jBASE Transaction Journaling

Version 4.1

Contents

Documentation Conventions

This manual uses the following conventions:

Convention	Usage
BOLD	In syntax, bold indicates commands, function names, and options. In text, bold indicates keys to press, function names, menu selections, and MS-DOS commands.
UPPERCASE	In syntax, uppercase indicates JBASE commands, keywords, and options; BASIC statements and functions; and SQL statements and keywords. In text, uppercase also indicates JBASE identifiers such as filenames, account names, schema names, and Windows NT filenames and pathnames.
UPPERCASE *Italic*	In syntax, italic indicates information that you supply. In text, italic also indicates UNIX commands and options, filenames, and pathnames.
Courier	Courier indicates examples of source code and system output.
Courier Bold	**Courier Bold** In examples, courier bold indicates characters that the user types or keys (for example, <Return>).
[]	Brackets enclose optional items. Do not type the brackets unless indicated.
{}	Braces enclose nonoptional items from which you must select at least one. Do not type the braces.
ItemA ¦ itemB	A vertical bar separating items indicates that you can choose only one item. Do not type the vertical bar.
. . .	Three periods indicate that more of the same type of item can optionally follow.
⇒	A right arrow between menu options indicates you should choose each option in sequence. For example, “Choose **File ⇒Exit**” means you should choose **File** from the menu bar, and then choose **Exit** from the File pull-down menu.

Syntax definitions and examples are indented for ease in reading.

All punctuation marks included in the syntax—for example, commas, parentheses, or quotation marks—are required unless otherwise indicated.

Syntax lines that do not fit on one line in this manual are continued on subsequent lines. The continuation lines are indented. When entering syntax, type the entire syntax entry, including the continuation lines, on the same input line.

Preface

Transaction Journaling

Transaction journaling provides the capability to log updates which are made to a jBASE database. The order in which the updates are logged will reflect the actual updates made by a particular user/program facility in a sequential manner. Updates made system-wide will be appended to the log as and when they occur on the database; i.e. the transaction log will reflect all updates in sequential order over time. The intention of the transaction log is that it provides a log of updates available for database recovery in the event the system uncontrollably fails.

These are the main transaction journaling administration utilities provided within jBASE:

jlogadmin	This command is provided to configure and start/stop/suspend transaction journaling.
jlogstatus	This command allows the administrator to monitor the activity of transaction journaling.
jlogdup	This command enables the recovery or replication of data.

Additional Administration Utilities

Jlogsync	synchronizes and flushes log files
Jlogmonitor	monitors the current state of transaction journaling

Transaction Journaling Strategy

This is the minimum setup for Transaction Journaling. The strategy to be employed will be as follows:
There will be 2 sets of transaction log files on each machine, logset1 and logset2. Logset1 will contain all the updates applied on Monday or Wednesday or Friday and logset2 will contain all the updates applied on Tuesday, Thursday or Saturday/Sunday.
The definitions of these files are maintained by the ‘jlogadmin’ command. The transaction log files should be switched by use of cron at midnight (or 1 minute past midnight) using the command ‘jlogadmin –l N’ command where N is 1 for Monday , Wednesday or Friday and N is 2 for Tuesday , Thursday and Saturday.
The administrator must ensure that all users are logged off the system prior to a system backup.
Transaction journaling is stopped by stop_tj command.
The backup script ‘backup_jbase’ should run to backup the system. This scenario allows for the backup failing and being restarted. Note the creation of a statistics file. This is used effectively to timestamp the transaction log with the start time of the backup. Thus if the save was restarted then the creation time of the statistics file will reflect the start of the last good backup. The operation is thus:
Stop the transaction log file to tape jlogdup process: database updates for the duration of the backup will be prevented by the administrator.
Remove and label the tape – this contains all database updates since just prior to the last backup.
Mount a tape in the tape deck to hold the backup.

Once this has been done, the operator responds to the prompt and the backup commences.
Upon completion of the backup and verify, the tape is removed and labeled appropriately.
A new tape to hold the transaction log file is then mounted in the tape deck.
The operator now responds to the prompt and the jlogdup process, dumping updates from the disk-based transaction log file to tape re-commences.
There is no need to switch the transaction log files after the completion of the backup, as this is performed automatically.
See start_tj and backup_jbase scripts.

Logging Strategy

When journaling is running the basic order of operations for updates is as follows:

- The database item is updated in memory
- The transaction log is updated in memory.
- The transaction log is flushed every 10 seconds by default. However this time period can be configured via an option on the administration command, jlogadmin. It is also possible to configure an independent process to execute the jlogsync command, to ensure the transaction log is continuously flushed from memory at the specified interval, thus alleviating the flush procedure from all of the update processes.
- The database updates are flushed to disk by the operating system.
- The database update to disk and the log update to disk can be forced to be an atomic operation.
- Corruption of the transaction log can occur during a system failure while flushing the transaction log; this is impossible to circumvent. However, corruption occurs only at the end of the transaction log file, therefore it is possible to recover right up to the point of the system failure.
- There is still the possibility that when the system crashes, the disks containing the data AND the disks with the transaction log data will be lost. By running, a continuous jlogdup to a secondary machine or tape device you can protect against this highly improbable scenario.
- To use jBASE Transaction Journaling, you must install an additional license key.

Transaction Log

Access to the transaction log is via a special file. Use the CREATE-FILE command to create the file stub:

CREATE-FILE TJ1 TYPE=TJLOG
[417] File TJ1]D created , type = J4
[417] File TJ1 created , type = TJLOG

This creates an entry in the current directory
TJ1
JBC__SOB jBASE_TJ_INIT SET: set=current terminate=eos

When a file of type TJLOG is created, it generates a set of dictionary records in the dictionary section of the TJLOG file, which is a normal j4 hash file. The data section of a TJLOG file is handled by a special JEDI driver which accesses the current log set. The log set can be changed by additional parameters when creating the TJLOG file after the TYPE specification.

EXAMPLE

CREATE-FILE TJ2 TYPE=TJLOG set=eldest
Transaction Log File Layout

The transaction log files contain the following information

Attribute	Name	Description
1	SET	Log Set
2	FILENO	File Number
3	OFFSET	File Offset
4	LOGSIZE	Total Log Record Size
5	TYPE	Log Record Type
6	TIME-UTC	UTC Time
6	TIME	Update Time
6	DATE	Update Date
7	TRANS	Trans
8	TYPENUM	Log Record Type
9	PID	Update Process
10	PORT	Update Port
11	ERR	Error Description
12	TRANSID	Transaction Identifier

21	PATH	Full file path name
22	RECKEY	Update Record Key
23	JBNAME	jBASE Login Name
24	OSNAME	Platform Login Name
25	TTY	Terminal Name
26	APPID	Application Identifier
	1	Default Macro will list TYPE JBNAME PATH TIME DATE
	ALL	Macro will list all fields
	ERRORS	Macro will list TYPE JBNAME PATH ERR

TRANSACTION LOG FILE LAYOUT

The following record types are used in the transaction log (see dictionary item TYPE).

Type	Description
EOF	end of file
WRITE	record Written
DELETE	record Deleted
CLEARFILE	file Cleared
DELETEFILE	file Deleted
CREATEFILE	file Created
TRANSTART	transaction Started
TRANSEND	transaction Committed
TRANSABORT	transaction Aborted

The jlogdup command enables selective restores to be performed by preceding the jlogdup command with a select list. The select list can be generated from the log set by generating a special file type, which uses the current log set as the data file.

EXAMPLE

```
CREATE-FILE TJFILE TYPE=TJLOG
[ 417 ] File TJFILE]D created , type = J4
[ 417 ] File TJFILE created , type = TJLOG
:SELECT TJFILE WITH PATH EQ "/home/jdata/CUSTOMER" AND WITH TYPE NE "CLEARFILE"

167 Records selected
>jlogdup INPUT set=current OUTPUT set=database
```

In this example, all updates to the CUSTOMER file, which have been logged, except for any CLEARFILEs, are re-applied to the CUSTOMER file.

The jlogmonitor command can be used to monitor potential problems with the jlogdup process. It will report errors when specific trigger events occur. jlogmonitor can be run in the foreground but will usually be run as a background process (using the standard –Jb option).

Switching Logsets

A logset consists of 1 to 16 files. Each file has a capacity of 2GB, so the maximum capacity of a logset is 32GB. Before the logset reaches its capacity, a switch must be made to another logset using the jlogadmin command. Failure to do so will render journaling inoperable and may result in database updates from jBASE programs failing.

Using 16 files in a logset does not consume more space than using just 1 file. This is because updates to the logset are striped across all the files in the logset. When journaling is active on a live system, the recommendation is to define 16 files for each logset.

At least 2 logsets must be configured (with jlogadmin) so that when the active logset nears capacity, a switch can be made to another logset. Switching to a logset causes that logset to be initialized, i.e. all files in that logset are cleared. The logset that is switched from remains intact. The usual command to switch logsets is jlogadmin -l next. If there are 4 logsets defined, this command works as follows:

Active logset before switch	Active logset after switch
1	2
2	3
3	4
4	1

If a jlogdup process is running in real time to replicate to another machine, it should automatically start reading the next logset when it reaches the end of the current logset. To effect this behavior, use the parameter terminate=wait in the input specification of the jlogdup command.

Selective Journaling

The jBASE journaler does not record every update that occurs on the system. It is important to understand what is and is not automatically recorded in the transaction log.
What is journaled? Unless a file is designated unloggable, everything is updated through the jEDI interface (i.e. use of the jchmod –L filename command). This includes non-jBASE hash files such as directories.
What is NOT journaled? The opposite of above, in particular:

- Operations using non-jBASE commands such as the ‘rm’ and ‘cp’ commands, the ‘vi’ editor.
- The UNIX spooler files.
- Index definitions.
- Trigger definitions.
- Remote files using jRFS via remote Q-pointers or stub files
- When a SUBROUTINE is cataloged, the resulting shared library is not logged.
- When a PROGRAM is cataloged the resulting binary executable file is not logged.
- Internal files used by jBASE such as jPMLWorkFile, jBASEWORK, jutil_ctrl will be set to non-logged only when they are automatically created by jBASE. If you create any of these files yourself, you should specify that they be not logged (see note on CREATE-FILE below). You may also choose to not log specific application files.

It is recommended that most application files be enabled for transaction journaling. Exceptions to this may include temporary scratch files and work files used by an application. Files can be disabled from journaling by specifying LOG=FALSE with the CREATE-FILE command or by using the -L option with the jchmod command. Journaling on a directory can also be disabled with the jchmod command. When this is done, a file called .jbase_header is created in the directory to hold the information.

Remote files are disabled for journaling by default. Individual remote files can be enabled for journaling by using QL instead of Q in attribute 1 of the Q pointer, e.g.

<1>QL
<2>REMOTEDATA
<3>CUSTOMERS
Adding L to attribute 2 of the file stub

EXAMPLE

```
JBC_SOB JediInitREMOTE CUSTOMERS darthv.jbaseintl.com
```

In general, journaling on specific files should not be disabled for "efficiency" reasons as such measures will backfire when you can least afford it.

Selective Restores

There may be times when a selective restore is preferable to a full restore. This cannot be automated and must be judged on its merits.
For example, assume you accidentally deleted it a file called CUSTOMERS. In this case you would probably want to log users off while it is restored, while certain other files may not require this measure. The mechanism to restore the CUSTOMERS file would be to selectively restore the image taken by a jbackup and then restore the updates to the file from the logger journal. For example:

```
# jrestore –h ‘/JBASEDATA/PROD/CUSTOMERS*’ < /dev/rmt/0
# cd /tmp
# create-file TEMPFILE TYPE=TJLOG set=current terminate=eos
[ 417 ] File TEMPFILE]D created , type = J4
[ 417 ] File TEMPFILE created , type = TJLOG
# SELECT TEMPFILE WITH PATH EQ \/JBASE_ACCOUNTS/PROD/CUSTOMERS]\
21 Records Selected
```

jlogdup input set=current output set=database

If required, use the jlogdup rename and renamefile options to restore the data to another file.

NOTE: In order to preserve the chronological ordering of the records do not use a SSELECT command on the time field. This may not produce the correct ordering (multiple entries can occur during the same time period – the granularity being one second).

Failure Conditions and Recovery Remedies

Failures may occur in operations at the following stages:

- Normal live running:
- Database updates to the disk-based transaction log file
- jlogadmin running, dumping from this transaction log file to tape

What is the nature of the failure?

- System corrupted – rebuild necessary. This failure could be a disk failure; nothing on disk can be relied upon. In this case a full system restore is required, using the last successful back set. Following this the transaction log file needs to be rolled forward from the saved transaction log tape.
- Tape device/tape failure. In the event of a tape device failure – the device has to be repaired/replaced. The tape should be replaced. For this case and the tape failure, the disk-based transaction log file is still valid. The start time of the last execution of the jlogdup to tape operation was saved automatically by either the start_tj or backup_jbase script.
- The recover_jbase should be run in either of the cases above.

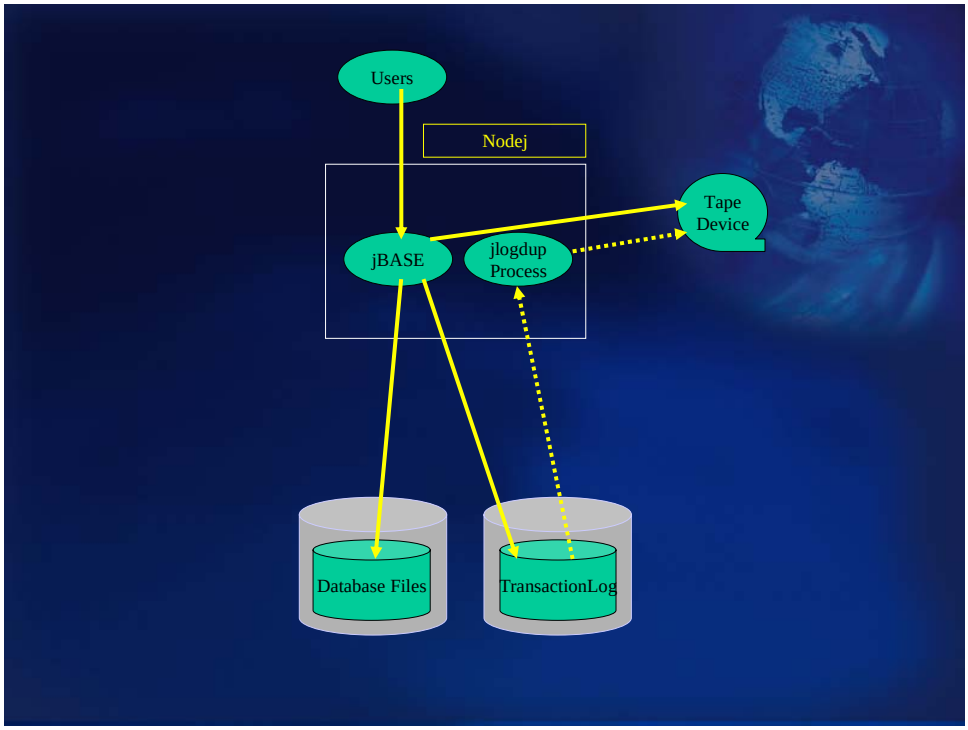
During the backup/verify procedure

- In this instance, the backup can be restarted. The act of restarting will update the transaction log file with a new start time (i.e. stat-file).
- During the dump of transaction log file information created during the backup/verify
- Problem with the tape: run recover_jbase after replacing the tape.

System/disk problem

The backup verified, so this is the backup set to be used for recovery by the recover_jbase script.
NOTE: that the jlogdup process to tape is still valid. Those transactions which have been dumped to tape can still be recovered.

Transaction Journaling on a single system

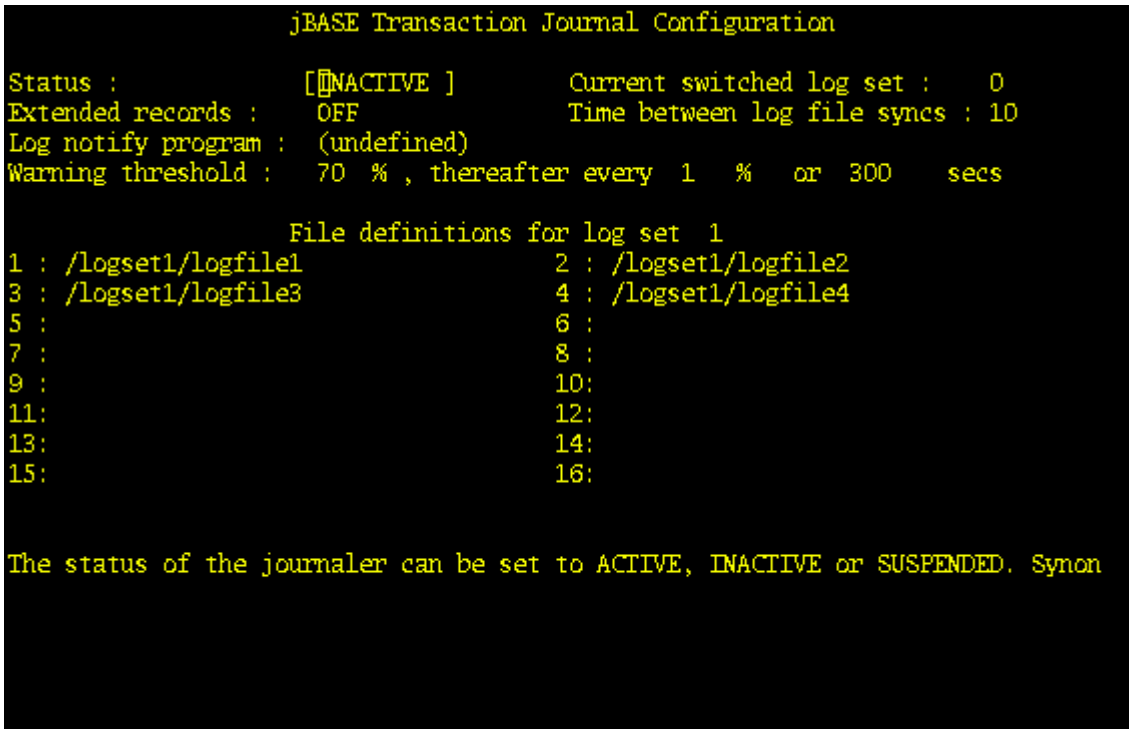


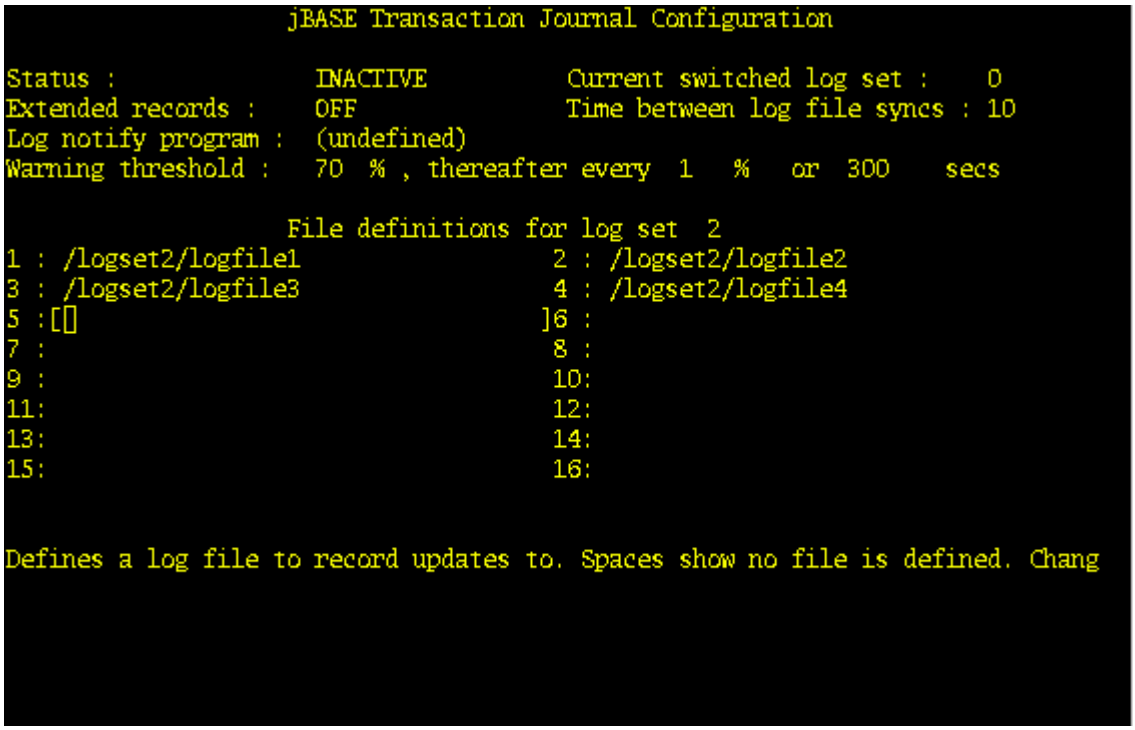
The diagram above represents the use of Transaction Journaling on a stand-alone system. Why would we use this kind of setup? In the event of system failure, the vast majority of processing which has taken place since the last system backup can be recovered and replayed. This is vital for those situations where the transaction cannot physically be repeated. The majority of these transactions can be replayed during system recovery when TJ is utilized.

Journal Configuration

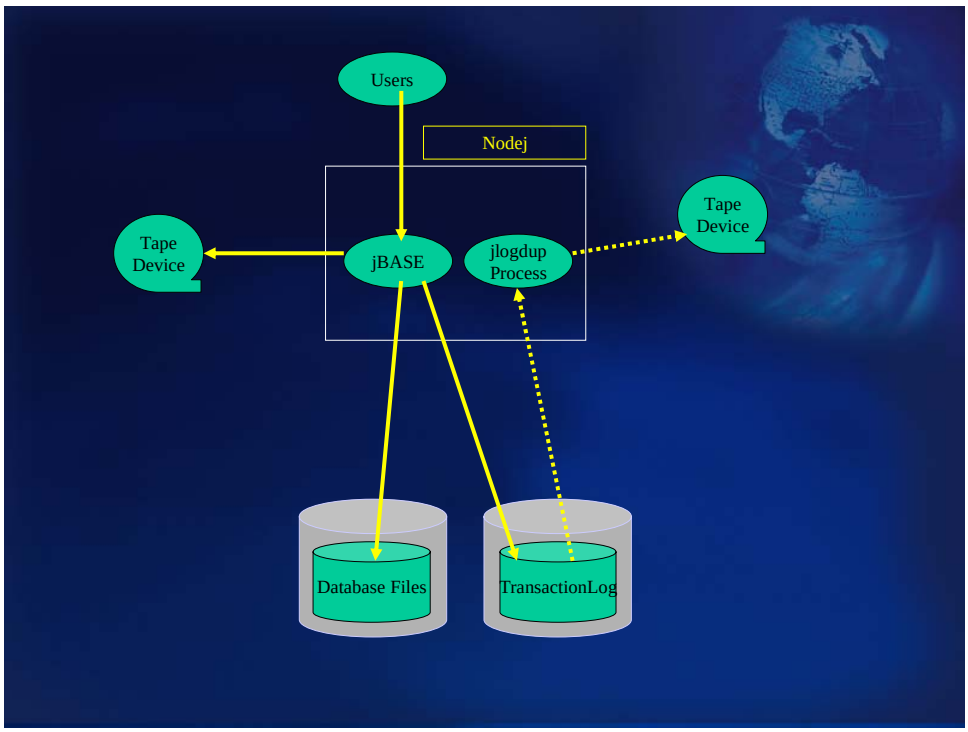
The Transaction Journal will be configured with two logsets; logset1 and logset2. Each of these logsets will occupy a separate volume partition on disk; this will allow for correct size monitoring of the logsets. The statistics of the logset usage indicated by the jlogstatus command is not at obvious at first glance. What is displayed is the proportion of the volume that has been used. Naturally, if the volume is shared by one or more logsets and/or other data files, then the percentage full will not necessarily reflect the percentage of the volume used by the transaction log. If the logset is contained within its own volume, then the figures are a fairer reflection of the TJ logset usage (albeit with certain storage overhead being present). Correct automatic invocation of the Log Nofity program relies on the accuracy of the percentages obtained.
Also if the logsets share a volume with other data, there is the possibility that the writing to the transaction log file may abort due to lack of space within the volume. The logset volumes should be created large enough for the expected population of updates between logset switches: i.e. if the logsets are switched every night at midnight, then the logset volume should be large enough to contain all the updates for a whole day (plus contingency).

Thus the logsets are created as below:





Transaction Journaling on a single system with two tape desks



The schematic shows the same system, except this time equipped with two tape decks. The advantages of this configuration over the previous are as follows:

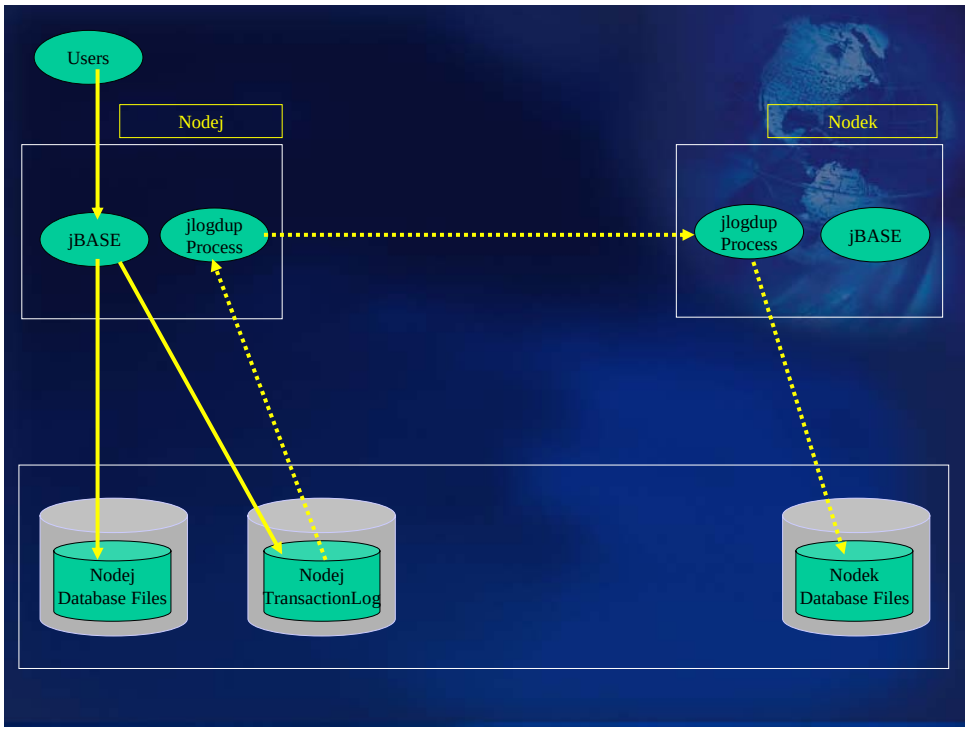
- For the majority of the time during the day, there is a tape deck free for other uses; either for system or development usage.

- This configuration allows for tape deck redundancy. If the event of a deck failure, the previous scenario can still be maintained while the tape deck is repaired or replaced.

- The jlogdup process can be left running during the backup/verify. This is the most important advantage over the previous scenario. Any database updates which are performed during backup/verify are likely not only be logged to the disk-based transaction log file, but also to the tape. This eliminates the lag between backing up the system and ensuring that database updates are logged to an external medium.

- The disadvantage of employing this configuration is that in the event of a system (or disk) failure, the machine has to be taken offline for the duration of the system restore process as well as the Transaction Log restore from tape. As time is money, this approach may be prohibitively costly to the organisation.

Failsafe/Hot Standby



The architecture depicted above shows the use of a Failsafe or Hot Standby machine. This allows for a failure of the live main machine (Nodej in this case). Unlike the previous configuration where the disk-based transaction logs are written to an external medium (tape), this configuration will enable database updates to be replayed to a standby machine, and indeed to the database on that standby machine, shortly after the update has been made (and logged) to the live machine.

Purpose of the Hot Standby configuration

Before describing how this configuration is set up and is managed, the role of the standby machine needs to be determined.

It is assumed that for the case of a full system reload, there is some external medium available for the operating system reload and configuration. This could also be contained on the standby machine as a system image. In the latter case, enough disk space should be available to hold this system image.

The database is to be replicated on the standby machine, so space must be available.

The processor/disk configuration should be fast enough on the standby machine, so as not to lag too far behind database updates on the live machine. The implication of the standby machine's unable to cope with the database update rate, may cause the live and standby machines' database to be unacceptably out-of-sync. – There may be too many disk-based transaction log entries on the live machine which have not been transferred via jlogup to the standby machine.

HOT STANDBY MACHINE AS A FAST RECOVERY MACHINE

If the Hot Standby machine is to be used within a fast recovery mechanism, then the following is required:
The network between the two machines should be fast and reliable.
The database on the standby machine must be sufficiently up-to-date, with reference to the live machine, as to be acceptable.

HOT STANDBY MACHINE TO BE USED BY ESSENTIAL STAFF DURING SYSTEM RECOVERY

The standby machine must have sufficient bandwidth to cope with the assigned tasks, within acceptable time frames. An example of this would be that if an assigned task were be to run End of Day processing, then the machine must be able to complete this task prior to the normal start of business the following day.
During the period when the live machine is unavailable, then the standby machine should be able to handle failures. A minimum configuration should be that Transaction Journaling should be initiated on the standby machine and the transaction log file produced should be backed up to an external medium (tape?).
Provision should be made to allow for disk-based transaction logs to be held.
Provision should be made for licensing of users on the standby machine.

HOT STANDBY MACHINE TO BE USED AS A LIVE MACHINE REPLACEMENT DURING SYSTEM RECOVERY

If the intent is that the standby machine becomes a temporary replacement for the live machine, then ideally the standby machine should be of similar configuration to the live machine.

Introducing a Hot Standby machine into the configuration

Assuming that the database on Nodej is of a consistent state, we may introduce a Hot Standby machine by means of the following procedure. The following describes how transactions are logged from system Nodej (the live machine) to a standby system Nodek, (the Failsafe/Hot Standby machine).

Transaction Journaling is started on Nodej; thus producing a transaction log file of updated records.

A jbackup/jrestore sequence is initiated from Nodej, by means of the script Backup_Restore. This will take a snapshot of the database on Nodej and reproduce it on Nodek. The jbackup option '-s /tmp/jbstart' is used to create a time-stamp file for later use.

find /JBASE_APPS /JBASE_SYSFILES /JBASE_ACCOUNTS –print | jbackup –s /tmp/jbstart –v –c | rsh Nodek –l jbasesu jrestore –N

Once this sequence completes, the updates which have occurred on Nodej since the start of the sequence, need to be updated onto the database on Nodej. This could be achieved with:

jlogdup -u10 input set=eldest start=/tmp/backup.logset0 terminate=wait output set=stdout | rsh Nodek /GLOBALS/JSCRIPTS/logrestore

This will start the updates from the oldest logset (set=eldest); the first database update will be at the time the backup stats file was written (start=/tmp/backup.logset0), i.e. the start of the backup; the transfer procedure will wait for further updates (terminate=wait). The method employed to effect the transfer is by means of a pipe. Data from the transaction log is put into the pipe (output set=stdout); this data is taken from the pipe by means of the logrestore command, initiated on Nodek by means of the rsh command (remote shell). The logrestore command sets up a jBASE environment and then initiates a jlogdup command, taking its input from the pipe (input set=stdin) and updating the database on Nodek (output set=database).

```
/GLOBALS/JSCRIPTS/logrestore Script

JBCRELEASEDIR=/usr/jbc
JBCGLOBALDIR=/usr/jbc PATH=$PATH:$JBCRELEASEDIR/bin
LD_LIBRARY_PATH=$JBCRELEASEDIR/lib:/usr/ccs/lib
JBCOBJECTLIST=/JBASE_APPS/lib: (or whatever it is for your usual users)
export JBCRELEASEDIR JBCGLOBALDIR JBCOBJECTLIST
jlogdup input set=stdin output set=database
```

The status of the jlogdup process can be monitored by running jlogstatus from a dedicated window:
jlogstatus -r5 –a

Optionally, jlogstatus can be run on Nodek to ensure the log is being restored correctly.

It is usual to configure more than one set of transaction log files. Initially logging will start to, say, set 1; and at some point logging to logset 2 will be initiated. This will usually be done daily just before each jbackup to tape. Then, typically, on the next day, logging will be switched back to logset 1 (and overwriting the previous transaction log) and the daily jbackup started.

Database update metrics should be established to determine the correct size of the logsets. The jlogstatus display should be monitored to ensure that the logsets don't fill the disk! Transaction Journaling can be configured to perform certain actions when the transaction log disks begin to fill past a configurable watershed.

Data Validity after a Failure Condition

In the event of a failure on Nodej, the standby machine, Nodek will contain an up-to-date database on which to continue operations. This is not necessarily strictly accurate because of several factors outside the control of this mechanism:
There is a configurable flush rate parameter which may be adjusted for Transaction Journaling. This parameter governs how often transaction log file updates, held in memory are flushed to disk. The minimum period between transaction log file flushes is 5 seconds. This will limit lost transaction log file updates to at maximum the last 5 seconds.

In the event of failure of the disk holding the transaction log file as well as the disk holding the database, the lost data is limited to those transactions which have been logged to disk, but not transferred to the standby machine, plus the logging of those transactions which have still to be flushed to disk. This situation is less quantifiable, but as the transaction log file reflects a sequential record of database updates over time, manual investigation would be required to determine the latest updates which were actually updated on the standby machine. Obviously, the database update transaction rate on the live machine governs the magnitude of this investigation.

Although the majority of database updates can be preserved after a system failure, what is not necessarily preserved is database integrity. The use of and management of transaction boundaries within application code ensures that only complete (multi-file) updates make it to the database. During system recovery (rebuild) only complete database transactions are rolled forward; those transactions which were not complete at the time of system failure are not written to disk. When initiating a transaction through the jBC command TRANSTART, the use of the option SYNC ensures that a memory flush will take place upon a transaction end or abort. This also ensures that the transaction log file is also flushed to disk, thus eliminating any delay in writing to disk. Subsequent to system failure, manual investigation is now targeted at complete application transactions rather than individual database updates, albeit at the possible expense of system performance.

System Recovery in a Hot Standby Configuration

If the standby machine (Nodek) is to be used as a temporary application machine, while the cause of the failure of Nodej is determined and resolved, then those users who are to continue, require to be "replugged" to Nodek. This could be automatic, whereby those users' PCs are automatically re-routed to Nodek on the unavailability of Nodej; otherwise a script could be run to re-assign Nodej's IP address to Nodek. The users in this case, would be requested to log on again to continue their work. This reassignment should only take place when the state of the database is established. The checks required are specific to each installation so cannot be predetermined here.

RECOVERY PROCEDURE

Wait for the TJ restore process on the standby system (Nodek) to finish. This will be known when the statistics on the number of records read remains constant.
Establish the validity of the database on Nodek and the transactions to be re-entered (if necessary).
Shut down the standby machine.
Shut down the Nodej machine if it isn't already completely down.
Restart Nodek in level 1. This is before the communications daemons are started. Create scripts to switch the IP addresses of the network and VTC cards to become those that Nodej formerly were. Continue the booting of Nodek.
Disable jBASE logons on Nodek.
Re-start the logger to a fresh set of transaction log files using the jlogadmin command.
When Nodej is repaired and first booted, you will need to boot it into level 1 so you can ensure the network and VTC addresses become those previously taken by Nodek.
Reload the operating system and jBASE on Nodek (if necessary). This can be contained in a system backup tape, held securely. This system backup should contain a skeleton system, including a valid jBASE installation, C++ compiler, peripheral and user logon definitions. Any upgrades the system should be reflected in this system backup.
Start a backup/restore sequence between Nodek and Nodej; thus
find /JBASE_APPS /JBASE_SYSFILES /JBASE_ACCOUNTS –print | jbackup –s /tmp/jbstart –v –c | rsh Nodej –l jbasesu jrestore –N
where :
The filesystems /JBASE_APPS etc. identified are examples for a jBASE system
/tmp/jbstart is the stats file to be dumped (used to timestamp start of save)
rsh Nodej will run a command on the Nodej machine
-l jbasesu identifies a jBASE user to be used for restores. This is important if indexes are to be rebuilt on restore (the user should have access to files and subroutines).
-N indicates that indexes should be rebuilt on-the-fly
Ensure the jBASE demons are started.
Enable jBASE logons. At this point it is safe for users to start using the system. Any updates since the start of the backup will be logged in the TJ log.

Once the backup/restore process has fully completed, the updates which have accrued on Nodek since is start can now be replayed thus:

jlogdup input set=current output set=stdout terminate=wait start=/tmp/jbstart | rsh Nodej –l jbasesu /JBASE_SYSFILES/logrestore

Once the two machines are in sync again both machines can be brought down, the network and VTC card addresses swapped, and users can be allowed to re-logon to the Nodej machine.

Other Considerations when running a Hot Standby Configuration

Password files must be kept in synchronization on both machines.

Spooler configurations need to be kept in sync.

Once the /JBASE_APPS have the developer sources in normal Unix files, the use of a nightly backup and a RAID configuration will be sufficient.

When developers BASIC and CATALOG their programs, they will go into their own directories rather than into /JBASE_APPS. At certain points in time, when no users are active, the programs and subroutine libraries will be copied en-bloc to both the Nodej and Nodek machines in /JBASE_APPS. This is the correct way to release new software and it needs to be done on both machines to ensure consistency of applications in the event of failure.

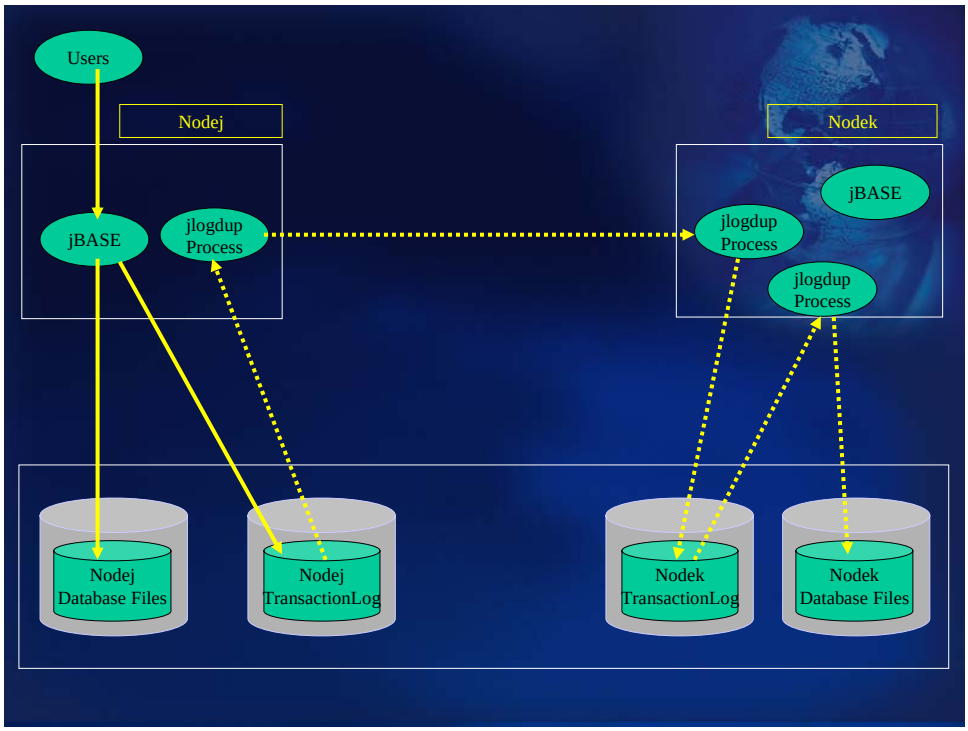
When an application developer changes an index or trigger definition, it should be done on files in their own environment. At some point you will want to release them into the live community. This again is best done when no users are active. To do this you will need to release the changed application and subroutine libraries (as shown above) and then release the new trigger and/or index definitions and apply the same changes to both the Nodej and Nodek machines. The indexes will need to be rebuilt on both machines.

All changes to jBASE scripts kept in the /JBASE_SYSFILES will need to be manually duplicated.

Many of the synchronization requirements should be checked nightly in a cron script and errors reported. Such a script could be made to verify the password file, the jBASE spooler configuration, the Unix spooler configuration., the scripts in the /JBASE_SYSFILES file system, check that the programs and subroutine libraries are identical on both Nodek and Nodej, and could check the index and trigger definitions are identical on both Nodek and Nodej, check the cron jobs are the same and the scripts they invoke are the same.

This verification of the two machines could also be run following a rebuild.

Refinement to Hot Standby Configuration



The configuration above shows a small, but significant refinement to the previous configuration. Essentially, the transaction log file is being replicated to Nodek, with the logrestore script showing the following change:

```
jlogdup input set=stdin output set=logset
```

Thus, all updates transferred from the transaction log file on Nodej are updated to the transaction log file on Nodek. Another jlogdup process is initiated thus:

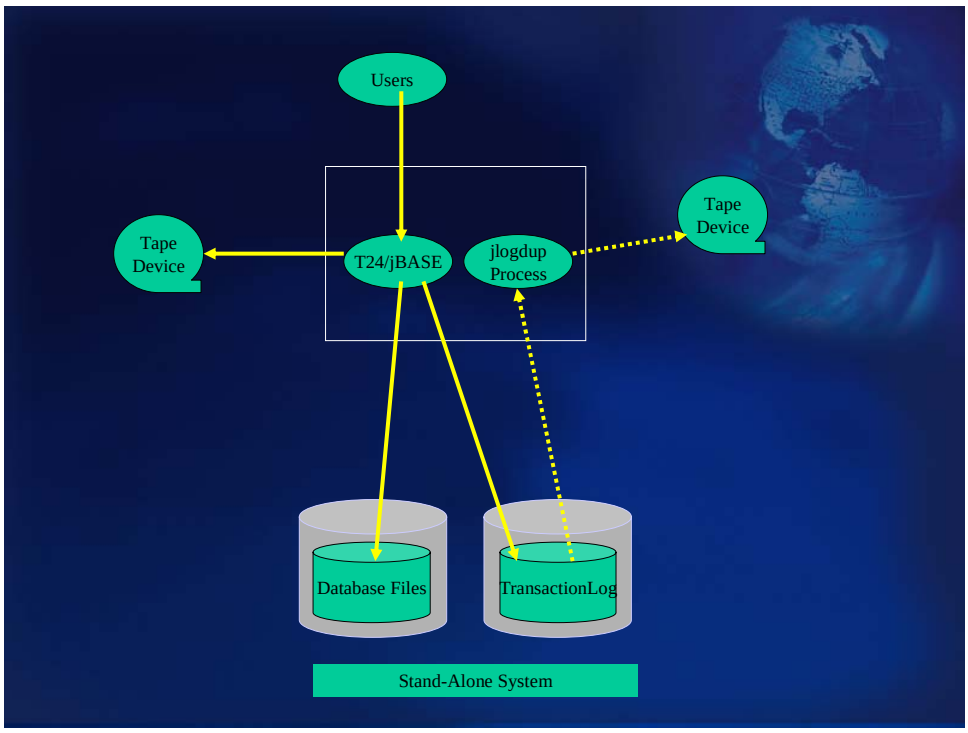
```
jlogdup input set=current output set=database
```

this takes these updates and applies them to the database. The reason for this becomes apparent when a recovery is required. Because there is now a copy of the transaction log file on the standby machine, by interrogation of the transaction log file, it is clear which updates have been transferred from the live machine. If the jlogdup is allowed to continue until all updates in the transaction log file have been applied to the database, then the recovery position can be established far more easily than by interrogating the database.

Resilient T24 Configurations

Each configuration which will be described adheres to those goals as identified in the Temenos Technology and Research White Paper - T24 Resilience High Availability in Failover Scenarios and the proposed new Close of Business Procedures as described in the Functional Specification Changes to Batch Operation for Global Processing Environments

Stand-Alone System – application server and database server on one machine



This should be the minimum standard configuration utilizing Transaction Journaling. The assumptions made here are that jBASE will be the database (native) server.
Transaction handling will be achieved by the use of the programming commands:

- TRANSTART
- TRANSEND
- TRANSABORT

Transactions which are not completed in their entirety will be completely “rolled back” by jBASE, when commanded to so do by the TRANSABORT command. Upon execution of the TRANSEND command all or none of the constituent database updates will be actioned, ensuring database consistency. Any transactional recovery will be achieved through the use of jBASE facilities.

jBASE transaction journaling will be used to record all database updates.
Transaction Journaling has been configured for example, with two logsets:

1. /bnk/bnk.jnl/logset1
2. /bnk/bnk.jnl/logset2

where: logset1 and logset2 are links to two mounted filesystems each containing the corresponding transaction log file definitions.
TJ is then activated by a script similar to **start_tj**, which activates transaction logging and also the backup of the transaction logs to tape (/dev/rmt/0 in this case).
The Transaction journal is copied to tape (or other external medium) on a continuous basis by means of the jlogdup facility.
A backup of the database (using the backup_jbase script) is initiated prior to the execution of Close of Business procedures. Logsets are "switched" following the successful completion of backups.

When a backup is required, a script, based on "**backup_jbase**" is run.
Actions performed by this script are:

- Disk-based transaction log file entries are still allowed to be dumped to tape. When there is no Transaction Logging activity, then all outstanding transactions have either been logged to tape or rolled back. **NOTE**: The time allowed for transactions to complete is dependent on application design. The end of dump activity can be checked by use of the jlogstatus command
- The transaction log file duplication process to tape is stopped.
- The logging tape is replaced by a new tape for the backup.

The command:

```
find /bnk -print | jbackup -v -f –c /dev/rmt/0 -s /tmp/jbstart
```

will dump all data to tape below **/bnk**; As all the transaction log data (bnk.jnl) data has already been dumped to tape prior to the backup, the exclusion of this directory would seem appropriate, by configuring the data thus:

Directory	Description
bnk	Main directory, object code etc.
bnk.run	Initial logon point
bnk.data	Data files
bnk.dict	File dictionaries

bnk.help	On-line help files
bnk.jnl	Transaction Journal

where bnk.jnl is not under the bnk directory structure.

NOTE

The use of the “-c” option will allow for the dumping of index files to avoid having to rebuild indexes on a restore process.

NOTE 2

Alternative backup mechanisms may be employed.

Once the backup has completed and verified, a new tape for tape logging replaces the last backup tape.
The logsets are switched, ready for any database updates.
Transaction logging to disk is re-enabled.
Database updates are enabled.

SYSTEM PROTECTION AND BENEFITS

The use of Transaction Journaling in this configuration allows for the recovery of transactions up to the point of failure. This configuration provides assistance to the administrator in identifying those transactions which have not been written to tape prior to system failure. The tape (set) contains a sequential history of database updates since the last backup.

SYSTEM RECOVERY PREPARATIONS

The administrator must ensure that a skeleton system save is made available. This skeleton system should contain:

- The operating system and configuration (device assignments, user login information, etc).
- A licensed copy of jBASE (configured as ready-to-run)
- This skeleton system must be kept up to date. Any changes to the operating system or jBASE configurations must be reflected in this skeleton system as a standard procedure; any such changes triggering the production of a new skeleton system.

SYSTEM RECOVERY AFTER FAILURE

If the operating system and/or jBASE is deemed corrupt or there has been a catastrophic disk failure, resulting in the loss of a disk, then the system should be reconstructed as a skeleton system as discussed above. The details of this recovery are out of the scope of this document.

Once the system has been brought to an operational state, the database needs to be brought back to a known state. The last backup set produced is recovered by the recover_jbase script. This not only restores the jBASE

database including saved indexes, but also replays all completed transactions which have been transferred to tape and initiates transaction logging to tape.

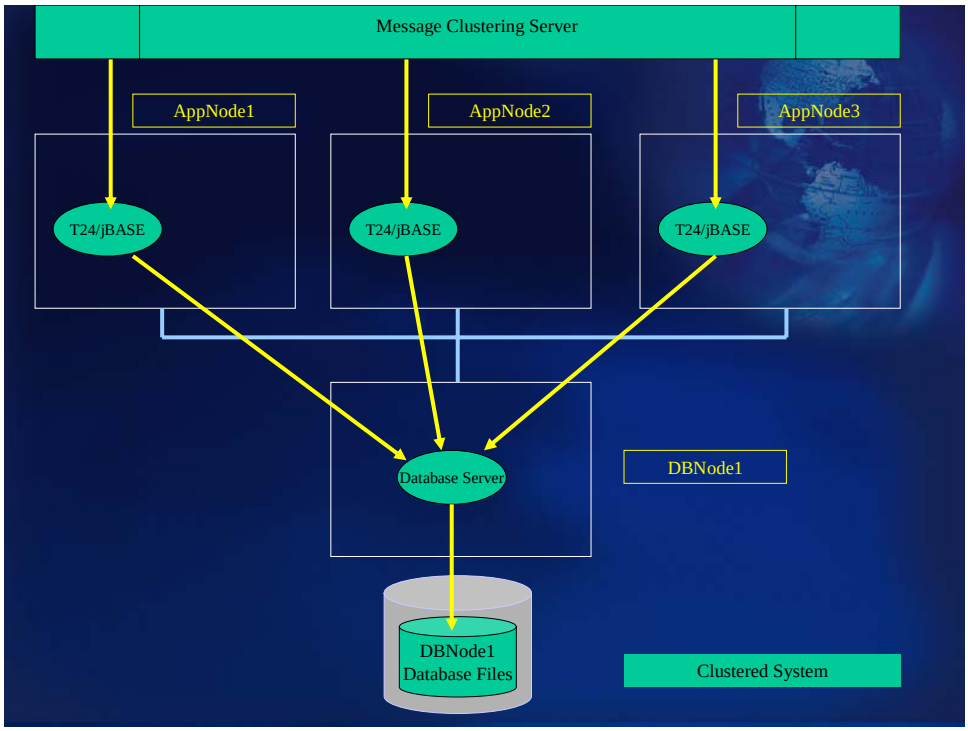
If there has been an application/database error which has resulted in the decision to perform a complete restore of the system, it is clear that if the error can be identified to have taken place at a particular time, (whether precisely or approximately), then the whole of the transaction log should not be replayed. Using the "end=timespec" option of jlogdup will cause the transaction log replay to terminate at the specified time rather than the end of the logset. (See jlogdup section for valid format of timespec). The recover_jbase script will prompt for a time or assume EOS (i.e. all the transaction log is to be replayed). As the

Warning: If an "end=timespec" parameter has been specified, then the time chosen may cause transactions which began before this time not to be rolled back. Additional database updates pertaining to such transactions and bounded by the corresponding TRANSEND commands may exist on the transaction log file, but will not be executed.

CLOSE OF BUSINESS PROCEDURES

This configuration, being a jBASE-only solution will allow for on-line backups to be taken prior to Close of Business procedures.

Cluster System – multiple application servers and a single database server



When clustering T24, two (at least) configurations can be utilised:

MULTIPLE APPLICATION SERVERS WITH A JBASE DATABASE SERVER

With this configuration, jBASE will be the sole database server. Communication between the application server(s) and the database server will be by using jRFS within jBASE. This allows multiple application servers to have pointers/stubs as file definitions. These pointers/stubs reference files exist on the database server. jRFS mechanisms allow for the updating of the database through jRFS server processes from requests made on the application servers. The implication of this is that each application server has no direct, individual database storage but shares access to a central (jBASE) database. As there is only one database server, Transaction Journaling facilities will be available, using the same mechanisms as the Stand-Alone system above.

MULTIPLE APPLICATION SERVERS WITH A NON-JBASE DATABASE SERVER

This configuration uses jBASE as a gateway to another DBMS (such as Oracle or DB2).
jBASE will handle any supported relational database connectivity (such as Oracle/DB2 etc.) through the appropriate jEDI driver. Data mapping will be achieved through the corresponding RDBMS stub file definitions. The jBASE/RDBMS stub file definitions can exist on one of various locations:

- On the Application Servers – this could (would) potentially create a locking minefield – how to communicate between the Application Servers the locked state of the database entities.

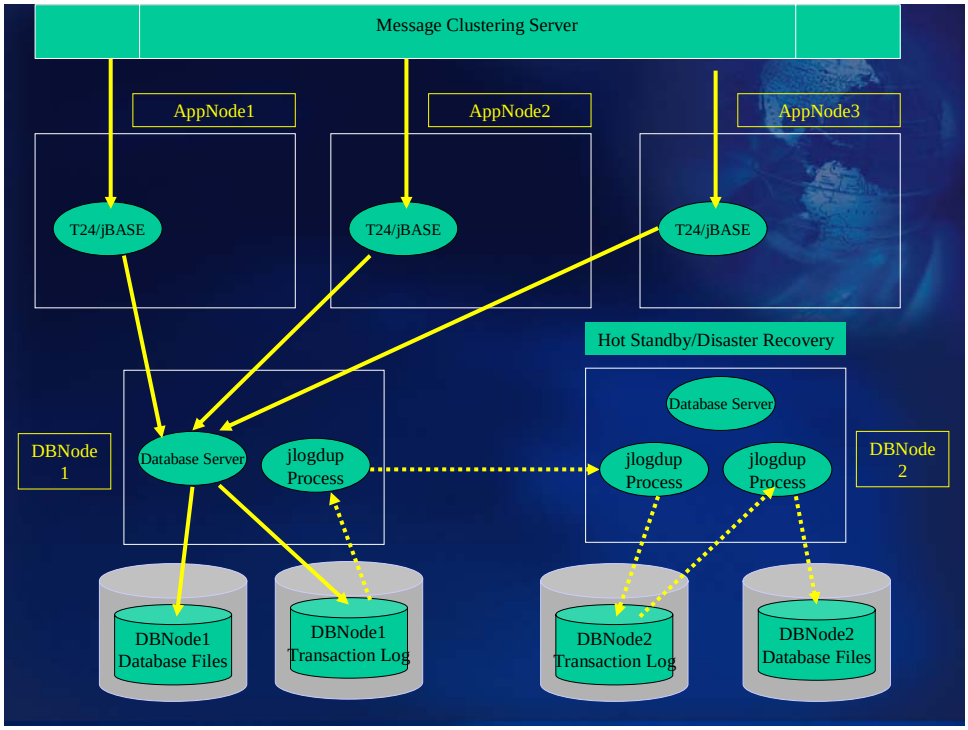
- On the Database Server (1) – Application Servers communicate over NFS mounts to RDBMS stub files defined on the Database Server. The downside of this approach is that RDBMS client components (at least) have to exist on each of the Application Servers. Also there is a problem with managing database locks. This can be achieved by inefficient application-level lock mechanisms whereby the locks are held within a central filesystem and are accessed by all Applications Servers, utilizing OS locks to manage access to the lock table.

- On the Database Server (2) – Application servers communicate using a jRFS driver to jRFS servers on the Database Server. The Database Server contains the RDBMS stub file mappings to the RDBMS, also residing on the Database server. As jRFS operates in a client-server relationship, there are no locks taken directly by any process within the Application Servers, but are taken by the jRFS server processes, on their behalf, running on the Database Server. As all the jRFS server processes run under control (for locking purposes) of a single jBASE server, there is no issue with locking between these processes. There is also likely to be a cost advantage over Database Server (1) approach, because no RDBMS components need to exist on the Application Servers.

- Transaction management (i.e. the use of TRANSTART, TRANSEND and TRANSABORT programming commands) within the Application Servers is handled within jBASE as for the Stand-Alone system.

Hot Standby database server

HOT STANDBY WITH A JBASE DATABASE SERVER



The Hot Standby configuration using jBASE as the database server has the same attributes as previously described in the Cluster Systems with the exception that all database updates to jBASE are duplicated to a separate server (or remote in the case of disaster recovery). The database duplication process, achieved by the jlogdup facility, would normally be an operation in addition to dumping the transaction log data to a local tape device.

Operation of the Hot Standby Configuration

Transaction handling will be achieved by the use of TRANSTART, TRANSEND and TRANSABORT programming commands.

- jBASE transaction journaling will be used to record all database updates.
- The Transaction journal is copied to tape (or other external medium) on a continuous basis by means of the jlogdup facility.
- A backup of the database (using jbackup) is initiated each night at 12:01 am (for example) to the tape deck /dev/rmt/0 (for example).
- Logsets should be switched automatically following the backup.

- A jlogdup process will be initiated on the database server which will, in tandem with a corresponding jlogdup server process on the standby server, transfer all transaction updates from the transaction log on the live cluster to the transaction log on the standby server.

Another jlogdup process on the standby server will take the updates from the previously transferred log files and update the database on the standby server.

HOT STANDBY WITH A NON-JBASE DATABASE SERVER

If a backend RDBMS is configured then Hot Standby/disaster recovery is handled by the RDBMS; jBASE Transaction Logging is not used as the recovery mechanisms are handled by the RDBMS. The RDBMS recovery mechanisms are outside of the scope of this document.

The updates contained within a transaction are cached until a TRANSABORT or TRANSEND command is executed for that transaction. No RDBMS activity takes place when the TRANSABORT command is executed, whereas the TRANSEND can result in many RDBMS interactions before success or failure is detected. The application code within T24 is unaware of the underlying backend database.

Scripts

setup_tj

```
#! /bin/ksh
export JBCRELEASEDIR=/data/colins/4.0_rels/jbcdevelopment
export JBCGLOBALDIR=/data/colins/4.0_rels/jbcdevelopment
export LD_LIBRARY_PATH=$JBCRELEASEDIR/lib:$LD_LIBRARY_PATH

jlogadmin -cf1,1,[logset1 directory]/logfile1
jlogadmin -cf1,2,[logset1 directory]/logfile2
jlogadmin -cf2,1,[logset2 directory]/logfile1
jlogadmin -cf2,2,[logset2 directory]/logfile2
jlogadmin -cf3,1,[logset3 directory]/logfile3
jlogadmin -cf3,2,[logset3 directory]/logfile3
```

start_tj

```
#! /bin/ksh
export JBCRELEASEDIR=/data/colins/4.0_rels/jbcdevelopment
export JBCGLOBALDIR=/data/colins/4.0_rels/jbcdevelopment
export LD_LIBRARY_PATH=$JBCRELEASEDIR/lib:$LD_LIBRARY_PATH

jlogadmin -l 1 -a Active
echo `date` > $JBCRELEASEDIR/logs/jlogdup_to_tape_start
jlogdup input set=current terminate=wait output set=serial device=[Device
Spec] &
```

stop_tj

```
#! /bin/bash
export JBCRELEASEDIR=/data/colins/4.0_rels/jbcdevelopment
export JBCGLOBALDIR=/data/colins/4.0_rels/jbcdevelopment
export LD_LIBRARY_PATH=$JBCRELEASEDIR/lib:$LD_LIBRARY_PATH

jlogadmin –a Off

start_jlogdup

#! /bin/ksh
export JBCRELEASEDIR=/data/colins/4.0_rels/jbcdevelopment
```

```
export JBCGLOBALDIR=/data/colins/4.0_rels/jbcdevelopment
export LD_LIBRARY_PATH=$JBCRELEASEDIR/lib:$LD_LIBRARY_PATH

echo `date` > $JBCRELEASEDIR/logs/jlogdup_to_tape_start
jlogdup input set=current terminate=wait output set=serial device=[Device
Spec]&


stop_jlogdup

#! /bin/ksh
export JBCRELEASEDIR=/data/colins/4.0_rels/jbcdevelopment
export JBCGLOBALDIR=/data/colins/4.0_rels/jbcdevelopment
export LD_LIBRARY_PATH=$JBCRELEASEDIR/lib:$LD_LIBRARY_PATH

jlogadmin -k* > discard
```

backup_jbase

```
#! /bin/ksh
export JBCRELEASEDIR=/data/colins/4.0_rels/jbcdevelopment
export JBCGLOBALDIR=/data/colins/4.0_rels/jbcdevelopment
export LD_LIBRARY_PATH=$JBCRELEASEDIR/lib:$LD_LIBRARY_PATH

typeset -u TAPEOUT
typeset -u REPLY
typeset -u BACKUPOK

set TAPEOUT = N
print -n "Are you backing up the logfiles to tape? (Y/N) "
while [ "$TAPEOUT" != Y -a "$TAPEOUT" != N ]
do
read TAPEOUT
done
if [ "$TAPEOUT" != N ]
then
print -n Has all logging to tape finished - press any key when it has
read REPLY
jlogadmin -k* >discard
print Logging to tape terminated
fi
```

```
if [ "$TAPEOUT" = Y ]
then
print Please remove the tape for logging and replace with the backup tape
set REPLY = N
while [ "$REPLY" != Y ]
do
print -n Enter Y to continue
read REPLY
done
fi
set BACKUPOK = N
while [ "$BACKUPOK" != Y ]
do
print Backup Started `date`
find /data/globus/jbbase13207/mbdemo.data -print | jbackup -v -c -f [Device
Spec] -s /tmp/jbstart
print  Waiting for tape to rewind
sleep 5
print Verify Started `date`:
jrestore -f [Device Spec] -P
print Verify Finished `date`
print -n "Backup successful <Y/N>"
read BACKUPOK
done
jlogadmin -l next -a Active
print logsets switched and logging to disk restarted
if [ "$TAPEOUT" = Y ]
then
print Mount a new tape for logging
print Enter any key to resume logging to tape
read INPUT
print `date` > $JBCRELEASEDIR/logs/jlogdup_to_tape_start
jlogdup input set=current terminate=wait output set=serial device=[Device
Spec] &
print Logging to tape restarted
fi
```

recover_jbase

```
#!/bin/ksh
if [ -z "$1" ]
then
```

```
        echo "\nWhat is the nature of the recovery :-\n"
        PS3="Option :"
        select Sel in "Full Restore Required" "Tape Logging Failure"
        do break; done
        if [ -z "$REPLY" ]
        then
                exit
        fi
else
        REPLY=$1
fi
if [ $REPLY = 1 ]
then
echo Put the first backup tape in the tape drive
echo -n Enter any key when ready
read DONE
jrestore -f  [Device Spec] -N
echo -n Is a Transaction Log tape available ?
read REPLY
if  [ $REPLY = "y" ]
then
echo Put the first  log  tape in the tape drive
echo -n Enter any key when ready
read DONE
echo -n "Enter a time to terminate the duplication process (or RETURN for
all logs)"
read  ENDTIME
if [-z $ENDTIME ]
then
jlogdup input  set=serial device=[Device Spec] backup terminate=EOS output
set=database
else
jlogdup input  set=serial device=[Device Spec] end=$ENDTIME output
set=database
fi
        fi
else
echo Put a new tape in the tape drive
echo -n Enter any key when ready
read DONE
```

```
jlogdup input set=current start=$JBCRELEASEDIR/logs/jlogdup_to_tape_start
terminate=wait output set=serial device=[Device Spec] &
fi
```

REMOTE.JOURNAL.STARTUP

```
find /JBASE_APPS /JBASE_SYSFILES /JBASE_ACCOUNTS –print | jbackup –s
/tmp/jbstart –v –c | rsh Nodek  –l jbasesu jrestore –N
```

Transaction Journaling Utilities

jlogadmin

The jlogadmin command allows for the administration of the jBASE Transaction Journal. The jlogadmin command will enabled for interactive usage when invoked by the super-user/Administrator; execution by other users being restricted to read-only. All administration tasks contained within the jlogadmin utility can also be invoked from the command line, using jlogadmin, with optional parameters.

When the jlogadmin command is executed interactively, navigation to the next field is by using the tab key or cursor-down key and to the previous field by the cursor-up key. Each field can be modified using the same editor type commands as available in jsh. Changes to a particular field are effected by the <Enter> key and CTRL-X is used to exit from interactive mode.

Interactive Configuration

INTERACTIVE DISPLAY

The first execution of jlogadmin will display the following screen:

Description of Fields

STATUS

Specifies the current transaction journal status, which can be On/Active, Off/Inactive or Susp/Suspended. Note: When the status is changed to Suspended, all transactions which would be updated in the transaction log file will also suspend awaiting a change of status.

CURRENT SWITCHED LOG SET

Specifies the current log set in use. There are four possible log sets – numbered 1 to 4. An entry of 0 indicates that no log set has been chosen at this time.

EXTENDED RECORDS

Specifies additional information:

- the application id
- the TTY name
- the login name

TIME BETWEEN LOG FILE SYNCS:

Specifies the number of seconds between each synchronization of the log set with the disk; All memory used by the log set is force flushed to disk. Should the system crash, the maximum amount of possible data loss is limited to the updates which occurred since the last log set synchronization.

Log notify program:

This specifies the program to execute when the warning threshold of the log set is reached. The log notify program is called every time a message is written to jediLoggerAdminLog. The text of the message can be captured by adding arguments to the command line which the notify program can examine using SENTENCE(). For example, possibly define the program as:

/usr/admin/bin/lognotify '%1' '%2' '%3'

In addition, when the program is loaded, the following are substituted:

%1 == {INFORMATION: | WARNING: | FATAL ERROR:} From user root at Wed Sep 04 12:38:23 2002

%2 == Process ID 12345, Port 23, tty /dev/pts/03

%3 == Depends upon the actual error message e.g. "Error number nnn while reading from file /dev/xxxxx"

NOTE: The message is designated INFORMATION, WARNING or FATAL ERROR. This designation can be used by the log notify program to decide on a course of action. The messages that can be logged are:

Type	Message	StdOut
INFORMATION	Log set changed to *s*	Yes
	Log set *s* truncated	Yes

	File *f* for log set *s* REMOVED	Yes
	File *f* for log set *n* changed to *newfilename*	Yes
	n files imported to log set *n* (see -i option)	Yes
	Status of logger set to *status* (current log set *s*)	Yes
	Sync count changed from every *n1* seconds to every *n2* seconds	Yes
	Log file warning threshold set to *p* initial percentage thereafter every additional *q* percent or *n* seconds	Yes
	Admin. Log Notify Program now set to *program*	Yes
	Admin. Log Notify Program REMOVED	Yes
	Extended Record Status now set to *on*|*off*	Yes
	Log set switch detected, was set *n1*, now set *n2*	No
	Kill initiated on jlogdup process id *pid* : Process id pid from port *n*	Yes
	First record read from set *n*	Yes
	Termination Statistics: usr *x* , sys *y* , elapsed *z* *r* records read from current log set number *n* : *r* records, *b* blocks, *rb* record bytes , *e* errors in *file*	Yes
WARNING	Journal Log Files now at *p*% capacity	No
FATAL ERROR	Unable to open logger configuration file *filename*	Yes
	Sync demon appears to have died prematurely	Yes
	Error number *errno* while reading from file *filename*	No
	Error number *errno* while writing to log file	No
	Error errno while writing to log journal file *filename*"	Yes
	Error errno while writing to log journal	Yes
	Unable to open logger file *filename*	Yes

	Out of memory to log update	Yes

WARNING THRESHOLD

If the amount of space consumed in the file system, which the active logset resides upon, exceeds the specified threshold, it runs the log notify program. Individual files in a logset have a capacity of 2GB. If the logsets are not switched, files in a logset can grow to the 2GB limit without the file system reaching the threshold capacity. If this happens, journaling will cease to function predictably and normal database updates may fail.
File definitions:
As indicated above, the maximum size of an individual file is 2GB. It is clear that if a single file were used for the log file, then this would likely be insufficient for most realistic application environments. Therefore the administrator is able to set up a log set consisting of a maximum of sixteen files, thus enabling a maximum log set of 32GB. The configuration will allow for a maximum of four log sets. Usage and switching of the four log sets will be described in appropriate sections. If the file specified by the administrator does not already exist, then it will be created automatically.

COMMAND-LINE SYNTAX

In addition to the interactive screen setup facility, there are options which can be added to the jlogadmin command execution. This allows the administrator to create scripts which can be run either at pre-defined times or intervals; or in response to transaction journal events (usually error handling events).
The command is invoked by the following:

```
jlogadmin –{options}
```

Where {options} are identified below:

SYNTAX ELEMENTS

Option	Description
-a status	Set status On/Active, Off/InActive, or Susp/Suspend
-c	Create file in log set if does not exist. (use with -f)
-f set,fileno,file	Change log filename in log set where Set log set Fileno File number File File name
-h	Display help

-i[1-4],filename{,filename...}{-o}	Import a log set to override one of the 4 standard log sets. The -o argument is optional. If used it suppresses the warning and confirmation message. You can specify up to 16 filenames to define the imported log set.
-k pid | * | ?	Kill jlogdup process ‘pid’ or ‘*’ all or ‘?’ to list.
-l num | next | eldest	Switch to log set where Num — log set number 1-4 Next — next sequential log set Eldest — earliest log set
-n program	Set threshold notify program.
-o	Perform operation without checking if the specified log set is empty. Used with -f and -t.
-s secs	Set synchronization period.
-tn	Truncates log set n. The log set may not be the current switched set. This option ensures that disk space will be freed and is sometimes preferable to "rm" which may not free the disk space if any process still has log files open.
-w pp, ii, ss	Set threshold where Pp — initial warning percent Ii — every percent after initial percent Ss — every second after initial percent
-x status	Set extended log record ON or OFF
-C	Clear transaction journal administration log file jediLoggerAdminLog.
-V	View transaction journal administration log file jediLoggerAdminLog.

jlogstatus

The jlogstatus command displays the status of the jBASE Transaction Journal. In its simplest form the jlogstatus, command shows a summary of the current Transaction Journal activities. Additional command line options are available for output that is more verbose. The jlogstatus command can also be used to present a rolling status screen, using the '-r n' option, which will update the display every 'n' seconds.

SYNTAX

```
jlogstatus -options
```

SYNTAX ELEMENTS

Option	Description
-a	display all available information
-c	display current log information
-d	display jlogdup process information
-g	display general information
-h	display help
-l	display all Log files information in summary mode
-r nn	set display to repeat every nn seconds
-v	verbose mode

EXAMPLE

```
jlogstatus -a –v –r 5
```

This will display all information and will refresh every 5 seconds.

Journal status:	active
Configuration file created:	10:39:00 08 APR 1998 , by root from port 9
Configuration file modified:	10:47:55 08 APR 1998 , by root from port 9
Journal file sets switched:	10:41:21 08 APR 1998 , by root from port 9

Full log warning threshold:	70 percent , thereafter every 1 percent or 300 secs
Journal files synced every:	10 seconds , last sync 10:49:59 08 APR 1998
Background sync demon:	(inactive)
Extended record:	OFF
Admin log file:	10 entries , in file /usr/jbc/config/jediLoggerAdminLog
Admin log notify program:	(undefined)
Current log file set:	1, date range 10:41:21 08 APR 1998 to 10:49:59 08 APR 1998
/home/jbasedev/fb1:	76.92% capacity
/home/jbasedev/fb2:	76.92% capacity
Total record count:	97.66k , 0 records/second
Total byte count:	19.47M , 0 bytes/second
jlogdup program status:	NONE active
Status log set 1 (current):	2 files, 100000 records , 20415568 bytes used
	Date range 10:41:21 08 APR 1998 to 10:49:59 08 APR 1998
Status log set 2:	No files defined
Status log set 3:	No files defined
Status log set 4:	No files defined
Status log totals:	2 files, 100000 records, 20415568 bytes used
	Date range 10:41:21 08 APR 1998 to 10:49:59 08 APR 1998

jlogsync

When a jBASE application performs a database update, it writes to the transaction log file (if active). It does this to a memory image and normally it is up to the platform file system to flush the memory image to disk every so often, by default on most platforms this is usually every minute.

You can use options in jlogadmin so that the jBASE processes themselves do this file synchronization more often. The default is every 10 seconds. This means in the event of a system failure, you will lose at the most 10 seconds worth of updates.

The use of the jlogsync program means the jlogsync process instead of individual jBASE processes performs file synchronization. Therefore alleviates the overhead of the synchronization from the update processes. Thus, the jlogsync process is not mandatory. However, in a large installation it may provide beneficial performance gains.

SYNTAX

```
jlogsync -options
```

SYNTAX ELEMENTS

Option	Description
-b	run in the background (normal operation)
-d	display jlogsync demon status
-i	initialize and become the jlogsync demon
-k	kill the jlogsync demon
-t nn	Inactivity timeout period (seconds) for detecting jlogsync being killed
-v	verbose mode
-S	force synchronization now

jlogdup

The jlogdup command provides the capability to duplicate transaction log set data from the jBASE Transaction Journal. Use this command to duplicate transaction log data to or from any platform file, including device files, standard input, ‘stdin’, and standard output, ‘stdout’.

SYNTAX

```
jlogdup -Options INPUT input_spec OUTPUT output_spec
```

SYNTAX ELEMENTS

Option	Description
-e file	error file for database update errors
-f	used with the -v or -V option; shows information for the next (future) update; by default information for past updates is displayed
-h	display help
-l file	log file to write all status and errors information
-m nn	maximum number of errors (default 10000)
-u nn	display '*' every nn input records
-v	verbose mode, 1 line per record
-x	exclusive use of the database, no group locks taken
-H	display verbose help screen
-V	Very verbose output, also 1 line per record

INPUT_SPEC/OUTPUT_SPEC

The input/output specification can specify one or more of the following parameters

Parameter	Description
blockmax=nnn (S)	the maximum size, in blocks, of a serial device
blocksize=nnn	the block size to read/write to TTY/SERIAL device or file

device=file%dev (S)	the file name for SERIAL device. Can be more than one
encrypt=true(O)	Output transfer is to be encrypted
end=timespec (I)	time in log set at which to stop restore/duplication
hostname=host(IOK)	Host for socket transfers to / from
key=encryptkey	The key to use for encryption
noflush=true (O)	suppress flush of output at end of transaction. (default false)
notrans=true (O)	ignore transaction boundaries. (default false)
port=portnum (IOK)	Socket port to use for socket transfer
prompt=true	prompt when switching serial devices or files
rename=from,to	convert path name directories ‘from’ to ‘to’ on restore
renamefile=file (O)	use rename file list of format ‘from,to’ to rename files
retry=nn (I)	specifies the interval between retries, when 'terminate=wait'
scheme=method	Encryption method
set=current (IL)	begin restore/duplication using the current log set as input
set=database (OD)	output is to the database, i.e. Restore mode
set=eldest (IL)	begin restore/duplication using the eldest log set
set=n (ILN)	begin restore/duplication using log set number n
set=null (O)	output is to be discarded
set=serial (S)	input/output is to a serial device or file. Requires ‘device=’
set=socket (IOK)	Input/output is to a socket. Requires “hostname=” and “port=”
set=stdin (IT)	the input data comes from the terminal stdin
set=stdout (OT)	the output data is directed to the terminal stdout
set=tty (T)	the input is from stdin or the output is to stdout
set=logset (OL)	the output is directed to the current log set as an update

start=timespec (I)	time in log set at which to start restore/duplication
terminate=eof (I)	terminate restore/duplication at eof of eldest log set
terminate=eos (I)	terminate restore/duplication at end of current log set
terminate=wait (I)	switch to elder log sets as required and wait for new updates
timeout=nnn (I)	timeout period in seconds for ‘terminate=wait'
verbose=true	display to stderr a summary of the specification

The indicators in brackets denote:

Indicator	Meaning
D	specification valid for type database
I	specification valid for type input
K	specification type for socket
O	specification valid for type output
L	specification valid for log set
N	specification valid for type of null
S	specification valid for type serial
T	specification valid for type terminal

TIMESPEC

The time specification, used in the ‘start=’ and ‘end=’ specification can be one of the following formats:

timespec	meaning
hh:mm:ss	time of day (todays date assumed)
DD-MMM-YYYY	date (midnight assumed), Any date convention accepted
hh:mm:ss,DD-MMM-YYYY	both time and date specified either way around

jbackup_file	time of file created. Use with 'jbackup -sfilename' option
filename	regular file, use the time the file was last modified

HOST

The IP address or the DNS name of the host to use for socket transfers

PORTNUM

The TCP port to use for socket transfers.

KEY

The string to be used as the encryption key for the transfer of journal entries.

METHOD

The encryption scheme to use for the transfer of journal entries. This mechanism utilizes OpenSSL high level cryptographic functions. The valid specifications for encryption are;

- RC2
- BASE64
- DES
- 3DES
- BLOWFISH
- RC2_BASE64
- DES_BASE64
- 3DES_BASE64
- BLOWFISH_BASE64

jlogmonitor

SYNTAX

jlogmonitor {-h|?} {-ccmd} {-Cnn} {-Dnn} {-E} {-Inn) {-Snn}

SYNTAX ELEMENTS

Option	Description
-ccmd	The command cmd is executed when an error occurs.
-Cnn	If the file system utilization of the journal log exceeds nn% full then an error message is displayed. The error message is repeated for every 1% increase in file system utilization.
-Dnn	If the jlogdup process processes no records (or if there is no jlogdup process active), then after nn minutes of inactivity it displays an error message. It repeats the error message every nn minutes while the jlogdup process(es) is inactive.
-E	If the jlogdup program reports an error, this option causes jlogmonitor to also display an error. You can view the actual nature of the error by either looking at the screen where the jlogdup process is active, or by listing the jlogdup error message file (assuming the –eERRFILE option was used).
-h	display help
-lnn	The status of the Journaler can be ACTIVE, INACTIVE or SUSPENDED. If the status of the journaler is either INACTIVE or SUSPENDED (with jlogadmin) for more than nn minutes, it s=displays an error message. The error message will be repeated every nn minutes that the journaler is not active
-Snn	Use this option to determine if any updates are being applied to the journal logs. If no updates are applied to the current journal log set for nn minutes it displays an error message. It repeats the error message for every nn minutes of system inactivity.

NOTES

You must specify at least one of the options, -C, -D, -E, -I or -S.

EXAMPLE

jlogmonitor –c"MESSAGE * %"

The command "MESSAGE * %" is executed for every message sent to the screen by jlogdup. The jlogmonitor specially interprets the use of the % by the program and will be replaced with the error message.