but still maintaining images in the raw log until all committed transactions and independent updates have been flushed from the UNIX buffers to the clean log.
PASSIVE -RESTORING (secondary only)	The secondary database is being restored with primary updates.
ACTIVE -SECONDARY PAUSED	Logging to the primary and secondary logs is active, but updating of the secondary database has been suspended by TL-SWITCH with the H option.

`Clean log file`

Name of the clean log to which transactions are currently being logged.

`Recovery file`

Name of the previous clean log retained while in a state of switching or switched.

`Raw log items waiting`

The number of 'After' images and transaction boundary images still held in the raw log.

`Clean log items logged`

The number of 'After' images saved in the clean log.

`Clean log in use`
The number of bytes currently stored in the clean log.

`Transactions open`
The number of transactions which are still active and open in the raw log.

`Database recovery mode`
The mode of recovery supported by the logging. The status options are, FULL RECOVERY or NONE.

`Time of last status change`
The time in hours:minutes and the date when logging status was last changed.

`Size of raw log`
The size of the raw log in bytes.

`Raw log usage`
The proportion of the raw log currently filled with images.

`Maximum raw log usage`
The maximum proportion of the clean log filled during the period of its use.

`Post processor status`
Indicates the presence or absence of the post processor.

`TL-RESTORE file and status`
Displays the status of a full restore in progress.

`Failsafe configured as`
Indicates whether PRIMARY or SECONDARY status is displayed

`Failsafe status`
Current state of the failsafe operation. This may be one of the following:

ACTIVE	FailSafe logging has been activated by TL-START on primary.
INACTIVE	FailSafe logging has been deactivated by TL-STOP on primary.
IDLE	FailSafe operation is disabled

TL-STOP

Purpose

Used to perform a controlled close-down of FailSafe and disable logging.

> **CAUTION**
>
> Use of TL-STOP ends the current chain of clean logs and TL-START starts a new chain. Hence, in order to maintain a continuous chain of logs for TL-RESTORE or TL-REDUAL purposes, it is recommended that you use TL-SWITCH and not TL-STOP/TL-START.

Command Class

Cataloged DATA/BASIC program.

Syntax

TL-STOP

Restrictions

Use is restricted to SYSMAN on the primary database. TL-STOP can be used only when all other users are logged off.

Comments

There may be a condition when the close-down cannot be completed because there are one or more transactions still in progress. An open transaction may be caused, for example, by an absent operator leaving a transaction open. You can use the TL-TRANSACTIONS command to see which users are within transactions, and when the transactions were started. This information may help with the decision to contact a user who can then terminate the transaction with TRANSEND, or you can log off the process, which will force a TRANSABORT.

TL-SWITCH

Purpose

Used to switch clean logs while logging is enabled. It can also be used to suspend update operations to the secondary database or close down the secondary database permanently, while maintaining the primary as a standalone database.

Command Class

Cataloged DATA/BASIC program.

Syntax

TL-SWITCH *log-file* **{(***options***}**

Syntax Elements

log-file	The name of the primary clean log to switch to.

Options

H	Suspends the restore process which applies updates to the secondary database.
K	Kills the secondary database.

Restrictions

Use is restricted to SYSMAN on the primary database. Logging must be enabled and the clean log named must be empty. The associated secondary log is created or cleared automatically.

Comments

The H option causes logging to switch to an empty clean log on both the primary and the secondary and suspends the restore process from updating the secondary. This allows the secondary database to be temporarily stopped for backup purposes, while maintaining an active primary database with logging to both primary and secondary logs.

Although secondary database operations are stopped, synchronisation is not lost, as updates are still logged to the secondary log. The TL-CONTINUE can be used to restart the restore processes which re-synchronises the secondary with the primary and resumes full FailSafe operation.

The K option disconnects the FailSafe link so that the secondary database becomes idle. In this case synchronisation is lost. It must then be recovered using TL-REDUAL as described in Chapter 6.

TL-TRANSACTIONS

Purpose

To display information about active transactions currently open on the database

Command Class

Cataloged DATA/BASIC program.

Syntax

TL-TRANSACTIONS {**(***options*}

Syntax Elements

L *{n}* Repeats (Loops) the TL-TRANSACTIONS command every *n* seconds, where n is a decimal, so that the screen is continually refreshed and the active transactions information updated. Enter CTRL+E or X to terminate the loop and return to TCL. If you do not specify the looping period *(n* seconds) in the command line, the default is 3 seconds.

Restrictions

Use is restricted to SYSMAN on the primary database.

Transaction Information Displayed

```
***Active transactions at 09:30:09 on 24 JUN 1991.  Page 1 of 1.***

PORT  USER ID...LOCATION.................    TRANSACTION START

 *24  SYSMAN   Room 5                        16:46   02 APR 19
 110  TYPING   Room 12                       17:56   02 APR 19
```

The information contained in each column is as follows:

PORT
The port from which the transaction was started.

USER ID
Identity of the user that started the transaction.

LOCATION
The location from which the transaction was started.

TRANSACTION STARTED
The time and date that the transaction was started.

Chapter 9
Log Files

This chapter describes the purpose and structure of three types of log file and a standard RealityX file. These are:

• Clean log

• Reject log, default name TL-REJECT

• Error log, default name TL-ERRORS

• History file, default name TL-LIST

It describes how you can use ENGLISH to examine these logs and carry out a selective recovery of items.

Overview

Reality X supports three types of log file on a database, each with the same file structure and created in the database's clean log sub-directory. They are:

Clean Log	Used to log committed transactions and independent updates applied to the database. It stores their 'After' images and transaction boundary images. A clean log is created using TL-CREATE-FILE.
Error Log (TL-ERRORS)	Used to log images of uncommitted transactions, still in the raw log, which fail to be applied to the database when a system is re-booted after a crash. The TL-ERRORS log is created automatically by TL-START. The file name TL-ERRORS is mandatory.
Reject log (TL-REJECT)	Used to log 'After' images which cannot be applied to the database by a TL-RESTORE. The TL-REJECT log is created automatically by a TL-RESTORE. TL-RESTORE with the L option allows you to specify an alternative name for the reject log, instead of TL-REJECT.

In addition, to these three log files, RealityX supports a normal RealityX file called:

History File (TL-LIST)	This file, as its name implies, contains history information about images successfully applied to a database by a TL-RESTORE. TL-LIST is created automatically by a TL-RESTORE. TL-RESTORE with the L option allows you to specify an alternative name for the file, instead of TL-LIST. The contents of TL-LIST are described in this chapter.

The ENGLISH retrieval language is used to display and analyse the contents of the log files.

Log Files

The clean log, reject log (TL-REJECT) and error log (TL-ERRORS) files all have the same structure, consisting of a dictionary with two data sections, one containing binary data and one containing ASCII data which can be viewed by the user.

For example, CLOG1, has a dictionary, DICT CLOG1, with two data sections, CLOG1 and CLOG1,BINARY.

CLOG1	This is the default data section containing items with 'visible' ASCII formatted attributes, extracted from the binary disk images, giving information about the logged update held in the log file. It is this data section which is accessed and viewed by the user using the dictionary name CLOG1, to list the log statistics, for example, LIST CLOG1. The user view is strictly read-only. Writing to this data section is not permitted.
BINARY	This is a non-default data section containing binary items which represent the logged images exactly as stored on disk. Items from this data section are not normally viewed by the user. When necessary it is referenced as CLOG1,BINARY.

The log dictionary contains D-pointers to the data sections, a set of attribute definition items which are used by ENGLISH to generate a meaningful listing of the visible log and a number of macros to help in the production of useful listings. The attributes defined in the visible log item are described below.

Log Item Format

item-id	OFFSET. This is an ASCII representation of the hex offset of the image in the binary file.
001	TYPE. The type of log image. This may be one of the following: Start, Switch, Before, After, Commit and Pre-commit. The Commit image is logged at 'Transaction end'. Before and Pre-commit images are logged in the TL-ERRORS log only.
002	SERVICE. The RealityX service which generated the image. This may be either REALITY File Services (RFS) which generates update images and REALITY Transaction Services (RXS) which generates transaction boundary images.
003	Reserved for future use
004	DATE. The date when the image was first logged, stored in internal format.

005	TIME. The time when the image was first logged, stored in internal format.
006	RLOGSEQ. The transaction id which is the sequence number of the transaction COMMIT image. All images in the committed transaction have the same RLOGSEQ id as the COMMIT image. Independent updates each have different ids.
007	CLOGSEQ. The sequence number of the image in the clean log.
008	PORT. The number of the port being used when the image was logged.
009	RESULT. This is a failure code which will appear in an error log image. It indicates the reason for the failure to restore the update. Clean log items where recovery has not been attempted or where recovery has been successful contains a '0'. You can use the **perror** at the UNIX shell to interpret the code and find out the reason for the failure. The use of **perror** is explained in a **man** page.
010	USER. The RealityX user id being used when the image was logged.
011	ACCOUNT. The RealityX account id being used when the image was logged.
012	FILENAME. The RealityX file for which the image was logged.
013	This is defined as one of three attributes: INFO. Information field from TRANSTART/TRANSEND/TRANSABORT image. ITEM. The item id of the associated item, except transaction boundaries. ITEMINFO. This combines the previous two attribute definitions and can be used instead of them to display both an information field from a TRANSTART/TRANSEND image and an item-id from update images, as appropriate.
014	OPERATION. The type of operation for which the image was logged.

History File - TL-LIST

TL-LIST contains two items for each log restored, *log* and *log*.PORTS, each containing a list of images which have been successfully applied to the database during a TL-RESTORE. For example, TL-LIST contains the following two items corresponding to CLOG1.

CLOG1	which lists the item-ids of the last 'n' images successfully applied to the database. The value of 'n' is set by TL-RESTORE using the H option, the default being 2000. Refer to Chapter 8.
CLOG1.PORTS	which lists item-ids for the last 20 images successfully applied to the database from each port on the database. Lists for each port are concatenated in the same item.

TL-LIST is the default named file which is automatically created by TL-RESTORE. However, if you enter TL-RESTORE with the L option, you are prompted for a name for the History File. You can then specify a different name. Refer to the description of TL-RESTORE in Chapter 8.

The ENGLISH verb NEW-GET-LIST is used to retrieve a list of items from TL-LIST for display. This facility is described next in the section on 'Using ENGLISH to Examine Logs'.

Using ENGLISH to Examine a Log

The ENGLISH retrieval language can be used to examine a log and analyse the information in the logged image items. Most of the facilities supported by ENGLISH can be used. An exception is the SORT verb which must not be used as it rearranges the order of images in the clean log. This affects the sequence in which images are restored on a database leading to data corruption.

Refer to the *ENGLISH Reference Manual* for details on the ENGLISH facilities referred to in this section.

> **CAUTION**
>
> You must not use the SORT verb to manipulate items in a clean log as this will cause incorrect sequencing of images during a TL-RESTORE.

Log Item Attributes

Log file information can be retrieved under 16 attribute names. They are:

TYPE	SERVICE	DATE	TIME
RLOGSEQ	CLOGSEQ.	PORT	RESULT
FILENAME	ACCOUNT	OPERATION	USER
ITEM	ITEMINFO	INFO	

The definitions of these attributes are given earlier in this chapter under the description of the log item format.

You can retrieve each piece of data stored in the log items using the LIST verb and by specifying the attribute(s) required.

Macros

To save time typing you can define an ENGLISH macro to execute a predefined ENGLISH statement and retrieve a predefined set of log item attributes for display on your screen.

Examples of these are the VIS and VISF macros provided on your installed database. VIS is used to list the default attributes. LIST CLOG1 displays the same as LIST CLOG1 VIS. VISF displays the RealityX File details for each clean log image in sequential order. If you enter an ENGLISH statement of the type:

```
LIST CLOG1 VISF
```

ENGLISH displays a report similar to the following.

```
CLog. Time.  Type..... Port User..  Account. File...... Item/Info. Operation..


 0   12:17  Start     24                                          START
 1   12:17  After     24  SYSMAN  HOTEL    BED-CODES   Q         DELETE ITEM
 2   12:17  After     24  SYSMAN  HOTEL    BED-CODES   WB        DELETE ITEM
 3   12:17  After     24  SYSMAN  HOTEL    BED-CODES   D         DELETE ITEM
 4   12:17  After     24  SYSMAN  HOTEL    BED-CODES   K         DELETE ITEM
 5   12:17  After     24  SYSMAN  HOTEL    BED-CODES             CLEAR SECT
 6   12:17  After     24  SYSMAN  HOTEL    GUESTS      140       DELETE ITEM
 7   12:17  After     24  SYSMAN  HOTEL    GUESTS      140       DELETE ITEM
 8   12:17  After     24  SYSMAN  HOTEL    GUESTS      140       DELETE ITEM
 9   12:17  After     24  SYSMAN  HOTEL    GUESTS      140       DELETE ITEM
 10  12:17  After     24  SYSMAN  HOTEL    GUESTS      140       DELETE ITEM
 11  12:17  After     24  SYSMAN  HOTEL    GUESTS      140       DELETE ITEM
 12  12:17  After     24  SYSMAN  HOTEL    GUESTS      140       DELETE ITEM
 13  12:17  After     24  SYSMAN  HOTEL    GUESTS      140       DELETE ITEM
 14  12:17  After     24  SYSMAN  HOTEL    GUESTS      140       DELETE ITEM
 15  12:17  After     24  SYSMAN  HOTEL    GUESTS      140       DELETE ITEM
 16  12:17  After     24  SYSMAN  HOTEL    GUESTS      140       DELETE ITEM
 17  12:17  After     24  SYSMAN  HOTEL    GUESTS      140       DELETE ITEM
 18  12:17  After     24  SYSMAN  HOTEL    GUESTS      140       DELETE ITEM
```

Using TL-LIST

You can select one of the two item lists in TL-LIST using the ENGLISH verb NEW-GET-LIST, then examine the associated images in the clean log using the LIST verb. For example, enter

```
NEW-GET-LIST TL-LIST CLOG1.PORTS
```

This retrieves the CLOG1.PORTS list item from TL-LIST. The system responds with

```
160  ITEMS SELECTED
>
```

You then use the LIST verb to display some or all of the selected items in the clean log CLOG1. For example, enter

```
LIST CLOG1 WITH USER = "SYSMAN"
```

This will then display details of updates made by SYSMAN and applied in the last 20 images from each active port on the database.

Selective Recovery

The SELECT verb is used to choose a subset of items in the clean log in order to carry out a selective recovery of the database.

Chapter 10
Applications Interface

This chapter describes the methods used to create, or modify, TCL, PROC, DATA/BASIC and ALL applications for transactions. Execution of a transaction start, end, abort and query are deatailed for each application. An example of a DATA/BASIC program containing transactions is also provided.

Introduction

Defining a Transaction

A transaction is defined by issuing a command to start the transaction followed by a command to end it. Updates between the start and end of the transaction belong within that transaction.

Note: An elementary introduction to the nature of transactions is given in Chapter 2.

Two commands are supported to mark out a completed transaction: 'transaction start' and 'transaction end'. These may be issued from TCL, PROC, DATA/BASIC or ALL. It is recommended as good programming practice that both commands are issued by the same language, that is, TCL, PROC, DATA/BASIC or ALL, although, technically, this is not necessary,

A 'transaction abort' command may be issued which will undo all of the updates performed since the transaction start: again, the abort may be issued from TCL, PROC, DATA/BASIC or ALL.

Optimum Size of Transaction

Transactions in general should be made as small as possible, to give maximum resilience to the system (minimise the work lost in the event of a system failure) and minimum impact on performance. Performance may be affected by large transactions, since during a transaction the release of item locks is suspended: this may prevent other transactions proceeding. Very large transactions also increase the possibility of deadly embraces (see Glossary).

Note: The possibility of deadly embraces can be reduced by always processing the same set of files and/or items in the same order.

Aborting or Ending a Transaction

When you LOGOFF or LOGTO another account whilst you are in a transaction, a message will be displayed giving you the option to abort or end the transaction.

If a port is logged off remotely whilst inside a transaction, no message is displayed and the transaction is forced to abort.

Aids to Update Analysis

Transaction starts, ends and aborts may be given identity labels. This enables the administrator to identify the following:

- updates and complete transactions which have been restored or saved in the clean log.
- transactions which were started but not finished due to failure before transaction end.
- complete transactions and updates which were rejected when an attempt was made to restore them, and the reason for rejection.

The administrator may then use this information to determine where to restart applications following a restore, when updates were done and so on. These labels are shown in the ITEM IDENTITY column of the standard listings of clean logs and TL-REJECT files.

Item Locking

This is a mechanism to prevent multiple processes accessing the same item at the same time. Transaction Handling suspends the release of item locks set within transactions to prevent interaction between transactions and other processes causing inconsistencies in the database. Item locks set before the start of a transaction are not released at transaction end. Also, once inside a transaction, Transaction Handling suspends item locks set outside the transaction.

Avoid File Creation/Deletion Within a Transaction

It is strongly recommended that you do not create or delete a file inside transaction boundaries. If you execute a CREATE-FILE command inside a transaction, the file is not committed to the database until transaction end. However, Transaction Handling is unable to keep a lock on the file, which means that other processes can use the file before it is committed.

For example:

1. Process A opens a transaction and executes a CREATE-FILE command.

2. Process B opens the file and begins creating items in the file via a DATA/BASIC application.

3. Process A executes a TRANSABORT; the CREATE-FILE operation is rolled back and the file space is returned to the system.

4. Process B is unaware of the roll-back and continues to write items to the space where the file once was, but when the file is closed, the UNIX file is deleted.

TCL/PROC Interface to Transactions

The Transaction Handling commands which can be executed from TCL or PROC are detailed in the following pages. These include:

TRANSTART	to mark the start of a transaction.
TRANSEND	to mark the end of a transaction.
TRANSABORT	to undo all updates performed by the current transaction.
TRANSQUERY	to determine the transaction status of a port.

These commands can also be executed as statements in DATA/BASIC. These are discussed later in the chapter.

TRANSTART Verb

Purpose

TRANSTART is executed to mark the start of a transaction.

Syntax

TRANSTART {*transaction-information*}

Syntax Elements

transaction-information optionally specifies the text to be saved in the TRANSTART item which is logged. This information can be useful when examining the item in the clean log, or during a TL-RESTORE. By making this parameter describe what you are doing (for example, PAY-EMPLOYEE-28090) it can be used to identify a particular transaction, or iteration of a repetitive transaction. A space is used as a delimiter.

Operation

Initially, TRANSTART checks that the current process is not within a transaction (transactions may not be nested) and that Transaction Logging is enabled. If this is not the case, it displays an appropriate error message and exits. Otherwise, it writes a TRANSTART item to the raw log.

Error Messages

Refer to Appendix A for descriptions of error messages.

TRANSEND Verb

Purpose

TRANSEND is executed to mark the end of a transaction and to 'commit' (see Glossary) the updates performed by the transaction.

Syntax

TRANSEND {*transaction-information*}

Syntax Elements

transaction-information optionally specifies the text to be saved in the TRANSTART item which is logged. This information can be useful when examining the item in the clean log, or during a TL-RESTORE. By making this parameter describe what you are doing (for example, PAY-EMPLOYEE-28090) it can be used to identify a particular transaction, or iteration of a repetitive transaction. A space is used as a delimiter.

Operation

Initially, TRANSEND tests whether a transaction is open. It then sets up a TRANSEND image in the raw log. All item locks that were set during the transaction are released and the transaction is committed.

Error Messages

Refer to Appendix A for descriptions of error messages.

TRANSABORT Verb

Purpose

TRANSABORT is executed to undo the updates performed inside the current transaction and release all item locks set during the transaction. The item locks are released only after the TRANSABORT image has been logged to the raw log.

Syntax

TRANSABORT {*transaction-information*}

Syntax Elements

transaction-information	optionally specifies the text to be saved in a TRANSABORT item. This information may be useful when examining the TL-ERRORS log. TRANSABORT images are not logged to the clean log during normal logging operations.

Operation

The undoing of updates inside a transaction is called roll-back. This is executed by restoring the 'Before' images of all updates inside the aborted transaction onto the database. The database is therefore 'rolled back' to its pre-transaction consistent state. All the other ports on the system remain active whilst this restore procedure is being carried out.

Error Messages

Refer to Appendix A for descriptions of error messages.

TRANSQUERY Verb

Purpose

This command is used to determine the transaction status of the port currently in use. (Use TL-STATUS for status of other ports.)

Operation

The port's transaction status is indicated by one of the following messages:

- `[CTL1151] Transaction Logging is not enabled`

  Transaction Handling is installed on the system but not enabled.

- `[CTL1155] There is a transaction already active for this process.`

  Transaction Handling is installed and the port performing the TRANSQUERY is inside a transaction.

- `[CTL1156] There is no currently active transaction for this process.`

  Transaction Handling is installed on the system but the port performing the TRANSQUERY is not inside a transaction.

DATA/BASIC Interface to Transactions

Like TCL and PROC, DATA/BASIC supports four Transaction Handling statements. Three to mark transaction boundaries and one to monitor transaction status. These commands are detailed in the pages following:

- TRANSTART
- TRANSEND
- TRANSABORT
- TRANSQUERY

More than one transaction may occur within a single DATA/BASIC program, or a single transaction may span several CHAINed programs. However, transactions may not be nested, i.e. a TRANSTART may not be followed by another TRANSTART without an intervening TRANSEND or TRANSABORT. This would cause a run-time error whereby the ELSE clause in the TRANSTART statement would be executed.

Note: If Transaction Logging has not been installed and enabled, then the ELSE clause is used in every case.

The function of item READ/WRITE statements and RELEASE statements is altered so that item locks set within a transaction are not released until transaction-end (or transaction-abort).

An example of a DATA/BASIC program using transactions is given later.

TRANSTART Statement

Purpose

Marks the start of a transaction, and precedes the first READ/WRITE operation on the database included in the transaction.

Syntax

TRANSTART {*transaction-information*} [**THEN** *statements*| **ELSE** *statements*]

Syntax Elements

transaction-information — optionally specifies additional text to be saved in the 'transaction start' record. This information can be useful when examining a clean log.

If you do not supply this parameter, *transaction-information* comprises the file-name and the item-name containing the program performing the transaction.

THEN *statements* — is a clause which specifies the statement(s) to be executed if transaction-start is successful.

ELSE *statements* — is a clause which specifies the statement(s) to be executed should the transaction-start fail (for example, transaction is already active or Transaction Logging is not enabled.)

Examples of transaction-information

If this parameter contains text describing the purpose of the transaction, it can be used to easily identify that particular transaction. For example,

```
TRANSTART ORDER ENTRY TRANSACTION ELSE GOTO 500
```

Alternatively, you can use variables to identify a particular iteration of a repetitive transaction. For example,

```
TRANSACTION.INFORMATION=ORDER:"-":CUSTOMER
TRANSTART TRANSACTION.INFORMATION ELSE GOTO 500
```

or

```
TRANSACTION.INFORMATION = "PAY-EMPLOYEE": PAYROLLNUM
TRANSTART TRANSACTION.INFORMATION ELSE GOTO 500
```

TRANSEND Statement

Purpose

Marks the end of a transaction and follows the last READ/WRITE operation on the database included in the transaction.

Syntax

TRANSEND {*transaction-information*} [**THEN** *statements* **ELSE**| *statements*]

Syntax Elements

transaction-information — optionally specifies text to be saved in the TRANSEND item which is logged. This information can be useful when examining a clean log. You can use literals or variables to identify a transaction or iteration of a repetitive transaction. (See description of TRANSTART statement.)

If you do not supply this parameter, *transaction-information* comprises the file-name and the item-name containing the program performing the transaction.

THEN *statements* — is a clause which specifies the statement(s) to be executed if the transaction-end is successful.

ELSE *statements* — is a clause which specifies the statement(s) to be executed should the transaction-end fail (for example, no transaction is active or Transaction Logging not enabled.)

TRANSABORT Statement

Purpose

This statement aborts the current transaction and undoes any updates to the database performed by it.

Syntax

TRANSABORT {*transaction-information*} [**THEN** *statements*| **ELSE** *statements*]

Syntax Elements

transaction-information	Optionally specifies text to be saved in the 'transaction abort' record. This information may be useful when examining a TL-ERRORS item. TRANSABORT is not logged to the clean log during normal logging. If you do not supply this parameter *transaction-information* comprises the file-name and the item-name containing the program performing the transaction.
THEN *statements*	is a clause which specifies the statement(s) to be executed if transaction-abort is successful.
ELSE *statements*	is a clause which specifies the statement(s) to be executed should transaction-abort fail (for example, no transaction is active) or Transaction Logging not be enabled. This clause is mandatory if you have not included a THEN clause.

Operation

The undoing of updates inside a transaction is called roll-back. This is executed by restoring the 'Before' images of all updates inside the aborted transaction onto the database. The database is therefore 'rolled back' to its pre-transaction consistent state. All the other ports on the system remain active whilst this restore procedure is being carried out.

TRANSQUERY Function

Purpose

This function is used to determine the transaction status of the current port. Alternative statements can then be executed, depending on the transaction status.

Syntax

IF TRANSQUERY() [THEN *statements* | **ELSE** *statements*]

Operation

The function TRANSQUERY() will evaluate to true (1) if the process is inside a transaction, or to false (0) if the process is not inside a transaction.

Example of Transaction Boundaries in a DATA/BASIC Program

The following program illustrates the use of Transaction Handling commands in DATA/BASIC and also illustrates use of item locks.

```
PAGE 1               DATA/BASIC        15:36:30 15 MAY 1990
PROGRAM1

001     *VERSION 0001
002     *-----------------------------------------------------*
003     * This program demonstrates the use of the Transaction*
004     * Handling commands and also some of the @(-n)        *
005     * commands.                                           *
006     *-----------------------------------------------------*
007     * Copyright:McDonnell Douglas Information Systems 1989*
008     *-----------------------------------------------------*
010     * Open all relevant files and do any necessary
011       initialisation
012     *
013     OPEN 'DATA1' TO DATA1 ELSE STOP 201, 'DATA1'
014     OPEN 'DATA2' TO DATA2 ELSE STOP 201, 'DATA2'
015     OPEN 'DATA3' TO DATA3 ELSE STOP 201, 'DATA3'
016     *
017     KEY = ''
018     RECORD= ''
        *
        *-----------------------------------------------
019 10  * This is the start of the main transaction loop
020     * It simply requests an item id from the user and
021     * then locks that item in each of three files.
022     * The program then prompts the user to enter data
023     * for each of the three files and updates the files
024     * as the data is entered (instead of doing all the
025     * updates at the end of the transaction).
026     * The user is allowed to abort the transaction at
027     * any input field by entering '/' which calls a
028     * transaction abort, automatically rolling back
029     * all of the updates which have taken place and
030     * also releases all of the locks previously set.
031     *-----------------------------------------------
032     *
033     CRT @(-1): ; * Clear the screen
034     *
035     CRT @(10,5): "Enter record key ": ; INPUT KEY
036     *
037     CRT @(-13) ; * Clear line 25
038     *
039     IF KEY = '' OR KEY = '/' THEN CRT @(-1): ; STOP
040     *
041     TRANSTART THEN
```

```
042     .  *
043     .  * The transaction is now in progress and the log
044     .  * will contain a start entry for this port plus
045     .  * the name of the program which started it.
046     .  *
047     .  READU RECORD FROM DATA1,KEY THEN
048      . .CRT @(-13): "Item '":KEY:"' exists in file
049        'DATA1'":@(-14):
050      . .   SLEEP 1
051      . .   GOTO 100 ; * Abort the transaction, it exists
052      . END
053      . *
054      . READU RECORD FROM DATA2,KEY THEN
055      . .CRT @(-13):"Item'":KEY:"' exists in file
056        'DATA2'":@(-14):
057      . .   SLEEP 1
058      . .   GOTO 100 ; * Abort the transaction, it exists
059      . END
060      . *
061      . READU RECORD FROM DATA3,KEY THEN
062      . .CRT @(-13):"Item '":KEY:"' exists in file
063        'DATA3'":@(-14):
064      . .   SLEEP 1
065      . .   GOTO 100 ; * Abort the transaction, it exists
066      . END
067      . *
068      . * All records are now locked and don't exist
069      . * Now get the data for each record and update them
070      . *
071      . *-------------------------------------------------
072      . CRT @(10,8): "Enter data for record 1": ; INPUT
073        RECORD
074      . *
075      . IF RECORD = '/' THEN GOTO 100 ; * Transaction abort
076        request.
077      . *
078      . WRITE RECORD ON DATA1,KEY; *File updated but lock
079        not released
080      . *-------------------------------------------------
081      . CRT @(10,10): "Enter data for record 2": ; INPUT
082        RECORD
083      . *
084      . IF RECORD = '/' THEN GOTO 100 ; * Transaction abort
085        request.
086      . *
087      . WRITE RECORD ON DATA2,KEY; *File updated but lock
088        not released
089      . *-------------------------------------------------
090      . CRT @(10,12): "Enter data for record 3": ; INPUT
091        RECORD
092      . *
093      . IF RECORD = '/' THEN GOTO 100 ; * Transaction abort
```

```
094       request.
095     . *
096     . WRITE RECORD ON DATA3,KEY; *File updated but lock
097       not released
098     . *-------------------------------------------------
099     . *All inputs and updates are complete so 'commit'
        . * the transaction
        . TRANSEND "Transaction completed - ITEM ID =
          '":KEY:"'" THEN
        . . *
        . .CRT @(-13):"Transaction accepted and logged -Item
           Id =
            '":KEY:"'":@(-14):
        . . GOTO 10 ; * Start another transaction
        . . *
        . END ELSE
        . . *
        . . *The ELSE clause will be taken for the following
             reasons:-
        . . * 1.If no TRANSTART command has previously been
               executed.
        . . * 2.If Transaction Logging has not been enabled
               on the
100       *    machine
101     . . *
102     . . GOTO 100 ; * Unable to commit the transaction so
103       attempt an
104       abort
105     . END
106     . *
107       END ELSE
108     . *
109     . * Unable to start a transaction so quit the program
110     . *This ELSE clause will be taken for the following
111       reasons
112     . * 1.If a transaction is already in progress
113     . * 2.If transaction Logging has not been enabled on
114         this
115          machine
116     . *
117     . CRT @(-13):"Unable to start a new transaction -
118       program
          cancelled":@(-14):
        . STOP
          END
          *
119 100   * Transaction abort routine
120       *
121       * Abort the current transaction incorporating some
122         text
123       * and the item id into the transaction log
124       *
```

```
125      TRANSABORT "Abort transaction - ID = '":KEY:"'" THEN
126      . *
127      . * Display a message on line 25
128      . *
129      . CRT @(-13):"Your transaction has been aborted":@(-
130        14):
131     . *
132     . GOTO 10 ; * Prompt for a new transaction start
133     . *
134     END ELSE ;* Cannot abort the transaction for some
135     reason
136     . *
137     . * The ELSE clause will be taken for the following
138         reasons:-
139     . * 1.If no TRANSACTION command has previously been
140           executed.
141     . * 2.If Transaction Logging has not been enabled on
142           the
143     . *   machine
        . *
        . CRT @(-13):"Unable to abort the transaction -contact
          your
        . supervisor":@(-14):
        . ABORT ; * EXIT COMPLETELY FROM ALL PROCESSING
        END
        END
```

ALL Interface to Transactions

In ALL, transactions can be defined at two levels:

- Function level, where a transaction consists of one or more complete functions, OR
- Block level, where a transaction comprises one logical update, screen or report.

Function Level Transaction Boundaries

In this case a transaction consists of one or more complete functions. Transaction boundaries are defined via the function definition screen. An ALL function may be specified as one of the following:

- A transaction on its own: that is, both the start and the end of a transaction.
- The start of a transaction.
- The end of a transaction.
- An intermediate function within a transaction consisting of a chain of three or more functions, or a function not within any transaction.

Function Definition Screen

The prompt given in the function definition screen is `Transaction?`. The screen appears as follows:

```
01/05/90      Function:        1-DEFINITION MIC/SYS/00


Function name:              Title:
 Type:               Transaction?:
Category:                 Analyst:
 Level:                Entry Date:
Exit Link:               Reset On:
Error Link:                    By:
```

The responses to the `Transaction?` prompt are shown in the table below.

Response to 'TRANSACTION?' prompt	Type of Function
S	Start of a chain of functions together comprising a transaction.
E	End of a chain of functions together comprising a transaction.
SE	Start and end of a transaction: function comprises a single transaction. (If the function is not one-time-only, all passes through the function are part of the one transaction).
null	Intermediate function within a chain of functions comprising a transaction ■ function not within any transaction.

Where a function is defined on the transaction boundary, the entire function is part of one transaction. If the function is both the start and end of the transaction, all file updates from the time the function is entered to the time the function is closed are part of one transaction. Where a function is defined as only the start of the transaction it must link to another function in that transaction.

Note: Having defined a transaction boundary at function level, you cannot then define transactions at block level within that function.

Block Level Transaction Boundaries

In this case, a transaction comprises one logical update, screen or report. When defining transaction boundaries at block level, you must not make any response to the `Transaction?` prompt on the function definition screen. Instead, you should define the transaction at the function characteristics screen.

Function Characteristics Screen

The prompt given in the function characteristics screen is '`Trans?`'. An update characteristics screen appears as follows:

```
01/05/90    Function:    2- CHARACTERISTICS MIC/SYS/00


Update#:-         Trans?:         Sort/Select?:
One-time?:    Start Upd#:          Descending?:
Logic ID:       End Upd#:          SSEL Lgc ID:
                  Paging?:         SSEL Efile#:
```

Note: If you have used the `Transaction?` prompt on the function definition screen to define a transaction boundary at function level, then your response to the `Trans?` prompt on the function characteristics screen must be N.

When transaction boundaries are defined at block level, each iteration through the block is one transaction. If you are using function level boundaries, transactions can cross functions, but block level transactions must start and end in the same logical block. If the block is one-time-only, then the block is equivalent to one transaction. Where a block is not one-time-only, there is a transaction for each primary record read in the block.

Non-Paging Screens

The transaction is opened before the first field is processed and is closed after the files are written. The default logic is executed outside the transaction, so any 'CHAIN' or 'LINK' statements in this logic are outside the transaction boundaries. All other logic and all nested screens are executed inside the transaction.

Paging Screens

As with other blocks, in a paging screen there is one transaction for each primary record read. These correspond to one or more lines on a paging screen, not to the entire screen.

In Add or Change mode, the transaction boundaries are the same as for non-paging screens, but in Insert or Delete mode an extra transaction is included. During Insert mode on paging files, all of the file items after the insert must be read and rewritten with a new sequence number. This re-sequencing is treated as a transaction in its own right. Similarly, in Delete mode, each record deleted is a transaction, with the transaction opened before the file reads and closed following the file writes. Once a block of records is deleted, all following records must be re-sequenced and this re-sequencing is treated as a separate transaction.

Random Paging Updates

If a paging file is accessed randomly in any type of block, the file is re-sequenced when records are deleted. In this case, the re-sequencing is included as part of the delete transaction, it is not a transaction in its own right.

Subfiles

Transaction boundaries on a subfile are meaningless because the subfile is not actually written until the master file is written. If a subfile is part of a transaction, the master file must be part of the same transaction.

Identifying Transactions

The name of the transaction can be supplied via the system variable @$TRVAR which can be specified by a string of up to 50 characters placed in logic thus:

@$TRVAR = "Invoice No. 552"

The contents of this variable are written to the Transaction-Log at the start and end of each transaction to enable the transactions to be easily identified. This information can be used in a number of ways: to identify transactions on a log tape, to provide audit trail information to identify complete and restored transactions or incomplete, rejected transactions.

Where to Set @$TRVAR

The Start Transaction command is issued before the first field in the transaction is processed. @$TRVAR can therefore be set in the start of function logic, to name a function level transaction or the first block-level transaction in a function. Subsequent block-level transactions can be given different names by re- setting @$TRVAR within the function logic.

Item Locks In ALL

ALL locks all items accessed unless in look-up mode. Transaction Logging maintains these locks until the end of a transaction. If a user's process is held up waiting for a lock the message 'Waiting for Lock' is output to inform the user of the reason for the delay.

Aborting Transactions

A transactions is aborted if:

- An attempt is made to start a new transaction before the current one has completed.
- The &$CANCEL.TR flag is enabled at the end of the Logical Screen/Report/Update.
- A TRANSABORT is performed from TCL or DATA/BASIC.

Logging Of Files

Whenever you create a file in ALL, you are asked "Should this file be transaction logged?". Enter Y(es) or N(o) as required.

The Chain Command

If the control is passed from one function to another via the CHAIN command the status of any on-going transaction is not affected. There is no automatic Abort issued and any Transaction End defined for the function is not issued after the CHAIN command has been carried out.

External Calls

Using the LINK command to pass control outside a function is not allowed unless an abort transaction has been issued by an ENABLE CANCEL.TR

Notes on Defining Transactions in ALL

1. Transactions cannot be defined on both function level and block level within one function.

2. Transactions cannot be nested.

3. Transactions must be kept as small as possible to avoid performance problems caused by holding item locks for longer than necessary.

4. The possibility of deadly embraces can be reduced by always processing the same set of files in the same order.

Appendix A
Error Messages

This appendix contains a list of error messages which may appear while running Transaction Logging and provides suggestions as to what to do when each message is displayed.

CTL1000 **`An unrecognised error has occurred`**

Call support.

CTL1001 **`Unable to open file 'file-name'`**

File name does not exist. Re-enter with command with correct file name.

CTL1002 **`Unable to read item 'item-id' from 'file-name'`**

File name does not exist. Re-enter with command with correct file name.

CTL1004 **`unable to create file 'file-name'`**

Call support.

CTL1005 **`Error 'code'`**

Call support.

CTL1006 **`System file 'file-name' exists`**

Use TL-CREATE-FILE again with the E option.

CTL1007 **`System file 'file-name' doesn't exists, don't use (E) option.`**

Use TL-CREATE-FILE again without the E option.

CTL1008 **`Operation incomplete, aborted by user`**

You have pressed the break key during TL-RESTORE. Re-execute TL-RESTORE

CTL1009 **`An invalid option has been requested`**

Re-enter command with correct option.

CTL2000 **`There is a transaction already active for this process`**

Issue a transaction end or abort before opening another transaction.

CTL2001 `There is no currently active transaction for this process`

Unless it is in response to a TRANSQUERY, it is caused by an applications programming error. Re-program with the necessary TRANSTART.

CTL2102 `Start logging operation failed with code 'code'`

Call support.

CTL2103 `Stop logging operation failed with code 'code'`

Call support

CTL2105 `Get file information on file 'file-name' failed`

Check to see if physical file is missing. If so, re-create UNIX file with the appropriate name.

CTL2106 `File 'filename' is not a log file`

You are trying to restore from a file which is not a clean log or does not exist. Use a valid clean log name.

CTL2107 `Transaction logging is unsupported`

Ensure that Transaction Logging is targetted on your system and the system and your database are configured correctly (using mklog) to support Transaction Logging. Refer to Chapter 4.

CTL2108 `Log file 'file-name' is not empty, use CLEAR-FILE`

Either use another log file which is empty or empty the log file using CLEAR-FILE.

CTL2109 `Logging is already active on 'log file name'`

Either continue logging to the current clean log or re-enter TL-START with a different empty log name.

CTL2111 `This file is the active log`

You cannot restore from an active clean log. Also, after switching clean logs the recovery file, from which you have just switched, remains active for 5 minutes.

CTL2112 `You can't do this when logging is in a state of switching/stopping`

Wait until switching/stopping is complete. This takes about 5 minutes.

CTL2113 `Logging is already inactive or in a state of becoming inactive`

You have already executed a TL-STOP.

CTL2114 `Logging state unknown - you have out of date transaction verbs`

Obtain the correct release of TL commands

CTL2115 `You can't do this when the secondary failsafe system is paused`

Use TL-CONTINUE to resynchronise failsafe before repeating operation.

CTL2150 `The restore operation has failed with code 'code' ('error message')`

Call support.

CTL2200 `This command must be run from the 'account-name' account`

Log to 'account-name' and re-run command.

CTL2201 `This command can only be run when transaction logging is active`

Re-run the command after executing a TL-STOP.

CTL2202 `This command cannot be run when transaction logging is active`

Re-run the command after executing a TL-START.

CTL2203 `This command cannot be run when you are inside a transaction`

Wait for transaction end or abort transaction, then retry.

CTL2204 `You must have account privilege level of 2 to execute this command`

Log to an account with the required privileges then re-try.

CTL2205 **`This command may only be used when no other users are logged on.`**

`There are in fact 'number' others users logged on,`

`Use LISTU to see logged on users.`

Either wait until all users are logged off or force all users off using LOGOFF. You can use INHIBIT-LOGONS to prevent more users from logging on.

CTL2206 **`You must specify a clean log file name with this command`**

Re-run the command with a valid clean log name.

CTL2207 **`You must run TL-SET-LOG-STATUS before you can start logging`**

You have not yet defined the logging status of the database

CTL2209 **`This command can only be run when transaction logging is active`**

Execute TL-START, then reenter command.

CTL2210 **`This command can only be run when clean logging is inactive`**

You cannot enter TL-START while logging is active.

CTL2211 **`This command can only be run on a primary or a standalone system`**

You cannot enter TL-START on the secondary database.

Appendix B
Installation of Transaction Handling/Logging

This appendix contains detailed examples of the procedures that you need to follow to install Transaction Handling and Logging on the UMAX V and M88 systems.

Introduction

The installation of transaction logging for a particular release involves

- ensuring that the disk is dedicated to logging. All swap partitions and file systems removed.
- creating raw log and clean log partitions.
- mounting the clean log file system.
- initialising the raw log.
- configuring the database with a clean log sub-directory.

> CAUTION
>
> For effective operation of transaction logging the raw and clean log partitions should be placed on their own disk. They must **not** reside on the same disk as any data bases or swap partitions.

This appendix contains detailed examples illustrating how to create and initialise the partitions for the raw log and clean log on both UMAX V and M88 systems.

The examples assumes that you have 8 disks, 0 to 7, and that disk 7 is to be set up for the sole use of raw and clean logs. In this example disk 7 is currently used for a file system called /user7 and also constitutes part of virtual partition vp0. File system /user7 will be removed and virtual partition vp0 will be redefined to exclude the physical partition on disk 7.

Notes:

1. This is only an example. Your system configuration may be different.
2. Remember to save the contents of /user7 and vp0 if you wish to keep them
3. The example contains some embedded comments, which are highlighted by the use of italics.

Removing Swap Partitions from Log Disk

The log disk must be dedicated to logging. If swap partitions are defined on the disk, they must be removed. The procedure is as follows:

1. Find out if there are any swap partitions on disk7

```
$ su
Password:
# cd /etc
# cat init.d/swap
USAGE="Usage:/etc/init.d/swap (start | stop)"

if [ ! -d /usr/bin ]
then          # /usr/not mounted ??
        exit
fi

case "$1" in
'start')
        # Add swap area described here
        /etc/swap -a /dev/dsk/1s1 0 131820
        /etc/swap -a /dev/dsk/7sl 0 131820              Yes there is

        ;;

'stop')
        # Don't bother deleting swap areas
        ;;
*)
        echo ${USAGE}
        exit 1
        ;;
esac
#
# swap -d /dev/dsk/7s1 0                                So remove it
# swap -l
path           dev   swaplo   blocks   free
/dev/dsk/0s1   0,1       0     196608   177880
/dev/dsk/1s1   0,17      0     131816   113288
/dev/dsk/7s1   0,23      0     131816   113288
#
```

2. Repeat 'swap -l' until swap partition on 7s1 is disabled. This is indicated by it no longer appearing on the swap -l output. On an idle system this may not take very long.

```
# swap -l
path           dev   swaplo   blocks   free
/dev/dsk/0s1   0,1       0     196608   177880
/dev/dsk/1s1   0,17      0     131816   113288
#
#
# vi init.d/swap
```

3. Remove the line "`/etc/swap -a /dev/dsk/7s1 0 131820`" to prevent this partition being used as swap space again

Removing File Systems from Log Disk

The log disk must also be cleared of any virtual partitions. The procedure is, as follows:

1. Find out if there are any virtual partitions which use disk7

```
#
# cat vptab

/dev/rdsk/vp0 16/dev/rdsk/2s5 /dev/rdsk/3s5 /dev/rdsk/7s5
```

Yes there is

2. Find out it there are any filing systems using disk 7

```
# df
/          (/dev/dsk/0s0 ):    458 blocks    4219 i-nodes
/tmp       (/dev/dsk/1s4 ):  41874 blocks    8082 i-nodes
/usr       (/dev/dsk/1s5 ):  11702 blocks   14615 i-nodes
/usr/tmp   (/dev/dsk/1s5 ):  42712 blocks    8188 i-nodes
/user0     (/dev/dsk/0s6 ):  31906 blocks   50165 i-nodes
/user1     (/dev/dsk/1s6 ): 155162 blocks   52868 i-nodes
/user2     (/dev/dsk/2s4 ):  73978 blocks   53109 i-nodes
/user3     (/dev/dsk/3s4 ): 115594 blocks   57361 i-nodes
/user4     (/dev/dsk/4s4 ): 107478 blocks   58855 i-nodes
/user4a    (/dev/dsk/4s6 ):  76868 blocks   47520 i-nodes
/logs      (/dev/dsk/4s9 ):   8024 blocks   24569 i-nodes
/user5     (/dev/dsk/5s4 ): 225356 blocks   43550 i-nodes
/user6     (/dev/dsk/6s4 ): 160824 blocks   42053 i-nodes
/user7     (/dev/dsk/7s4 ):  23680 blocks   58334 i-nodes Yes there is
/usr/cora  (/dev/dsk/vp0 ): 120434 blocks   53853 i-nodes Yes there is
```

3. Unmount the filing systems.

```
#umount /user7
#umount /usr/cora
#
```

4. Disable the offending virtual partition

```
# vpadmin -d /dev/rdsk/vp0
# vpadmin
   Virtual     Inter No.
  Partition    leave Prt       Size   Real Partitions
/dev/rdsk/vp0     16   3    1759680   /dev/rdsk/2s5  /dev/rdsk/3s5
                                            /dev/rdsk/7s5
```

5. Repeat the 'vpadmin' command until virtual partition vp0 is disabled. On an idle system this may not take very long.

```
#
# vi vptab
```

6. Remove the string /dev/rdsk/7s5 to exclude disk 7 from the definition of virtual partition vp0. The line becomes:

```
/dev/rdsk/vp0 16 /dev/rdsk/2s5 /dev/rdsk/3s5
```

7. Remove file system name /user7 from fstab, as its virtual partition is to be obliterated. File system /usr/cora on virtual partition vp0 is ok because the virtual partition still exists, we've just reduced it's size

vi fstab

Remove line `"/dev/dsk/7s4 /user7 BSD"`

8. Remove mount point of the file system, the definition of which has just been removed.

```
# rmdir /user7
```

Defining the Raw Log and Clean Log Partitions

1. Check disk 7 for partitions which may be in use but which aren't used for an automatically mounted file system.

   Note: Partitions 2 and 3 are always defined, 15 will usually be defined. So we expect partitions 4 and 5 to be the only additional partitions. If there are any others you must determine their purpose and ensure that they aren't used in the future.

```
# partdisk /dev/rdsk/7s3
Enter 'initialize', 'edit', '?' or 'quit' [e]: e

              Current Partition Layout

Partition    Offset    Size         Type     Name

    2          0   1173930     All           all
    3          0       780     Header        header
   15    1172340      1590     Diagnostic    diagnostic
```

Partition 2, 'all', defines the whole accessible disk, all other partitions map onto some part of partition 2. Partition 3 occupies the first 780 blocks, partition 15 occupies the last 1590. This leaves us 1171560 blocks to define the raw and clean log partitions.

```
Enter 'add', 'delete', 'rename', 'copy', 'view', 'geom', 'quit',
'?' [?]: add
Enter partition number: 4
Enter partition name: rawlog
Enter size of partition in sectors: 204800
205140 sectors makes partition size a multiple of cylinders
do you wish to use 205140 sectors instead?: yes
First unallocated space of this size at sector number 780.
Enter sector number of partition offset: 780
Is this a diagnostic partition?: no
Enter 'add', 'delete', 'rename' 'copy', 'view', 'geom', or'quit':
add
Enter partition number: 5
Enter partition name: clogs
Enter size of partition in sectors: 966420
966420 sectors makes partition size a multiple of cylinders
Do you wish to use 966420 sectors instead?: yes
first unallocated space of this size at sector number 205920
Enter sector number of partition offset: 205920

Is this a diagnostic partition?: no
```

```
Enter 'add', 'delete', 'rename', 'copy', 'view', 'geom', 'quit',
'?' [?]: view

                Current Partition Layout

Partition    Offset    Size         Type     Name

    2          0   1173930          All     all
    3          0       780       Header     header
    4        780    205140     Standard      rawlog
    5     205920    966420     Standard      clogs
   15    1172340      1590     Diagnostic    diagnostic

Enter 'add', 'delete', 'rename', 'copy', 'view', 'geom', 'quit',
'?' [?]: quit
Enter 'y' to save changes: y
```

This completes the partitioning of the transaction logging disk.

2. Re-enable the virtual partition vp0

```
#
# vpadmin -e /dev/rdsk/vp0
# vpadmin
```

Repeat vpadmin command until the virtual partition is shown enabled. On an idle system this may not take very long.

```
# vpadmin
      Virtual     Inter No.
    Partition     leave Prt      Size   Real Partitions
  /dev/rdsk/vp0     16   3    1173120  /dev/rdsk/2s5 /dev/rdsk/3s5
```

Creating the Clean Log File System

1. Update fstab so that the new clogs file system is automatically mounted at system initialisation.

```
# vi fstab
```

Add line to define the clogs filing system:

```
/dev/dsk/7s5 /clogs BSD
```

2. Make the clean log file system and remake the file system on virtual partition vp0. Then mount them.

```
#
# bsdmkfs /dev/dsk/7s5
# bsdmkfs /dev/dsk/vp0
#
# mkdir /clogs                          Defines the mount point for clogs file system
# chmod +rw /clogs          Allows all users read/write access to the clean log directory
# chmod +rw /dev/rdsk/7s4               Allows all users access to raw log partition
# mount /usr/cora
# mount /clogs
```

Initialising the Raw Log

Lastly initialise the raw log for the release, as follows:

> **CAUTION**
>
> Ensure that you identify the correct partition, otherwise a valid file system may be corrupted.

```
# REALROOT=/usr/realman/3.1X                X being the rev number
# export REALROOT
# cd $REALROOT/bin
# ./mklog -r /dev/rdsk/7s4 $REALROOT/bin    7s4 is what we defined with
# exit                                      partdisk above for the rawlog
$
```

Configuring a database for logging

After a database has been created, it may be configured to use transaction handling/logging with the following command:

```
$ mklog <clean_log_directory> <data_base_path> { -c subdir }
```

For example:

```
$ mklog /clog /usr/jones/dbase1
```

Creates a clean log sub-directory 'dbase1' in clean log directory '/clog' and updates the database 'dbase1' configuration file to reference this sub-directory.

To use a different name for the clean log sub-directory, the '-c' option should be used:

For example:

```
$ mklog /clog /usr/jones/dbase -c jones_clog
```

This does the same as the previous example, but instead names the clean log sub-directory 'jones_clog'.

Once a database has been configured, the system manager can log onto the database and enable logging using the TCL commands:

:TL-CREATE-FILE *cleanlog-name*
:TL-START *cleanlog-name*

Procedure for M88 Systems

This example is for one of the larger deskside M88 machines. On the smaller desktop machines, the disk controller is on the motherboard and so the disk partitions will typically be named /dev/dsk/m187_000s7

1. The log disk must be dedicated to logging. If swap partitions are defined on the disk, they must be removed. The procedure is as follows:

2. Find out if there are any swap partitions on disk 7

```
$su
Password:
# swap -l
path               dev swaplo blocks free
/dev/dsk/m328_000s ll6,1     1 255992 255992
/dev/dsk/m328_007s ll6,16    0 32768   32768
```

Yes there is.

```
# swap -d /dev/dsk/m328_007s1 0
```

To remove it .

3. Repeat 'swap -l' until swap partition on 7sl is disabled, ie. until it no longer appears on the 'swap -l' output. On an idle system this should not take very long.

```
# swap -l
path               dev swaplo blocks free
/dev/dsk/m328_000sl ll6,l      1 255992 255992
```

4. Edit the /etc/init.d/rc2 file to ensure this partition is not enabled as swap again

```
# cd /etc/init.d
# vi rc2

#    "Run Commands" executed when the system is changing to init
state2,
#     traditionally called "multi-user".

umask 022
. /etc/TIMEZONE
```

```
#    Pickup start-up packages for mounts, daemons, services, etc.
set 'who -r'
if [ $9 = "S" ]
then
  stty sane tab3 2>/dev/null
        echo 'The system is coming up.  Please wait.'
        BOOT=yes
        if [ -f /etc/rc.d/PRESERVE ]    # historical segment for
vi and ex
        then
                mv /etc/rc.d/PRESERVE/etc/init.d
                ln /etc/init.d/PRESERVE/etc/rc2.d/SO2PRESERVE
        fi

elif [ $7 = "2" ]
then

    echo 'Changing to state 2.'
    if [ -d /etc/rc2.d ]
    then
            for f in /etc/rc2.d/K*
            {
                    if [ -s ${f} ]
                    then
                            /bin/sh ${f} stop
                    fi
            }
    fi
fi

if [ -d /etc/rc2.d ]
then
        for f in /etc/rc2.d/S*
        {
                 if [ -s ${f} ]
                 then
                         /bin/sh ${f} start
                 fi
        }
fi
```

```
if [ "${BOOT}"="yes" ]
then
       stty sane 2>/dev/null
fi

if [ "${BOOT}"="yes" -a -d /etc/rc.d ]
then
       for f in 'ls /etc/rc.d'
       {
                if [ ! -s /etc/init.d/${f} ]
                then
                          /bin/sh /etc/rc.d/${f}
                fi
       }
fi

if [ "${BOOT}"="yes" -a $7="2" ]
then
       echo 'The system is ready.'
elif [ $7="2" ]
then
       echo 'Change to state 2 has been completed.'
fi
/etc/swap -a /dev/dsk/m328_001s0 0 32768
/etc/swap -a /dev/dsk/m328_007s1 0 32768
```

5. Remove the line '/etc/swap -a /dev/dsk/m328-007sl 032768' to prevent this partition being used for swap area again.

Removing File Systems from Log Disk

The log disk must also be cleared of any virtual partitions. The procedure is as follows:

1. Find out if there are any virtual partitions which use disk 7, by looking in the /etc/vdsk.conf file.

```
$ cat /etc/vdsk.conf
#
#      NAME
#     vdsk.conf - Virtual Disk Configuration file
#
#      DESCRIPTION
#    Each line defines a specified Virtual Device.
#    The first device name is the Virtual Device configured with
#    white space separated list of the physical devices specified
#    after the virtual device name.
#    The optional size of the physical device can be specified
#    after the physical device name separated by a colon.
#    Each Virtual Device configuration is separated
#    from the next by a newline.
#    Lines can be continued by the backslash character before
#    the newline character.
#
#     EXAMPLE
#    /dev/dsk/vdsk0    /dev/dsk/m328_000sl:10000 /d v/dsk/m328_001sl:10000
#    /dev/dsk/vdsk1    /dev/dsk/m328_100sl /dev/dsk/m328_101sl \
#                /dev/dsk/m328_110sl /dev/dsk/m328_lllsl
#
/dev/dsk/vdsk0 /dev/dsk/m328_00s4 /dev/dsk/m328_007s2 Yes there is
```

2. Find out if there are any filing systems using this virtual partition.

```
# df
/         (/dev/dsk/m328_000s0):     25080 blocks      5962 i-nodes
/usr      (/dev/usr        ):        200596 blocks     53317 i-nodes
/real     (/dev/dsk/m328 000s3):     132428 blocks     37270 i-nodes
/user7    (/dev/dsk/7s4):            23680 blocks      58334 i-nodes
/usr/cora (/dev/dsk/vdsk0:           1867608 blocks    65190 i-nodes   Yes
there is
```

CAUTION

The following operations will obliterate /user7 and reduce the size of /usr/cora (and initialise it), if necessary, save these filestores before continuing.

3. Unmount the filing systems

```
# umount /user7
# umount /usr/cora
```

4. Edit the vdsk.conf file to remove the partition on disk7 from the virtual partition.

```
# vi /etc/vdsk.conf
```

5. Change the line

```
/dev/dsk/vdsk0  /dev/dsk/m328-000s4  /dev/dsk/m328-007s2
```

to

```
/dev/dsk/vdsk0     /dev/dsk/m328-000s4
```

6. Remove the file system name /user7 from /etc/fstab because we are going to obliterate it. File system /usr/cora on virtual partition vdsk0 is ok because the virtual partition still exists, we've just reduced it's size.

```
# vi  /etc/fstab

/dev/dsk/m328_000s3     /real
/dev/dsk/vdsk0          /usr/cora
/dev/dsk/m328_007s4     /user7
```

7. Remove the line **/dev/dsk/m328_007s4 /user7**

8. Remove the mount point of the file system

```
# rmdir /user7
```

9. Check all partitions defined for disk 7 and ensure they are freed off in one of the above ways before repartitioning the disk.

Note: Partition 7 is used to access the whole disk and will always be defined.

Defining the Raw Log and Clean Log partitions

Repartition disk 7 using msledit to define a rawlog and a cleanlog partition.

The following is an example of a msledit session.

```
# msledit /dev/rdsk/m328_007s7

slice  offset  sl size fs size fsname  vol-id  info
0      648     32768   0                       1h8
1      33416   1994149 1994148                 1h8
2      0       0       0                       1h8
3      0       0       0                       1h8
4      0       0       0                       1h8
5      0       0       0                       1h8
6      0       0       0                       1h8
8      0       0       0                       1h8
9      0       0       0                       1h8
10     0       0       0                       1h8
11     0       0       0                       1h8
12     0       0       0                       1h8
13     0       0       0                       1h8
14     0       0       0                       1h8
15     0       0       0                       1h8
7      0       2027565 0                       1h8

slice 0>                   offset: 648
slice 0>                slice size: 32768   1800000
slice 0>           filesystem size: 1800000
slice 0>           filesystem name: swap    clean
slice 0>               vol-id name:         R32
slice 0>    filesystem information: 1h8

slice 1>                   offset: 1800648
slice 1>                slice size: 1994149  225916
slice 1>           filesystem size: 0
slice 1>           filesystem name: /user0   raw
slice 1>               vol-id name:          R32
slice 1>    filesystem information: 1h8

slice 2>                   offset: 0 w

'/dev/rdsk/m328_00s7' written

slice 2>                slice size: 0        q
```

Enter return to keep this value.

Enter size for clean log partition.

Calculate new offset

Enter raw log size.

Enter 0 to inhibit the file system build.

Enter to write the new config away

msledit will now automatically rebuild any filesystems in this case, quit.

```
mkfslk:  /dev/rdsk/m328_007s
(DEL if wrong)
bytes per logical block=1024
total logical blocks=41020
total inodes=10240
Space reservation: 10% (4102 logical blocks)
cluster size=8
mkfslk: Available blocks=40377
```

This completes the partitioning of the transaction logging disk.

Update *fstab* so that the new clean log filing system is automatically mounted when the system is booted.

Note: when using **msledit** it is your responsibility to calculate the correct offsets. **msledit** performs no validation to ensure that partitions do not overlap.

New offset = Previous offset + Previous slice size.

e.g. In the previous example the new offset for slice 1

= Previous offset (slice 0) + Previous slice size (slice 0)

= 648 + 1800000

= 1800648

After Image Defines the item update and is used to recover the updated item in the event of a system/database failure.

ALL Application Language Liberator.

Before Image Defines how the updated item is restored to its original value and is used to 'roll-back' the associated update to its original value, if the system/database fails in mid-transaction.

Clean Log A file containing a log of changed items, other updates, transaction start/end and other records.

Commit Permanently update the database with updates made during a transaction. (At any point prior to commitment, all updates belonging to the transaction may be undone.)

Deadly Embrace A condition which arises when two or more processes active at the same time become suspended while competing to lock the same set of items or other resources.

Dirty Read A situation where transaction T1 updates an item which is then read by transaction T2, and T1 aborts, causing all its updates to be undone. T2 will have read a non-existent record.

'Hard' System Failure Any hardware or software fault which causes the database to become corrupted.

Hit Process A process which terminates abnormally (for example, crashes) or is killed.

Image A set of information which collectively defines an operation for an application. For example, an item update is logged as an 'After image'. this may be passed back to Reality X to perform the associated update and restore it on the database. The structure of the image is such that it can be transferred within Reality X without the contents needing to be known.

Item Locking A mechanism to prevent multiple processes attempting to access the same item at the same time.

Logging The process which takes data relating to changes to the database and writes them to a clean log.

Lost Update A situation where transaction T1 updates an item which has previously been updated, but not committed, by transaction T2, and T2 is aborted. The update performed by T1 is then also lost.

Primary Database The active database in a FailSafe pair which is currently logged on to by users.

Raw Log A central repository in a raw disk partition which holds the recently logged images of updates from all databases on the system. Images are held in a circular queue until their transaction has been committed, after which the 'After' images are transferred to a clean log.

Rolled Back All updates since the start of a transaction are deleted by restoring the 'before' image to the database, maintaining it in a consistent and predictable state.

Secondary Database A database in a FailSafe pair which currently operates as the standby. It cannot be logged on to.

Transaction A group of updates or other changes to the database that are interrelated such that if one update is committed then all updates within the group should also be committed in order to maintain a consistent database.

Unrepeatable Read A situation where transaction T1 reads an item which is then updated and committed by transaction T2. T1 then re-reads:definition the same item and sees two different committed values.

TIPH Process Terminal Independent Process Handler process - one which does not have a terminal associated with it.