jBASE Dataguard

jBASE resilience features in jBASE 5.0 and upwards

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 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.

Copyright

Acknowledgements

Information regarding Unicode has been provided in part courtesy of the Unicode Consortium. The Unicode Consortium is a non-profit organization founded to develop, extend and promote use of the Unicode Standard, which specifies the representation of text in modern software products and standards. The membership of the consortium represents a broad spectrum of corporations and organizations in the computer and information processing industry. The consortium is supported financially solely through membership dues. Membership in the Unicode Consortium is open to organizations and individuals anywhere in the world who support the Unicode Standard and wish to assist in its extension and implementation.
Portions of the information included herein regarding IBM's ICU has been reprinted by permission from International Business Machines Corporation copyright 2001

Errata and Comments

If you have any comments regarding this manual or wish to report any errors in the documentation, please document them and send them to the address below:

Technical Publications Department
TEMENOS UK Limited
2 People Building
Hemel Hempstead
Hertfordshire HP2
4NW
England
Tel SB: +44 (0) 1442 431000
Fax: +44 (0) 1442 431001

Please include your name, company, address, and telephone and fax numbers, and email address if applicable. documentation@temenos.com

Contents

PREFACE

This document is intended as a guide for system/database administrators in their configuration of and maintenance of resilient systems utilising the jBASE database. Knowledge of the administration of and the maintenance of standard systems base on jBASE is assumed.

INTRODUCTION

Components

Traditional jBASE systems essentially comprise three parts: User- and System- related files – "the database"; an application suite of programs to manipulate the data in the database - "the application" and a DBMS system comprising jBASE programs and user-developed programs to service database requests made by the application. The database is the only component which requires special attention with regard to resilience; the others can merely be reloaded from an archive image. The database is the only fluid component – it changes from day-to-day and probably from second-to-second. This document will describe the features of jBASE which exist in order to protect the database from potential problems which could occur, as well as the methods to use when confronted by each of such circumstances.

Databases

The database is the collection of data which supports any business. This valuable commodity must be protected as much as possible and be restored to a known, stable, state when the computer facilities fail to perform normally. The database comprises not only application data, but also configuration data pertaining to: the users of the computer along with their access and restrictions; and peripherals connected to the computer. The configuration data is not part of the data resilience referred to in this document. Any changes to such data should be archived (normally during the O/S archiving procedures).

Transaction Journaling

Transaction Journaling provides the capability to prevent permanent data loss following a media or system failure. The Transaction Journal is a copy of database updates in a chronological order. In the event of such a failure, the Transaction Journal may be replayed, thus restoring the database to a usable state. Transaction Journaling preserves the integrity of the jBASE database by ensuring that logically related updates are applied to the database in their entirety or not at all.

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

Selective Journaling

The jBASE journal 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 mach1.temenos.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).

Resilient Files

Resilience is the ability of a file to remain uncorrupted in adverse conditions such as system failure. The implementation of resilient files is essential for warmstart recovery to guarantee recovery from failure by rolling forward from the transaction journal with or without a system restore.

If a file is structurally corrupt this will stop any database level updates being applied; preventing the possibility of a roll forward and hence invalidate the warmstart recovery. Logical database corruption will be resolved by the roll forward.

A resilient file must have a singularity of update where one disk operation cannot rely on another in a change of file structure. For this reason new substructures are built within the file before a single disk operation redirects the file to the new structure and the old structure is released.

Online Backup and Restore

The Online Backup facility has been developed to enable system managers to perform necessary regular database backups while still allowing users the ability to perform updates on the database. The three jBASE components used for this feature are jbackup; Transaction Journaling and the use of transaction boundaries. While jbackup may not necessarily be perceived as the tool of choice for database archiving, it is the only facility which may be used while the database is online. Using this facility enables the automatic restoration of the database including all updates up to the point of failure – without operator intervention or scripting. Following the restoration, the database will be left in a consistent state.

The functionality of the restore process, jrestore, has been extended to allow for the automatic roll-forward of logsets after a database restore has completed. This extension uses the Transaction Journal configuration (JediLoggerConfig) which was active at the time of the last backup along with the corresponding Transaction Journal Logfiles.

Warmstart

This facility is designed to enable the databases defined by the administrator to be brought back to a stable, working position, following a power failure. Without this it is not clear whether all transactions have been committed to the database following such events. Databases which have been shutdown prior to the power outage will not require recovery, so recovery is not attempted on them. Those databases which were active at the time the computer lost power will be recovered. This recovery will take the form of a database roll-forward of all complete transactions. A complete transaction is deemed to be one which has entered the commit phase of processing. Those transactions which were incomplete will not be recovered at all. The databases will be left in a consistent state following recovery. It is the database administrator's responsibility to determine which transactions require re-entry.

Transactions

Database transactions are a group of logically related file updates. These updates are intended to be processed as a whole. In order to maintain database consistency, all of the updates within a transaction must occur or none of them.

For instance, assume there is a standing instruction where in the bank needs to debit customer A's account with USD500 and credit customer B's account on account of house rent payable by A to B. This has to happen on the 1st day of each month. In the above-mentioned transaction, two important tasks need to be carried out.

One is a debit to Customer A's account with $500

Another is a credit to customer B's account with $500

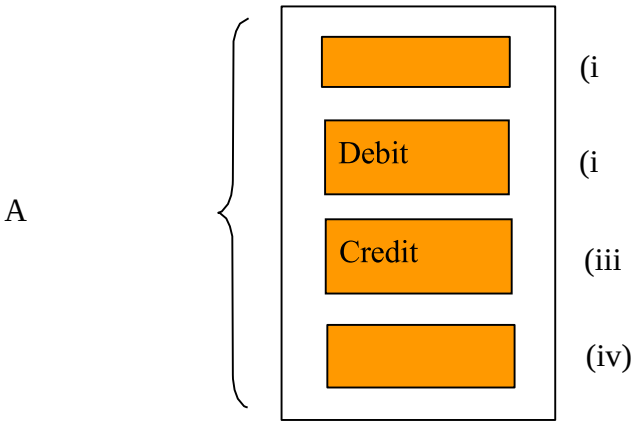
It is vital that both the above-mentioned tasks happen or none of them to happen. If either of them only happens, it would lead to database inconsistency.

Transaction Boundaries and Locking

Updates within a transaction block normally require database locks to be taken to prevent inconsistencies arising from different users simultaneously updating the same record. It is clear that if these locks are held for an extended time, then access to the database may be impaired. It is vital, then, that locks are only held when about to write to the database and certainly not when waiting for user input.

Transaction Processing within Transaction Journaling

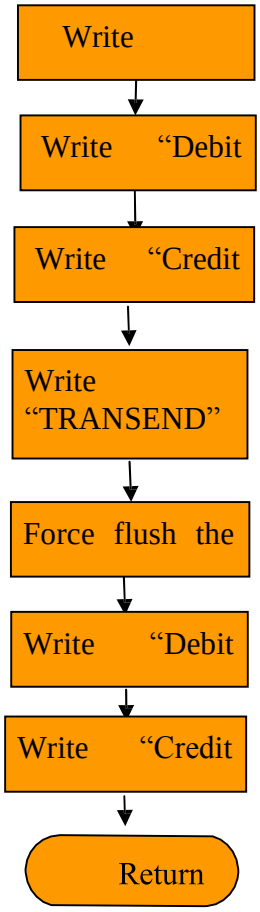
As indicated earlier a transaction is a group of logically related updates to jBASE files. Take the banking example and introduce transaction boundary markers:



(i) The jBC command - "TRANSTART …." when executed by the jBC runtime system causes a "TRANSTART" record to be placed in the transactional cache for this user.

(ii) & (iii) "WRITE" records containing not only the record data, but details about the origin of the updates (see Viewing the Transaction Journal in a later section. These journal records are cached following the "TRANSTART" record.

(iv) The jBC command - "TRANSEND …" causes the process to enter the "COMMIT" phase of execution. Up to this point no data has been written either to the Transaction Journal or to the database. The following procedure is then followed:



Note: Any Journal writes which fail causes a failed message to be displayed and recorded in the

This is configured as the default action – can be

DATABASES

Concept

Until recently there has been no way or to manage and control subdivisions of the application – departmental control or to duplicate the application/DBMS to support more than one instance of the database – multi-customer hosting. The database grouping is achieved by the used of the JBASE_DATABASE environment variable;

not specifying this will result in the user being assigned to the "default" database group. This allows the system administrator to control access to various populations of user/applications, without affecting the other users/applications.

Departmental Control

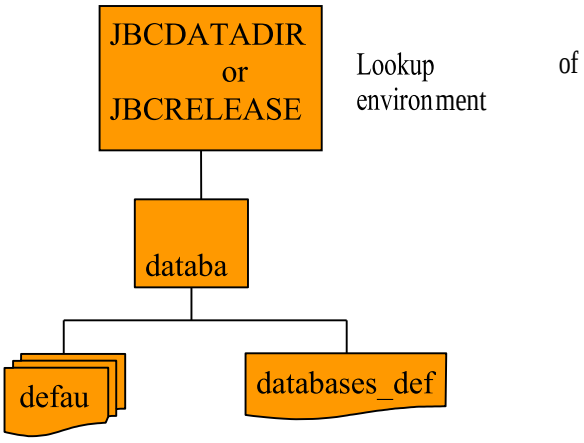
Users may now be assigned a target grouping or "database" when they access applications. This "database" enables finer control over which groups of users may access the database. This grouping is likely to be on a departmental or functional basis, i.e. users may be assigned to the "Sales" or "Accounts" database or even the "Administrators" database. This physical database may thus be physically or logically split by functional areas. Control of each of these areas is by the assigned "database" name. Thus it is possible to restrict access to the database to only, say, those users who are in the "Administrators" database group etc. The database could be designed such that each functional area contains files pertinent to each area and that files which are shared between functional groups are stored in a central repository, with access available to all.
It should be stated that this "database" grouping is not intended to replace file ownership and access permissions which are normally in existence.

Multi-customer hosting

An application could be replicated such that a provision is made to support multiple customers each running the same application but with each having their own copy of database files. In this instance the "database" grouping could be by customer, thus allowing control over each distinct customer database.

Configuration

The default configuration of databases is as follows :



There will always be a "default" database file and a "databases_defined" file defined within the system. In order for the system to run, the environment variable

“JBCRELEASEDIR” must exist in order to find where jBASE resides. This will be the default entry within the “databases_defined” file. Each defined database will contain information about where to find Transaction Journaling configuration and administration files. This will be defaulted to JBCRELEASEDIR location if not specified.

Environment Variables

Two environment command may be used to assign a user or application to a particular database and to use a particular set of Transaction Journal files:

```
set JBASE_DATABASE=”dbname”
```

will assign this user/application to the specified database.

```
set JBCLOGCONFDIR=”path of TJ configuration”
```

will assign Transaction Journal log files as defined in the configuration files found at this location. This command is not mandatory. If not used the default location will be used.

Database Control Commands

Command	Options	Operation
DB-START	-n	Database name – “default” if not specified
	-t	Location of TJ configuration – defaults to $JBCRELASEDIR/config
DB-PAUSE	-a	Administrators are still allowed access to the database
	-n	Database name – “default” if not specified
	-r	Read type operations are still allowed on the database
	-t	Transactions are allowed to complete
DB-SHUTDOWN	-a	All databases
	-n	Database name – “default” if not specified
	-t	Transactions are allowed to complete
DB-RESUME	-n	Database name – “default” if not specified
DB-REMOVE	-n	Database name – “default” if not specified
DB-STATUS	-a	All databases
	-n	Database name – “default” if not specified
	-t	Display users inside a transaction

-v Verbose mode

-w users currently waiting for DB-RESUME

-V Very verbose mode

DB-START

Syntax

```
DB-START {-nt}

      where:

      -n    Database name
      -t     TJ Logger Config Base
```

This command is used to start a database. Upon completion, users/applications which have been configured to use this database, may then do so. Prior to this point the following message will be displayed to the user/application :

jBASE: Database not started

The use of a particular database is trapped very early on in the creation of an application process. If the expectation is that the database should be available for use, then the system administrator should be contacted for resolution.

The DB-START command not only control access to a particular database, but is used also to define the location of the configuration files for Transaction Journaling operations. If the "-t" switch is not used, then the default location ($JBCRELEASEDIR or %JBCRELEASEDIR% for Windows platforms, will be used to record the location of the "config" directory/folder. This information is used by the "Warmstart" facility in order to provide recovery in case of power failure.

The DB-START command will write two entries into the "databases" directory/folder:

The first file will be named after the Database name as specified by the "-n" switch. If there is not database name specified, then this will default to the creation of a file called "default". This file is used to hold the status of the database. This file will contain the following identifier :

JBC DB

The remainder of the file contains information about the database itself, notably the state of the database.

The second file involved in recovery is the "databases_defined" file within the "databases" directory/folder. If this file does not exist, then it will be created during the execution of the DB-START command. Each entry within the "databases_defined" file will take the following form:

database_name $JBCRELEASEDIR TJ config location
or
%JBCRELEASEIR% (Windows)

e.g. for Unix/Linux platforms :

```
default /home/jbase     /home/tjlogs
Sales   /home/jbase     /home/SalesLogs
Accounts        /home/jbase     /home/AcctsLogs
HR      /home/jbase     /home/tjlogs
```

And for Windows platforms:

```
Default C:\jbase4.1     C:\jbase4.1\tjlogs
Sales   C:\jbase4.1     C:\SalesLogs
Accounts        C:\jbase4.1     C:\AcctsLogs
HR      C:\jbase4.1     C:\jbase4.1\tjlogs
```

Note: Each field is separated by a space character.

The above example shows a sample configuration. The databases “default” and “HR” will both use the default configuration for Transaction Journaling, whereas “Sales” and “Accounts” will each have their own set of Transaction Journal log files. All databases will use the same set of jBASE executables.

DB-PAUSE

Syntax

```
DB-PAUSE {-anrt}

      Where:

      -a    Administrators are still allowed access to the
      database    as such must be used with care.
      -n   Database to pause
      -r    Read type operations are still allowed on the
      database.   Write operations including DELETE-FILE,
      FILELOCK,   CLEARFILE, WRITE and DELETE record will be
      paused.
      -t    Transactions are allowed to complete.
```

The DB-PAUSE command is used when the administrator wishes to selectively pause the named database. The pause effected by this command prohibits all access to the database from this time, dependent on the options chosen. Processes will wait until this condition is cleared from the database , with no application programming required to effect this wait.

Briefly, database transaction is defined so :

TRANSTART
READs
WRITEs
DELETEs
Etc.
TRANSEND

When the TRANSEND instruction is executed, this process is now deemed to be "in a transaction" for database purposes. No database updates have occurred up to this time, but they are cached. The "–t" option refers to those processes which have entered this state. Once a transaction has been processed fully this state is exited. The process will now be paused – depending on other options chosen.
Note: A complete description of the life of a transaction will be documented later.

DB-SHUTDOWN

Syntax

```
DB-SHUTDOWN {-ant}

      Where:
      -a    All Databases
      -n    Database name
      -t    Transactions are allowed to complete
```

This command will allow the system administrator to shutdown databases in an orderly manner. This allow for a clean system shutdown,
ensuring database integrity. The effect on processes is the same as for DB-PAUSE.

DB-RESUME

Syntax

```
DB-RESUME {-n}

      Where:
      -n    Database name
```

This command will set the specified database to active – no restrictions on update will be in effect.

DB-REMOVE

Syntax

```
DB-REMOVE {-n}

      Where
      -n    Database name
```

This command will remove the specified database from the databases directory/folder. If the defined database is the “default” database, then this is ignored, else the database definition is removed from the “databases-defined” file. The “databases-defined” file is used by the WARMSTART utility when recovering a database following a power failure.

DB-STATUS

Syntax

```
DB-STATUS {-ntvwV}

      Where:
      -a    All Databases
      -n    Database name
      -t     Display users inside a transaction.
      -v    Verbose mode.
      -w    Display users currently waiting for DB-RESUME.
      -V    Very verbose mode.
```

This command allows the system administrator to determine the state of each defined database. The following will show various states of each of the defined databases (e.g.).

e.g.1 – No options declared – default database to display is then “default”

:DB-STATUS
Database 'default' is active , resumed at Tue Oct 03 12:54:16 2006

e.g.2 – Defined database set to different states – description is self-evident as to the state of each.

Database 'default' is active , resumed at Tue Oct 03 12:54:16 2006

Database 'HR' paused at Tue Oct 03 13:04:08 2006
For READ and WRITE operations
Administrators still have full database access
Transactions will be blocked immediately

Database 'Accounts' paused at Tue Oct 03 13:02:35 2006
For WRITE only operations
Administrators still have full database access
Existing transactions can continue until complete

Database 'Sales' paused at Tue Oct 03 13:02:57 2006
For READ and WRITE operations
Updates are denied also to administrators
Existing transactions can continue until complete

TRANSACTION JOURNAL CONFIGURATION AND ADMINISTRATION

The jBASE Transaction Journal configuration comprises these essential components :

jediLoggerConfig

This file is the repository for all configuration and operational details concernin Transaction Journaling. The default location of this file is at $JBCRELEASEDIR (or %JBCRELEASEDIR% for Windows computers). For a system with one active Journal, this will be the location of its configuration. If other system topologies require separate Journaling facilities (for separate databases?), then the environment variable JBCLOGCONFDIR is used to identify the location of such configurations.

jediLoggerAdminLog

This file contains logged data regarding the running of Transaction Journaling. The details of this file refer to changes to the Journaling configuration as well as error/warning messages generated bt the Journaling system.

jediLoggerTransLock

This file is used by the Journaling system to act as a lock table during checkpointing. No user information is contained therein.

Configuring Transaction Journaling

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 Display

The first execution of jlogadmin will display the following screen:

```
                jBASE Transaction Journal Configuration

Status :            INACTIVE          Current switched log set :    0
Extended records :    OFF               Time between log file syncs : 10      Time between log file checkpoints : 10
Log notify program :  (undefined)
Warning threshold :   70  % , thereafter every  1   %   or  300    secs
Sync Transactions :   ON                Encrypt Records :     OFF
                File definitions for log set  1
1 :                             2 :
3 :                             4 :
5 :                             6 :
7 :                             8 :
9 :                             10:
11:                              12:
13:                              14:
15:                              16:


Use the cursor keys and tab keys to move around the screen. The fields can be
Press RETURN to continue :
```

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 and the login name, will be added in the jBASE transaction journal for each update.

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.

Time between log file checkpoints

Specifies the number of minutes between system checkpoints. After a transaction has completed, this time is checked. If expired, then a system checkpoint is performed.

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/switchlogs "%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"
```

An example of a log notify program, "switchlogs" may be designed to allow automatic switching of logset when the warning threshold is reached:

```
0001 REASON = FIELD(SENTENCE(1),":",1)
0002 PROCESS = SENTENCE(2)
0003 MESSG = TRIM(SENTENCE(3))
0004 IF REASON # "WARNING" OR MESSG[1,21] # "Journal Log Files now" THEN
0005    STOP
0006 END ELSE
0007    EXECUTE "jlogadmin -lnext" CAPTURING OUTPUT
0008 END
0009 END
```

The program identified by the "log notify program" is called each time that a message is entered into jediLoggerAdminLog. It is the responsibility of the called program to deal with the reason for the message being entered. The function SENTENCE returns information from JediLoggerAdminLog about the latest entry.

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.

Sync Transactions

An option "SYNC" exists for the TRANSTART command which will force-flush the database and journal following a transaction commit. The option in jlogadmin allows for this option to be invoked globally. If "Sync Transactions" is set to "on" then all committed transactions will cause the force-flush. If set to "off" then committed will not automatically force-flush the database and journal unless the "SYNC" option is present in individual TRANSTART commands.

Encrypt Records

The transaction journal is not normally encrypted. This option will allow the data content of each record to be encrypted on disk. The data content of each record will be encrypted with an (internally-specified) industry-standard encryption scheme, using an internal key. The record headers remain unencrypted so that all utilities accessing the journal will be unaffected.

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:

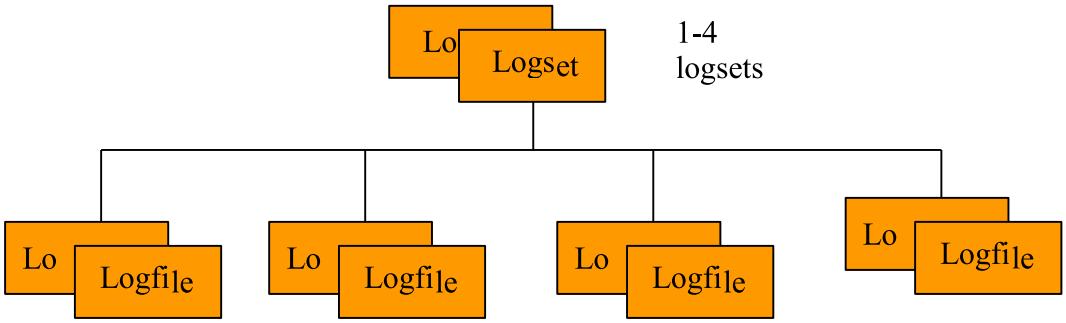
SYNTAX ELEMENTS

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)
-d[1-4]	Delete logset
-f set,fileno,file	Change log filename in log set where Set Logset fileno File number File File name
-h	Display help
-i[1-4],filename{,filename...}	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
-j pwd	Password protect journal with “pwd”
-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
-m	Set to encrypt all records “ON” or “OFF”
-n program	Set threshold notify program.
-o	Perform operation without checking if the specified log set is empty. Used with -f and -t.
-p	Checkpoint every “nn” minutes

-r	Set the sync all transactions to “ON” or “OFF”
-s secs	Set synchronization period
-t	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
-I	Display the statistics if a log set using the embedded information
-V	View transaction journal administration log file jediLoggerAdminLog

Defining Logsets

The following diagram illustrates the constituent parts of a Transaction Journal installation



Each logset should, ideally, be defined within a separate filesystem/partition. The definition of the logset can either be the root of such a filesystem/partition or some sub-directory therein. Each logfile within such logsets are special files; the implication of this is that they should not be created/restored without using the

jlogadmin utility. N.B. This will not only create the files where specified but will also enter such configuration in the jediLoggerConfig file.

Example: Creating 2 logsets with 3 logfiles in each

Using the interactive display:

```
                jBASE Transaction Journal Configuration

Status :            INACTIVE          Current switched log set :    0
Extended  records :     OFF                  Time between log file syncs :    10        Time
between log file checkpoints : 10
Log notify program :  (undefined)
Warning threshold :   70  % , thereafter every  1   %   or  300    secs
Sync Transactions :   ON              Encrypt Records :     OFF
                File definitions for log set  1
1 : e:\logset1\logfile1              2 :
3 :                                  4 :
5 :                                  6 :
7 :                                  8 :
9 :                                  10:
11:                                  12:
13:                                  14:
15:                                  16:

e:\logset1\logfile1: No such file or directory
jlogadmin: File 'e:\logset1\logfile1' cannot be opened.
Do you want to create this file (Y/N) ?
```

Note that because the logfile does not exist, the operator is asked to create. If e:\logset 1 does not exist, then a message will be displayed:

```
e:\logset1\logfile1: No such file or directory
jlogadmin: File 'e:\logset1\logfile1' cannot be opened.
Do you want to create this file (Y/N) ? y
e:\logset1\logfile1: No such file or directory
```

Following the completion of the creation of this logfile, the operator moves to the next file definition (2) by tabbing to the
next field. When all three files have been created in this way, by entering the cursor key several times, the log set number is changed from a “1” to a “2”. The same procedure may now be followed to create the logfiles for logset 2.

The following command lines may be used to create both logsets thus:

```
jlogadmin -c -f1,1,e:\logset1\logfile1
jlogadmin -c -f1,2,e:\logset1\logfile2
jlogadmin -c -f1,3,e:\logset1\logfile3
jlogadmin -c -f2,1,f:\logset2\logfile1
```

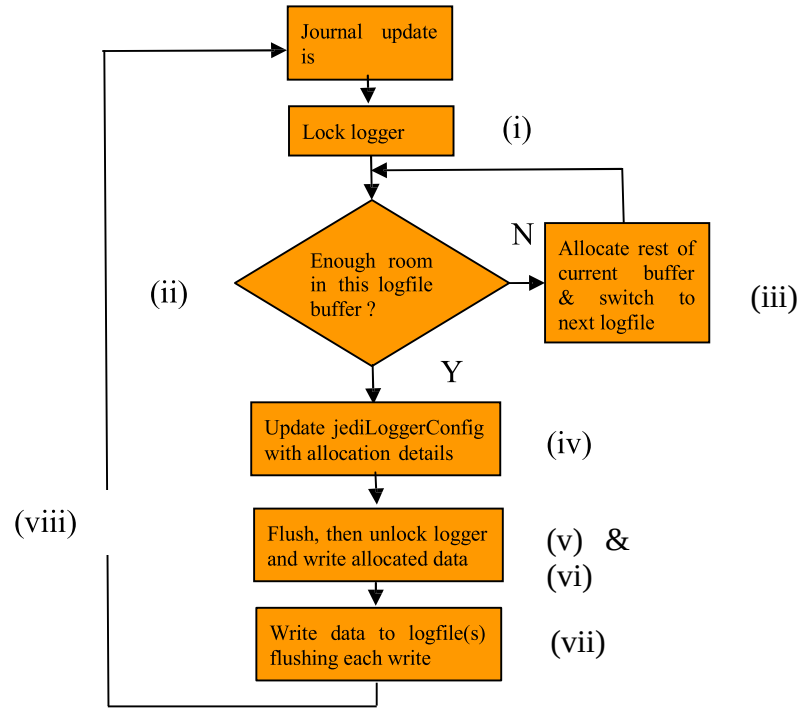
```
jlogadmin -c -f2,2,f:\logset2\logfile2
jlogadmin -c -f2,3,f:\logset2 \logfile3
```

If the "-c" option is omitted then the files will be created automatically without prompting the operator. The same caveat still exists – if the logset directories do not exist then the commands will fail.

A batch file is usually created to perform logfile creation.

Use of logsets and logfiles

Each logfile is designated as having a maximum size of 2Gb on all platforms: with a maximum population of 16 such files per logset, this make the maximum size of each logset 32Gb. Transaction Journaling relies on a logfile not exceeding its 2Gb limit, otherwise Journaling will be suspended automatically. In order to achieve the 32Gb size per logset the following procedure is followed internally by the Journaling routines:



Notes:

(i) Access to the logger is restricted to this process. Other process will wait until unlocked.

(ii) Writes to a logfile are contained in 4k blocks. If the record (and associated update information) can fit into the current block, then it is allocated as much space as it needs in that block.

(iii) If the current block is too small then the remainder of the block is allocated; the next logfile is selected and the test for fit is repeated.

(iv) Once all the requested size of update has been allocated, the configuration file is updated with details of: which logset to update next; which file in that logset and the offset in that file to write the next data.

(v) The logger is now unlocked, the next process may now allocate space in the logger.

(vi) The space has been allocated in the logger, this process may now write the updated record data to the assigned space. This allows for a rapid throughput of logger space allocation, while allowing asynchronous writes to the logger.

(vii) The process can now write to the logger asynchronously, knowing that the allocated space cannot be written to by other processes. The use of asynchronous writes is vital during writes of large updates. Note that utilities which access the logfiles directly are aware of this situation and will retry the reads of the logger until a complete record is read.

(viii) The next process requiring writing to the logger may now do so.

Observing the use of the buffers in step (iii), the writes to the logfiles contained in a logset is in a "striping" manner. The file space initially used when creating a logfile is approximately 4k. As allocated buffers in a logfile as used, then the logfile grows accordingly. So if one logfile were to be allocated to a logset, then once the 2Gb limit was reached, then Transaction Journaling would be suspended. Now if (say) 16 logfiles are allocated in a logset and we use the "striping" of each file to contain data in a round-robin basis, it can be seen that if the first logfile allocated exceeds the 2Gb limit, then each of the other logfiles in the logset would be almost to that limit. So Transaction Journaling would be suspended after almost 32Gb of journal information has been stored.

Logset Switching

As stated above, there can be up to four logsets defined. The number of logsets which need to be defined is dependent on particular system operation requirements.

Single Logset

Transaction Journaling may be run with a single defined logset. The expected maximum size that the logset will reach must be calculated. This depends on the following factors: the transaction frequency and the average size of transactional data. If the intended backup regime is daily, then the logset will be constrained to 32Gb. This is unlikely to be a limiting factor for most installations. The backup strategy is forced to be off-line; all outstanding transactions must complete prior to

the backup being taken. Following this backup, the logset can itself be archived and then the logset re-initialised back to an empty state.
An example of a command to use to achieve this is (assuming that logset 1 is defined):

```
jlogadmin -l1 -aon
```

This command switches to logset 1 (make current) and then sets Transaction Journaling active.

or alternatively, if journaling already active:

```
jlogadmin -lnext
```

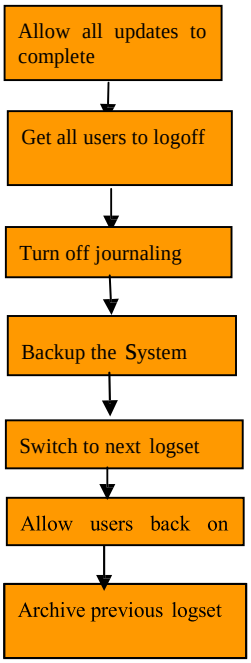
These commands take advantage of the fact that when a logset is re-used, it is automatically truncated to an empty state.

The system is now usable again for multi-user operation.

Note: The "Log notify program" may be used to automate this switching as previously described.

Multiple Logsets

The normal configuration for Transaction Journaling is to use at least two logsets. If the maximum logset usage between backups is greater than 32Gb (?), then multiple logsets will have to be defined to increase this capacity. This is not the normal case. Multiple logsets are normally used so that the updates since the penultimate backup are preserved. This has two benefits: if there is a problem with the last backup, the administrator has the option of recovery to the previous backup, followed by a roll-forward of the transactions since that backup. The operation is then:



(i)

(ii)

(iii)

(iv)

Notes:

(i) This is done to ensure, positively, that no more updates are added to the Journal

(ii) This logset will be set to an empty state automatically

(iv) This archive would be used if the backup at (ii) failed and the previous backup used instead for recovery.

The Transaction Journal/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
27	SOURCE	Source Program Name
28	LINENO	Program Line Number
29	REALTIME	Program Update Time
30	TRANSTIME	Transaction Elapsed Time
31	TRANSCOMP	Number of Completed Transactions
32	TRANSABORT	Number of Aborted Transactions
	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
CHECKPOINT	Checkpoint Marker

Selective Restores

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.

Note: This type of operation must be used with great care! It is highly possible that the database may be left in an inconsistent state ifs an individual file is rolled-forward. If transactions contain updates to more than one file (the normal case), then regard must be paid to other file updates occurring within those transactions in order to maintain database integrity.

Monitoring Transaction Journaling

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:     18:15:42 21 DEC 2006 , by colins from port 10
Configuration file modified: 18:18:37 21 DEC 2006 , by colins from port 11
Journal file sets switched: 18:15:50 21 DEC 2006 , by colins from port 10 Full
log warning threshold: 10 percent , thereafter every 1 percent or 5 secs Journal
files synced every:     10 seconds , last sync 18:55:40 21 DEC 2006
Journal checkpointed every: 10 minutes , last checkpoint 18:20:47 21 DEC 2006
Background sync demon:      ACTIVE at process id 14655
Extended record:             OFF
Encrypted records:          OFF
Transaction Sync:           ON
Admin log file:             297 entries , in file
/home/colins/4.0_rels/jbcdevelopment/config/jediLoggerAdminLog
Admin log notify program:   (undefined)
Current log file set:        1 , date range 18:15:50 21 DEC 2006 to 18:21:24    21
DEC 2006
/home/colins/logdev1:       86.37% capacity
Total record count:          92
Total byte count:            6,609,649
jlogdup program status:     NONE active
Status log set 1 (current):  1 files , 92 records , 6609649 bytes used
                            Date range 18:15:50 21 DEC 2006 to 18:21:24 21
DEC 2006
                            Not Archived
                             /home/colins/logdev1 , created 18:15:42 21 DEC 2006
, 86.37% capacity

Status log set 2:           No files defined
Status log set 3:            No files defined
Status log set 4:           No files defined
Status log totals:           1 files , 92 records , 6609649 bytes used
                             Date range 18:15:50 21 DEC 2006 to 18:21:24 21
DEC 2006
Committed transactions:      7
Aborted transactions:        0
Total transaction time:     0.42 Secs
Average transaction time:   0.0600 Secs
```

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, thereby alleviating 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.
Note: This command is not available for Windows platforms.

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

Options –i and –b – starting jlogsync

The most common way of starting jlogsync is by using the "-i" and "-b" options. This will stat the process in the background. The command will be typically be used in a machine startup script, prior to allowing multi-user mode.

```
jlogsync -ib

jlogsync:   Started on pid 4030
```

Option d – displaying status of jlogsync

Standard display is obtained thus:

```
jlogsync -d

jlogsync:          $Revision: 3.3 $

Program at pid:    4030

Program started:   Wed Dec 20 11:12:17 2006

Memory:                  39928 bytes used , 95240 bytes jfree

CPU usage:         0.00 usr , 0.03 sys
```

Option k – kill the jlogsync daemon

The daemon may be killed by the administrator by the use of the ”-k” option. No message is displayed unless the kill fails in which case “kill” will be displayed.

Option tnn – set the jlogsync inactivity timeout

When jlogsync is initialized, a default inactivity timeout count of 5 minutes is set up to determine whether the daemon is still working correctly. If this time does expire and the daemon has not done anything in the meantime, it is deemed at this point that the daemon has died prematurely. The “-tnn” option allows for an inactivity timeout period of “nn” seconds. This value can be any values greater than 60 seconds (despite the “nn” description).
If the inactivity timeout period is reached then a message is displayed:

```
Sync demon appear to have died prematurely
```

Option v – verbose output

When this option is used details of the last sync events are displayed along with details of the inactivity timeout and logset warning values.

```
jlogsync -vd
jlogsync:              $Revision: 3.3 $
Program at pid:        15859
Program started:       Thu Dec 21 21:19:40 2006
Memory:                39944 bytes used , 95224 bytes jfree

CPU Usage:              0.03 usr , 0.55 sys
Last forced sync:      Thu Dec 21 21:28:55 2006
Last check time:       Thu Dec 21 21:28:59 2006
Inactivity timeout:    100 seconds
Warning threshold:     70 initial percent
                       1 increment percent thereafter
                       300 increment seconds thereafter
```

Option S – force a sync. now

This option is used to force-flush the journal.

jlogmonitor

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).

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
-Inn	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.

EXAMPLES

-Cnn

A monitor may be set up which will display a message once the warning threshold (as defined in jlogadmin) has been reached. The monitor will then wait until the percentage full has increased by 1% at which point a new message indicating this is displayed. This will continue indefinitely (or until aborted).

jlogmonitor -C10
09:43:30 14 DEC 2006 Journal File System capacity exceeds 10% , actual 89%
09:46:30 14 DEC 2006 Journal File System capacity exceeds 10% , actual 90%

–ccmd

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.

jlogmonitor -C91 -c"jlogadmin -lnext"

-Dnn

This option allows the operator to monitor any jlogdup processes which may be running. If there is no activity for the specified time, then an error message is displayed. Note that this command will report inactivity for all running jlogdup processes. It is not possible to specify one of many jlogdup processes to monitor.

jlogmonitor -D1
10:09:25 14 DEC 2006 No reported activity from the jlogdup programs
10:10:25 14 DEC 2006 No reported activity from the jlogdup programs
10:11:25 14 DEC 2006 No reported activity from the jlogdup programs

–E

If one or more jlogdup processes are reporting errors, jlogmonitor may be used to display this condition. The process will interrogate all running jlogdup processes for erros which have been encountered. If any are reporting errors a message similar to the following will be displayed:

jlogmonitor -E
15:52:52 18 DEC 2006 jlogdup is reporting 2 errors

Further information about any such errors can be found on those screens running the jlogdup processes which are reporting errors.

-Inn

If journaling is suspended or stopped for any period, jlogmonitor may be used to trap such occasions. The “nn” parameter is in minutes, so if journaling is stopped/suspended for more than this time a message to that effect will be displayed.

jlogmonitor -I1
16:08:31 18 DEC 2006 The status of the logger is not active
16:09:31 18 DEC 2006 The status of the logger is not active

This will be repeated every “nn” minutes or until aborted.

–Snn

This option is similar to the “-I” option, but will display a message if no updates have been made to the journal for “nn” minutes.

jlogmonitor -S1
16:13:07 18 DEC 2006 No reported activity being applied to the journal log sets
16:14:07 18 DEC 2006 No reported activity being applied to the journal log sets

The options may be combined on the command line to trap any or all of the possible conditions described above.

So,
jlogmonitor -S1 -I1

may display:

16:14:25 18 DEC 2006 The status of the logger is not active
16:15:25 18 DEC 2006 The status of the logger is not active
16:15:25 18 DEC 2006 No reported activity being applied to the journal log sets
16:16:25 18 DEC 2006 The status of the logger is not active
16:16:25 18 DEC 2006 No reported activity being applied to the journal log sets

which indicates the reason for there being no updates to the logger.

Managing logsets

jlogdup

The jlogdup command provides the capability to duplicate transaction log set data from the jBASE Transaction Journal. The transfer may be in the simple case an archive of the Transaction Journal to an external device or may be used in a combination of transfers to produce a “hot standby” machine. The whole or part of a transaction logset may be transferred, either following a jBASE SELECT statement or by specification in the jlogdup command line. The transfer process(es) may be monitored utilising a comprehensive range of dynamic statistics.

SYNTAX

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

An “input specification” consists of a source device for the transfer with optional run-time parameters and an “output specification” consists of an output device and associated optional run-time parameters. The “Options” parameters are generally used to display/record information about the transfer overall.

SYNTAX ELEMENTS

Options

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
-V	display verbose help screen
-H	display verbose help screen

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
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
terminate=waiteos(I)	switch to elder log sets as required and wait for new updates until logset switched, then terminate
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

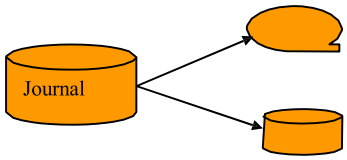
The time specification, used in the ‘start=’ and ‘end=’ specification can be one of the following formats:

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
checkpoint	use last checkpoint time as start of transfer time

Examples of use

In order to expand on the description of each of the many specifications and options, a series of example usages will be used for illustration.

Example 1: Archiving Journal to Off-line Media



e.g.:
/dev/rmt0

e.g:
E:\jrnl_save

Input and Output Specifications

If the Journal, depicted above, contains 4 logsets: logset1-4 and logset2 is the active logset, then a snapshot of this logset may be made to either a real tape drive e.g. /dev/rmt0 on AIX or a tape image file E:\jrnl_save on Windows, so:

current or 0-4

The **current** logset refers to that logset which is selected for use at this time within transaction journaling. This logset may be active or inactive. Logset **0** is a special case and means that there is no logset currently being used at this time. It is possible to define up to 4 logsets and the number **1-4** refer the specific logset.

```
jlogdup input set=current output set=serial device=/dev/rmt0
```

or

```
jlogdup input set=2 output set=serial device=E:\jrnl_save
```

It can be seen that the input set, whether specified as current or 2 refer to the same logfile data and that this is the source of the transfer.

eldest Logset

Logsets may have been switched since the last backup, so the updates made to the journal may exist in more than one logset. Consider that the data in logset1 contains the oldest data and logset2 contains the more recent, a command such as:

```
jlogdup input set=eldest output set=serial device=/dev/rmt0
```

This will take all the data in logset1 and all in logset2 (to this point) and output to the destination as specified by the "output spec".

blocksize

The output specification indicates where to put the logfile data. The size of the blocks written to a tape device can be specified using the blocksize parameter, thus:

```
jlogdup input set=current output set=serial device=/dev/rmt0
blocksize=16384
```

blockmax

In the likelihood that the tape capacity is less than the journal size, another parameter, "blockmax" may be used to specify how many blocks (as specified by "blocksize") may be written before the media is required to be changed.

```
jlogdup input set=current output set=serial device=/dev/rmt0
blocksize=16384 blockmax=200000
```

Multiple Devices

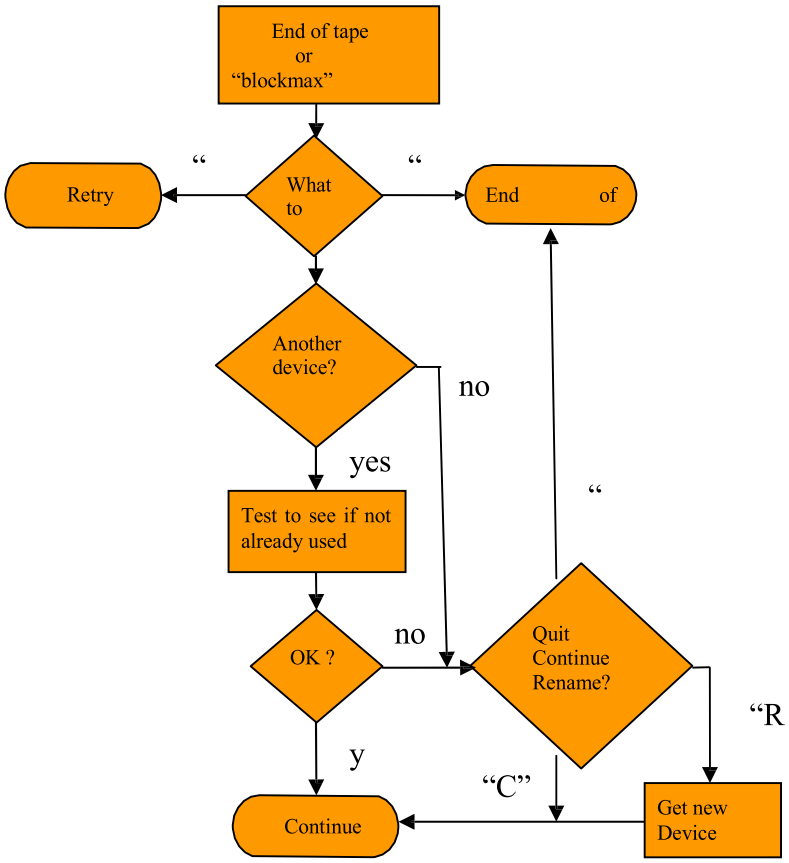
When using tape devices it is possible to specify multiple devices so that in the event of media overrun, the jlogdup operations may continue without intervention

```
jlogdup input set=current output set=serial device=/dev/rmt0
device=/dev/rmt1 device=/dev/rmt2 ...
```

When the end of the tape on /dev/rmt0 is reached (or blockmax is reached), operations will automatically continue on /dev/rmt1, and so on. A check is made that

the media being used is not reused from the same jlogdup operation (by timestamps). If there is a conflict, user intervention is required.

The following diagram depicts the flow at end of media.



If user intervention is required then the user may "R"etry the write "Q"uit the jlogdup operation, or "N"ext device. If "N" is entered, then if the number of devices specified in the command is greater than 1, then the next in sequence in the command is used. If all else fails the user is asked to "C"ontinue, "Q"uit or "R"ename. The rename option allows a different device to be specified. After this prompt no further checks are made, automatic checking will be over-ridden.

prompt=true

If no automatic cascading of tapes is desired the use of "prompt=true" on the command line will force operator intervention:

```
jlogdup input set=current output set=serial prompt=true
device=/dev/rmt0 device=/dev/rmt1 device=/dev/rmt2 ...
```

Verifying a logging tape

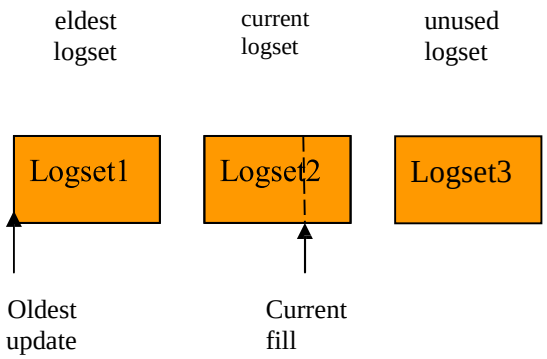
A tape holding journal data may be verified for readability and for correct formatting for such a tape. An example of this could be :

null specification

```
jlogdup input set=serial device=/dev/rmt0 output set=null
```

Terminating jlogdup

Firstly, a diagram:



These definitions will assist in explanation of how jlogdup is terminated.

Taking the earlier example :

```
jlogdup input set=current output set=serial device=/dev/rmt0
```

This command will terminate when the end of the current logset (i.e. 2), is reached (unless terminated externally). This will only give a partial snapshot of the journal.

terminate=eos, terminate=eof

These specifications are normally used when more than one logset is defined and

more than one logset contains valid logfile data; as above.
The command:

```
jlogdup input set=1 terminate=eos output set=serial
device=/dev/rmt0
```

will backup all entries in the journal from the start of logset1 (the eldest) up to the last update in logset1 – no further updates will be saved. Again this is only part of the information required to recover all of the data.

And the command:

```
jlogdup input set=1 terminate=eof output set=serial
device=/dev/rmt0
```

or

```
jlogdup input set=1 output set=serial device=/dev/rmt0
```

will take all updates from the start of the journal and output to the tape all records up to and including the last update on set 2, the current logset. Note that omitting "terminate=??" will default to "EOF" the end off all logset containing valid data.

terminate=wait

What is more normal is that we want to transfer from the beginning of all logsets, transfer all the logfile data and the wait for new updates, then transfer them (e.g.) to tape as they arrive. Thus:

```
jlogdup input set=eldest terminate=wait output set=serial
device=/dev/rmt0
```

will achieve this.

terminate=waiteos

This is a combination of two previous terminations: wait and eos.

```
jlogdup input set=eldest terminate=waiteos output set=serial
device=/dev/rmt0
```

This will perform as the previous example, with the exception being that while waiting for new updates, if the logsets are switched then the jlogdup process will terminate. This may be used to trigger some batch operation, ensuring that all updates from that point will reside in another logset.

timeout

When using "terminate=wait" or "terminate=waiteos" it is possible to set a limit of the amount of time the process will wait for new updates into the journal. If the "timeout" option is missing the process will wait indefinitely, otherwise it will wait for the number of seconds dedfined in the "timeout" option.

```
jlogdup input set=eldest terminate=waiteos timeout=300 output
set=serial device=/dev/rmt0
```

The jlogdup process will wait for 5 minutes or the switching of the logset before terminating.

retry

The "retry" option is used to attempt to re-read the journal for a complete record and refers to the time delay between re-reads. A complete record may exist in the journal when the update to the journal is from a slow device (i.e. tape device) or is a large record, or a combination of both. The start of the record may have been written but the rest of the write may not yet have completed. This option allows this the operator to change the default delay time from 5 seconds to another value in seconds. The re-read is attempted 10 times and is normal operation.

The "retry" time allows the operator to override the default wait time of 5 seconds to so other value. Therefore:

```
jlogdup input set=eldest terminate=wait retry=3 output
set=serial device=/dev/rmt0
```

will wait for 3 seconds between retries.

verbose=true

This option confirms the input and responds with details about when and how the process will be terminated.

```
jlogdup -V input set=1 terminate=waiteos timeout=25 output
set=null verbose=true



Process waits for 25 secs. or until set switched, or
terminated by operator
```

timespec, start= and end=

The timespec options are optional parameters which allow the operator to start and/or end the jlogdup transfer from positions which are not at the start or end of the logset(s). The “start=” and “end=” specifications have several formats:

Firstly without time specs.:

```
jlogdup input set=current output set=null



10:49:20 20 NOV 2006 : STATUS:

    Begin jlogdup process: From set 'set=current' to  set
'set=null'

10:49:21 20 NOV 2006 : STATUS:

    Termination Statistics: usr 0.30 , sys 0.00 , elapsed
0m0.27

    INPUT : 1851 records , 0 blocks , 69495014 record bytes ,
0 errors

    OUTPUT: 0 records , 0 blocks , 0 bytes , 0 errors

10:49:21 20 NOV 2006 : STATUS:

    Program terminated. Exit code is 0
```

Using simple start time specification:

```
jlogdup input set=current start=10:36:00 output set=null



11:14:26 20 NOV 2006 : STATUS:

    Begin jlogdup process: From set 'set=current
start=10:36:00' to  set 'set=null'

11:14:26 20 NOV 2006 : STATUS:

    Termination Statistics: usr 0.30 , sys 0.00 , elapsed
0m0.27

    INPUT : 1838 records , 0 blocks , 69043664 record bytes ,
0 errors

    OUTPUT: 0 records , 0 blocks , 0 bytes , 0 errors

11:14:26 20 NOV 2006 : STATUS:

    Program terminated. Exit code is 0
```

Note the difference if record and byte counts.

Using a date specification:

```
jlogdup input set=current start=20-NOV-2006 output set=null


11:24:17 20 NOV 2006 : STATUS:

    Begin jlogdup process: From set 'set=current start=20-NOV-
2006' to  set 'set=null'

11:24:17 20 NOV 2006 : STATUS:

    Termination Statistics: usr 0.29 , sys 0.00 , elapsed
0m0.27

    INPUT : 1838 records , 0 blocks , 69043664 record bytes ,
0 errors

    OUTPUT: 0 records , 0 blocks , 0 bytes , 0 errors

11:24:17 20 NOV 2006 : STATUS:

    Program terminated. Exit code is 0
```

Using time and date specification:

```
jlogdup input set=current start=10:36:27,20-NOV-2006 output
set=null



11:27:30 20 NOV 2006 : STATUS:

    Begin jlogdup process: From set 'set=current
start=10:36:27,20-NOV-2006' to  set 'set=null'

11:27:30 20 NOV 2006 : STATUS:

    Termination Statistics: usr 0.36 , sys 0.00 , elapsed
0m0.33

    INPUT : 1838 records , 0 blocks , 69043664 record bytes ,
0 errors

    OUTPUT: 0 records , 0 blocks , 0 bytes , 0 errors

11:27:30 20 NOV 2006 : STATUS:

    Program terminated. Exit code is 0
```

Using start and end times to show all updates in 1 minute:

```
jlogdup input set=current start=10:36:00 end=10:37:00 output
set=null

11:31:44 20 NOV 2006 : STATUS:
    Begin jlogdup process: From set 'set=current
start=10:36:00 end=10:37:00' to  set 'set=null'

11:31:44 20 NOV 2006 : STATUS:
    Termination Statistics: usr 0.12 , sys 0.00 , elapsed
0m0.09

    INPUT : 540 records , 0 blocks , 20306362 record bytes , 0
errors

    OUTPUT: 0 records , 0 blocks , 0 bytes , 0 errors

11:31:44 20 NOV 2006 : STATUS:
```

```
Program terminated. Exit code is 0
```

notrans=true

Start and End times are chosen by the operator. At (probably) any specified time there are likely to be more than one transaction open (i.e. records are being updated between transaction boundaries). During normal operation, when the destination is "database" , jlogdup will alert the operator that if a record is to be transferred and that record is part of a transaction, and that the transaction start record has not been detected. This is not a fatal situation, but alerts the operator to those records so found. These records will not cause the database to be updated with their contents. These records will cause a message like:

```
16:39:54 22 NOV 2006 : ERROR:  For definition set=database

TRANSACTION violation: Originally in a transaction, but not
now during jlogdup
```

to be displayed.

If however a command like :

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

is issued, then the fact that the updates were part of a transaction is ignored and the database will be updated. This may cause the database to enter an inconsistent state. It is advisable that this option is not used without careful analysis of the outcome.

If a backup is now performed with the –sfilename option (create statistics file), and then use this as the start of jlogdup (after adding some more updates to the journal):

```
jfind CUSTOMERS -print | jbackup -sbackup stats -fbackup file


Scanned Files  : 2

Written Blocks : 9078

        Reels  : 1


141.8438 MB processed


jlogdup input set=current start=backup_stats output set=null


11:48:21 20 NOV 2006 : STATUS:

    Begin jlogdup process: From set 'set=current
start=backup_stats' to  set 'set=null'

11:48:22 20 NOV 2006 : STATUS:

    Termination Statistics: usr 0.37 , sys 0.00 , elapsed
0m0.34

    INPUT : 301 records , 0 blocks , 11281526 record bytes , 0
errors

    OUTPUT: 0 records , 0 blocks , 0 bytes , 0 errors

11:48:22 20 NOV 2006 : STATUS:

    Program terminated. Exit code is 0
```

If a file is now created (as a marker), and more records added to the journal:

```
CREATE-FILE NEWFILE 1 1

[ 417 ] File NEWFILE]D created , type = J4

[ 417 ] File NEWFILE created , type = J4


jlogdup input set=current start=NEWFILE output set=null



11:52:16 20 NOV 2006 : STATUS:

    Begin jlogdup process: From set 'set=current
start=NEWFILE' to  set 'set=null'

11:52:17 20 NOV 2006 : STATUS:

    Termination Statistics: usr 0.44 , sys 0.00 , elapsed
0m0.41

    INPUT : 315 records , 0 blocks , 11733072 record bytes , 0
errors

    OUTPUT: 0 records , 0 blocks , 0 bytes , 0 errors

11:52:17 20 NOV 2006 : STATUS:

    Program terminated. Exit code is 0
```

Described earlier in the documentation, checkpoint records are written to the journal periodically. A jlogdup transfer can be specifying the last checkpoint as the starting point within the journal. This will contain all updates since the database was deemed to be in a consistent state:

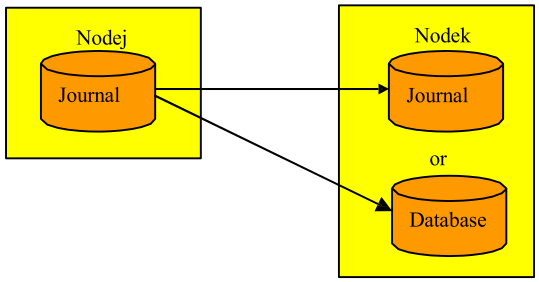
```
jlogdup input set=current start=checkpoint output set=null



11:57:48 20 NOV 2006 : STATUS:

    Begin jlogdup process: From set 'set=current
start=checkpoint' to  set 'set=null'

11:57:48 20 NOV 2006 : STATUS:

    Termination Statistics: usr 0.44 , sys 0.00 , elapsed
0m0.41

    INPUT : 313 records , 0 blocks , 11732790 record bytes , 0
errors

    OUTPUT: 0 records , 0 blocks , 0 bytes , 0 errors

11:57:48 20 NOV 2006 : STATUS:

    Program terminated. Exit code is 0
```

The end of a jlogdup process may also be specified by the last checkpoint within the journal:

```
jlogdup input set=current end=checkpoint output set=null



16:13:37 20 NOV 2006 : STATUS:

    Begin jlogdup process: From set 'set=current
end=checkpoint' to  set 'set=null'

16:13:37 20 NOV 2006 : STATUS:

    Termination Statistics: usr 0.40 , sys 0.00 , elapsed
0m0.37

    INPUT : 2167 records , 0 blocks , 81228172 record bytes ,
0 errors

    OUTPUT: 0 records , 0 blocks , 0 bytes , 0 errors

16:13:37 20 NOV 2006 : STATUS:

    Program terminated. Exit code is 0
```

Example 2: Transfers between computers

This example will be used to illustrate transfers between two computers.



Journal data may be transferred to a different computer by one of two techniques: using “stdin” and “stdout” and “rsh”. Though this method does work for Unix/Linux-based computers, because of the lack of security it is now not the

recommended method of transfer. A “socket” interface exists which allows the operator to manage the transfers more robustly.

stdout

To specify the output destination of a jlogdup transfer, a command like the following may be issued:

```
jlogdup input set=current output set=stdout
```

The output from this command is omitted as it will contain non-printable characters.

stdin, database

This specification will be used for the source end of the transfer, taking input from the pipe and outputting to the destination, in this case directly updating the local database.

```
jlogdup input set=stdin output set=database
```

rsh

To tie these two commands together it is usual to use rsh – remote shell daemon. All activity is controlled from the local host (Nodej, here) and will execute a command on the remote host to run a jlogdup process on that computer (Nodek).

```
jlogdup input set=current terminate=wait output set=stdout |
rsh Nodek /GLOBALS/JSCRIPTS/logrestore
```

This script will set up to run jBASE commands and then run a jlogdup process to update the database on Nodej.

/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
```

tty

For the specifications “stdin” and “stdout”, the specification “tty” may be used instead.

```
jlogdup input set=current output set=stdout
```

can be replaced by:

```
jlogdup input set=current output set=tty
```

and

```
jlogdup input set=stdin output set=database
```

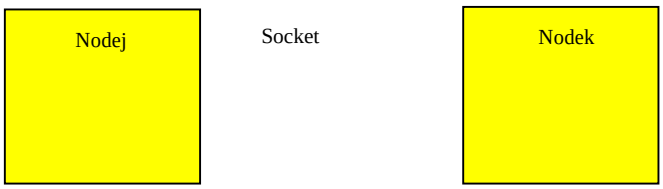
can be replaced by:

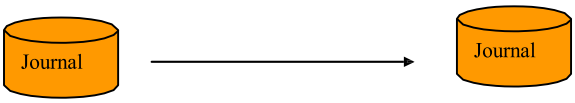
```
jlogdup input set=tty output set=database
```

Sockets

socket, hostname, port, logset

The socket interface between computers Nodej and Nodek allows the operator to set up one or more client-server relationships. On Nodej a simple jlogdup command may be set up which will take input from the specified set device and output to a socket. This socket refers to a TCP port on a destination computer (Nodek). On Nodek, again a simple jlogdup command may be set up which will take input from a socket and output the journal data to the destination device: to the current logset. This time the socket refers to a TCP port on Nodek – a listening socket.





Note: The receiving jlogdup must be set up before the sending jlogdup; failure to do this will cause the sending jlogdup process to fail with an error message :

```
No connection could be made because the target machine
actively refused it.

Unable to connect to host Nodek , error 0
```

Note2: As the output device is “logset”, transaction journaling must be set up on Nodek and active. If there is a “current” logset defined but not active, then a message similar to this will be displayed:

```
FATAL ERROR: From user jdata4.1(anon) at Thu Nov 23 11:22:18
2006

Process ID 3520 , Port 163 , tty CONIN$

    Logging is not active
```

If logging has not been set up at all, the transfer will stop immediately and a message similar to this is displayed:

```
jlogdup input set=socket hostname=localhost port=4089 output
set=logset

11:58:19 23 NOV 2006 : ERROR:

    Error: For command 'set=logset': Unable to log updates -
logging not set up
```

This message will be displayed periodically until logging is set active. If logging is subsequently made active then the transfer will complete as normal.

An example set up for Nodek could be:

```
jlogdup input set=socket hostname=Nodek port=4089 output
set=logset
```

where:

socket is the device specification.

hostname is the IP address or the DNS name of the host to use for socket transfers – in this case Nodek – this host is waiting for a connection to be made to it.

portnum is the TCP port to use for socket transfers. In this example port 4089 is chosen. This can be any unused TCP port (and therefore must be decided for each system).

Once Nodek has been set up, Nodej can be set up thus:

```
jlogdup input set=current output set=socket hostname=Nodek port=4089
```

This will connect to the jlogdup process running on Nodek, transfer all the journal data in the current logset and then terminate. The termination of the jlogdup process on Nodej will cause the jlogdup process on Nodek also to terminate.

The command :

```
jlogdup input set=current terminate=wait output set=socket hostname=Nodek
port=4089
```

Will now connect, transfer all the journal data from the current logset and then wait for new updates, transferring the updates as they arrive. This process will not terminate and will thus keep the socket open for transfers to Nodek.

```
jlogdup input set=socket hostname=Nodek port=4089
terminate=wait output set=logset
```

This command will listen for a connection, then receive journal updates and output to the current logset. If the jlogdup process on Nodej terminates, then this process

will also terminate that connection and will return to listening for a new connection, and so on.

If “terminate=wait” is present on both ends of the socket then this will form a continuous client-server mechanism.

Note:
If the “timeout” option is used on either end, then the operation will perform as expected, except for one instance. If the receiving end of the socket (on Nodek) is terminated by the operator, then the sending jlogdup process on Nodej may be sending journal data or be waiting for new updates.
If sending data, then the forced closure of the socket will force the termination of the sending jlogdup process and display an error message of the form:

```
An existing connection was forcibly closed by the remote host.

14:43:22 21 NOV 2006 : ERROR:  For definition set=socket
hostname=localhost port=4089

    Error number 22 writing to socket

0*0*0*1164120203      EOF
102     14:43:23  21 NOV 2

006

14:43:23 21 NOV 2006 : STATUS:

    Termination Statistics: usr 2.18 , sys 0.00 , elapsed
0m2.16

    INPUT : 862 records , 0 blocks , 32355306 record bytes , 0
errors

    OUTPUT: 0 records , 8020 blocks , 0 bytes , 1 errors

14:43:23 21 NOV 2006 : STATUS:

    Program terminated. Exit code is 7
```

However, if waiting for new updates, the jlogdup process on Nodej will not be informed of the socket failure until new updates are added to the journal and the process attempts to transfer them to Nodek, at which point the failure is reported as above.

rename

The rename option is used to change the location of files used within journal updates to other locations. It is typically used when transferring data between machine when the directory structure of the two machine is different. As each update is read, if the rename option is effect it will change the destination location on-the-fly.

The journal may contain a transaction:

Type.......	jBASE..... login	Full file path name................	Update.. Time	Update..... Date
TRANSTART	jdata4.1		13:06:29	23 NOV 2006
WRITE	jdata4.1	C:\jdata4.1\CUSTOMERS	13:06:29	23 NOV 2006
WRITE	jdata4.1	C:\jdata4.1\CUSTOMERS	13:06:29	23 NOV 2006
WRITE	jdata4.1	C:\jdata4.1\CUSTOMERS	13:06:29	23 NOV 2006
WRITE	jdata4.1	C:\jdata4.1\CUSTOMERS	13:06:29	23 NOV 2006
WRITE	jdata4.1	C:\jdata4.1\CUSTOMERS	13:06:29	23 NOV 2006
WRITE	jdata4.1	C:\jdata4.1\CUSTOMERS	13:06:29	23 NOV 2006
WRITE	jdata4.1	C:\jdata4.1\CUSTOMERS	13:06:29	23 NOV 2006
WRITE	jdata4.1	C:\jdata4.1\CUSTOMERS	13:06:29	23 NOV 2006
WRITE	jdata4.1	C:\jdata4.1\CUSTOMERS	13:06:29	23 NOV 2006
WRITE	jdata4.1	C:\jdata4.1\CUSTOMERS	13:06:29	23 NOV 2006
TRANSEND	jdata4.1		13:06:30	23 NOV 2006

If the following command were run on Nodek (in our case), you will see that the destination is changed appropriately.

```
jlogdup -V input set=socket hostname=Nodek port=4089 output set=database rename=c:\jdata4.1\CUSTOMERS,c:\temp\CUSTOM
ERS_COPY
13:11:55 23 NOV 2006 : STATUS:
   Begin jlogdup process: From set 'set=socket hostname=localhost port=4089' to set 'set=database rename=c:\jdata4.1\CUSTOMERS,c:\
temp\CUSTOMERS_COPY'
0*1*3636*1164287189     TRANSTART                                    jdata4.1
168     13:06:29  23 NOV 2006
0*1*3737*1164287189     WRITE    c:\temp\CUSTOMERS_COPY              168*1*0
jdata4.1        168     13:06:29  23 NOV 2006
0*1*49140*1164287189  WRITE    c:\temp\CUSTOMERS_COPY              168*1*1
jdata4.1        168     13:06:29  23 NOV 2006
0*1*94571*1164287189  WRITE    c:\temp\CUSTOMERS_COPY              168*1*2
jdata4.1        168     13:06:29  23 NOV 2006
0*1*139974*1164287189  WRITE    c:\temp\CUSTOMERS_COPY             168*1*3
jdata4.1        168     13:06:29  23 NOV 2006
0*1*185377*1164287189  WRITE    c:\temp\CUSTOMERS_COPY             168*1*4
jdata4.1        168     13:06:29  23 NOV 2006
0*1*230780*1164287189  WRITE    c:\temp\CUSTOMERS_COPY             168*1*5
jdata4.1        168     13:06:29  23 NOV 2006
0*1*276183*1164287189  WRITE    c:\temp\CUSTOMERS_COPY             168*1*6
jdata4.1        168     13:06:29  23 NOV 2006
0*1*321586*1164287189  WRITE    c:\temp\CUSTOMERS_COPY             168*1*7
jdata4.1        168     13:06:29  23 NOV 2006
0*1*366989*1164287189  WRITE    c:\temp\CUSTOMERS_COPY             168*1*8
jdata4.1        168     13:06:29  23 NOV 2006
0*1*412392*1164287189  WRITE    c:\temp\CUSTOMERS_COPY             168*1*9
jdata4.1        168     13:06:29  23 NOV 2006
0*1*457795*1164287190  TRANSEND                                     jdata4.1
168     13:06:30  23 NOV 2006
```

renamefile

This option allows for a number of automatic file translations, an that the file only is specified on the command line to enable the translation to occur.
If a file is defined as C:\move_data and the contents are :

```
C:\jdata\CUSTOMERS,C:\jdata\CUSTOMERS_COPY
```

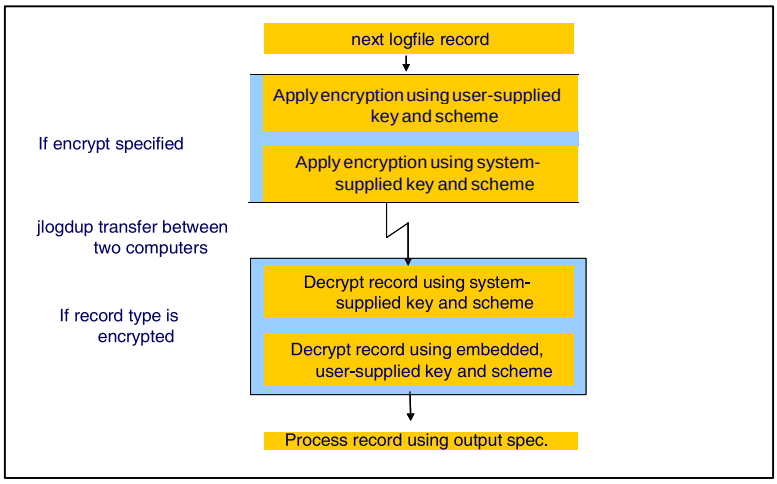
Then using the following command will create the same result as the rename example above.

```
C:\jdata4.1>jlogdup -V input set=socket hostname=localhost port=4089 output set=database renamefile=c:\move_data
```

Note:
The rename file may contain many entries, one per line of the form “from,to” to effect many automatic redirections. Note also that the content of the “from” field must be exactly as it appears in the journal and is case sensitive.

Socket Stream Encryption

Journal transfers via jlogdup are normally on an unencrypted stream, leaving the data unprotected during the session. The operator is able to specify on the jlogdup command line the form of encryption required for the session.
The first thing to note is that encryption is specified on the sending jlogdup process only; embedded information in the stream will identify that this stream of data is encrypted, the encryption scheme used and the key to use to encrypt/decrypt the stream. In order that the key (especially) and the scheme is not sent in clear-text format, the blocks sent between the two jlogdup processes will undergo a further encryption using an internally-specified encryption scheme and key. Note that the encryption options are only allowed on output specifications..



Using the examples above and extending for encryption usage, the following will illustrate the use of this facility.

```
jlogdup input set=current terminate=wait scheme=blowfish
key=Nodejkey output set=socket hostname=Nodek port=4089
```

where, more generally:

scheme

Is 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
blowfish
des
3des
blowfish
rc2_base64
des_base64
3des_base64
blowfish_base64

If key is omitted from the command line then a default internal value will be used.

key

Is the string to be used as the encryption key for the transfer of journal entries. If scheme is omitted on the command line, then a default internal value will be used.

encrypt=true

If either scheme or key are omitted, their values will be internal values. If either key or scheme or both are set then they will override the default internal values.

Notes:

- If the logset is encrypted, then this encryption is in addition to any transient encryption during jlogdup transfers.
- If the logfile is encrypted on the source machine then:
    - If the output set is to "logset" then the resulting destination logset will also contain the encrypted records.
    - If the output set is to database then the encrypted records are decrypted prior to storage on the database.
    - If the output set is anything else then the encrypted records remain encrypted.

Example 3: Report Options

There are various reporting options which may be used with jlogdup, some are displayed in real-time, others record the output to the specified file location.

-e file

This option will produce an error log containing any update errors during the jlogdup session. The file specified must exist as a hash file.

The use of this option may be illustrated by the following examples:

Starting with an empty logset, set to current and active.
Now create a new hash file thus:

```
CREATE-FILE NEW_FILE 1 1

[ 417 ] File NEW_FILE]D created , type = J4

[ 417 ] File NEW_FILE created , type = J4
```

If an attempt to roll the database forward, there will be an error as NEW-FILE already exists. This will be reported to the specified error file.

```
jlogdup -e error_file input set=current output set=database

10:23:07 28 NOV 2006 : STATUS:

    Begin jlogdup process: From set 'set=current' to  set
'set=database'

10:23:07 28 NOV 2006 : ERROR:  For definition set=database

    CREATE-FILE of 'C:\jdata4.1\NEW_FILE]D' failed, error
number 17

    Error code
'JEDI_FILEOP_FILE_EXISTS_DATA]C:\jdata4.1\NEW_FILE]D'

10:23:07 28 NOV 2006 : ERROR:  For definition set=database

    CREATE-FILE of 'C:\jdata4.1\NEW_FILE' failed, error number
17

    Error code
'JEDI_FILEOP_FILE_EXISTS_DATA]C:\jdata4.1\NEW_FILE'

10:23:07 28 NOV 2006 : STATUS:

    Termination Statistics: usr 0.02 , sys 0.00 , elapsed
0m0.01

    INPUT : 2 records , 0 blocks , 284 record bytes , 0 errors
```

```
    OUTPUT: 2 records , 0 blocks , 284 bytes , 2 errors

10:23:07 28 NOV 2006 : STATUS:

    Program terminated. Exit code is 0
```

The error_file may now be edited by jed (say), to display the following (edited out blank lines):

```
jed error_file *


File error_file , Record '0*1*3636*1164709008'

Command->

0001 1

0002 0

0003 3636

0004 143

0005 CREATEFILE

0006 1164709008

0007 0

0008 6

0009 2144

0010 227

0011 CREATE-FILE of 'C:\jdata4.1\NEW_FILE]D' failed, error
number 17.    Error code
'JEDI_FILEOP_FILE_EXISTS_DATA]C:\jdata4.1\NEW_FILE]D'



0021 C:\jdata4.1\NEW_FILE]D
```

```
0022

0023 jdata4.1
```

A new item will be written for each error a new item in the error file.

Note: This option is only valid for "output set=database".

-f future update

This option is used in conjunction with the "–v" or "–V" option to display the next update to the screen. This may be useful if the updates are very long and some activity is seen after starting the command. The normal display for "-v" or "-V" is to display the update after completing the update.

-h help screen

This option is used to display a help screen. It contains an overview of the command and all the reporting options.

```
jlogdup -h

Usage: Called as:

   jlogdup: {-options} INPUT input_spec OUTPUT output_spec

Where {-options} can be one or more of :

  -efilename     Error file for database update errors

  -f             Used with -v or -V, show Future update not
Past update

  -h             Display the concise help screen

  -lfilename     Log file name to write all status and errors
information

  -mnn           Maximum number of errors (default 10000)

  -unn           Write '*' to the screen every nn input
records

  -v             verbose output , 1 line per record

  -x             eXclusive use of the database , no group
locks taken

  -H             Display the verbose help screen
```

```
  -V            Very verbose output , 1 line per record
```

–l assign a log status file

This option is used to assign a file to which all status and error information may be stored. This is not the same as the "-e" option in that this file will record not only the final status of the operation, but also a high-level description of an errors which may have occurred during the session.

If the following command is issued:

```
jlogdup -l logger  -e error_file input set=current output
set=database

11:53:14 28 NOV 2006 : STATUS:

    Begin jlogdup process: From set 'set=current' to  set
'set=database'

11:53:14 28 NOV 2006 : ERROR:  For definition set=database

    CREATE-FILE of 'C:\jdata4.1\new-file]D' failed, error
number 17

    Error code 'JEDI_FILEOP_FILE_EXISTS_DATA]C:\jdata4.1\new-
file]D'

11:53:14 28 NOV 2006 : ERROR:  For definition set=database

    CREATE-FILE of 'C:\jdata4.1\new-file' failed, error number
17

    Error code 'JEDI_FILEOP_FILE_EXISTS_DATA]C:\jdata4.1\new-
file'

11:53:14 28 NOV 2006 : STATUS:

    Termination Statistics: usr 0.03 , sys 0.00 , elapsed
0m0.01

    INPUT : 2 records , 0 blocks , 284 record bytes , 0 errors
```

```
    OUTPUT: 2 records , 0 blocks , 284 bytes , 2 errors

11:53:14 28 NOV 2006 : STATUS:

    Program terminated. Exit code is 0
```

Then “error_file” will contain two records as before.

```
error_file....


0*1*3636*1164714772

0*1*3779*1164714773


 2 Records Listed
```

and the “logger” file will contain the run-time errors and status, thus:

```
11:53:14 28 NOV 2006 : STATUS:

    Begin jlogdup process: From set 'set=current' to  set
'set=database'

11:53:14 28 NOV 2006 : ERROR:  For definition set=database

    CREATE-FILE of 'C:\jdata4.1\new-file]D' failed, error
number 17

    Error code 'JEDI_FILEOP_FILE_EXISTS_DATA]C:\jdata4.1\new-
file]D'

11:53:14 28 NOV 2006 : ERROR:  For definition set=database

    CREATE-FILE of 'C:\jdata4.1\new-file' failed, error number
17
```

```
    Error code 'JEDI_FILEOP_FILE_EXISTS_DATA]C:\jdata4.1\new-
file'

11:53:14 28 NOV 2006 : STATUS:

    Termination Statistics: usr 0.03 , sys 0.00 , elapsed
0m0.01

    INPUT : 2 records , 0 blocks , 284 record bytes , 0 errors

    OUTPUT: 2 records , 0 blocks , 284 bytes , 2 errors

11:53:14 28 NOV 2006 : STATUS:

    Program terminated. Exit code is 0
```

–m nn Set the maximum number of errors

The "-m" option allow the operator to specify the maximum number of errors before aborting the jlogdup process. This is normall set very high (the default is 10,000). In this example if we set the maximum count to 1, the process will abort following the first error.

```
jlogdup -l logger  -e error_file -m1 input set=current output
set=database

12:14:38 28 NOV 2006 : STATUS:

    Begin jlogdup process: From set 'set=current' to  set
'set=database'

12:14:38 28 NOV 2006 : ERROR:  For definition set=database

    CREATE-FILE of 'C:\jdata4.1\new-file]D' failed, error
number 17

    Error code 'JEDI_FILEOP_FILE_EXISTS_DATA]C:\jdata4.1\new-
file]D'

12:14:38 28 NOV 2006 : ERROR:  For definition set=database

    CREATE-FILE of 'C:\jdata4.1\new-file' failed, error number
17
```

```
    Error code 'JEDI_FILEOP_FILE_EXISTS_DATA]C:\jdata4.1\new-
file'

12:14:38 28 NOV 2006 : ERROR:

    Aborting -- exceeded the maximum of 1 errors

12:14:38 28 NOV 2006 : STATUS:

    Termination Statistics: usr 0.03 , sys 0.00 , elapsed
0m0.01

    INPUT : 2 records , 0 blocks , 284 record bytes , 0 errors

    OUTPUT: 2 records , 0 blocks , 284 bytes , 2 errors

12:14:38 28 NOV 2006 : STATUS:

    Program terminated. Exit code is 4
```

The error file contains:

```
error_file....


0*1*3636*1164714772


 1 Records Listed
```

and the “logger” file contains:

```
12:10:38 28 NOV 2006 : STATUS:

    Begin jlogdup process: From set 'set=current' to  set
'set=database'

12:10:38 28 NOV 2006 : ERROR:  For definition set=database
```

```
    CREATE-FILE of 'C:\jdata4.1\new-file]D' failed, error
number 17

    Error code 'JEDI_FILEOP_FILE_EXISTS_DATA]C:\jdata4.1\new-
file]D'

12:10:38 28 NOV 2006 : ERROR:  For definition set=database

    CREATE-FILE of 'C:\jdata4.1\new-file' failed, error number
17

    Error code 'JEDI_FILEOP_FILE_EXISTS_DATA]C:\jdata4.1\new-
file'

12:10:38 28 NOV 2006 : ERROR:

    Aborting -- exceeded the maximum of 1 errors

12:10:38 28 NOV 2006 : STATUS:

    Termination Statistics: usr 0.04 , sys 0.00 , elapsed
0m0.01

    INPUT : 2 records , 0 blocks , 284 record bytes , 0 errors

    OUTPUT: 2 records , 0 blocks , 284 bytes , 2 errors

12:10:38 28 NOV 2006 : STATUS:

    Program terminated. Exit code is 4
```

-u nn Display "*" every "nn" records

Use of this option provides a periodic display of an asterisk when jlogdup is running. The "-u" option allows the operator to set the number of records to appear in the journal the display of the next asterisk. Sample output:

```
jlogdup -u10 input set=current output set=database

12:29:23 28 NOV 2006 : STATUS:

    Begin jlogdup process: From set 'set=current' to  set
'set=database'

*************************
```

```
12:29:23 28 NOV 2006 : STATUS:

    Termination Statistics: usr 0.29 , sys 0.00 , elapsed
0m0.25

    INPUT : 265 records , 0 blocks , 9930396 record bytes , 0
errors

    OUTPUT: 265 records , 0 blocks , 9930396 bytes , 0 errors


12:29:23 28 NOV 2006 : STATUS:

    Program terminated. Exit code is 0
```

Verbose options

Two options exist which allow the operator to view the records being worked on by jlogdup

-v verbose

This option shows the journal update details ("*" separated field showing whever, precisely in the journal the record exists; the type of journal entry, the file being updated and finally the record being updated

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

12:32:12 28 NOV 2006 : STATUS:

    Begin jlogdup process: From set 'set=current' to  set
'set=database'

0*1*3636*1164716955   TRANSTART
```

```
0*1*3737*1164716955   WRITE   C:\jdata4.1\CUSTOMERS
227*1*0

0*1*49140*1164716955  WRITE   C:\jdata4.1\CUSTOMERS
227*1*1

0*1*94571*1164716955  WRITE   C:\jdata4.1\CUSTOMERS
227*1*2

0*1*139974*1164716955  WRITE   C:\jdata4.1\CUSTOMERS
227*1*3

0*1*185377*1164716955  WRITE   C:\jdata4.1\CUSTOMERS
227*1*4

0*1*230780*1164716955  WRITE   C:\jdata4.1\CUSTOMERS
227*1*5

0*1*276183*1164716955  WRITE   C:\jdata4.1\CUSTOMERS
227*1*6

0*1*321586*1164716955  WRITE   C:\jdata4.1\CUSTOMERS
227*1*7

etc
```

-V very verbose

In addition, the very verbose option also shows the user name; the port number, the time and the date of the update:

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

12:34:12 28 NOV 2006 : STATUS:

    Begin jlogdup process: From set 'set=current' to  set
'set=database'

0*1*3636*1164716955   TRANSTART
jdata4.1         227     12:29:15  28 NOV 2006

0*1*3737*1164716955   WRITE   C:\jdata4.1\CUSTOMERS
227*1*0        jdata4.1         227     12:29:15  28 NOV 2006
```

```
0*1*49140*1164716955  WRITE   C:\jdata4.1\CUSTOMERS
227*1*1        jdata4.1         227     12:29:15  28 NOV 2006

0*1*94571*1164716955  WRITE   C:\jdata4.1\CUSTOMERS
227*1*2        jdata4.1         227     12:29:15  28 NOV 2006

0*1*139974*1164716955  WRITE   C:\jdata4.1\CUSTOMERS
227*1*3        jdata4.1         227     12:29:15  28 NOV 2006

Etc
```

-x exclusive use of database

The speed of database recovery may be improved by the "-x" option. This option **must** be used with care. No group locks will be taken when the output set is to database. This is for recovery only, when there should be no processes updating the database.

-H verbose help screen

This option will display all options for input/output specs.; timespec details plus the output of the "-v" option.

```
jlogdup -H
Usage: Called as:
  jlogdup: {-options} INPUT input_spec OUTPUT output_spec
Where {-options} can be one or more of :
 -efilename     Error file for database update errors
 -f             Used with -v or -V, show Future update not Past update
 -h             Display the concise help screen
 -lfilename     Log file name to write all status and errors information
 -mnn           Maximum number of errors (default 10000)
 -unn           Write '*' to the screen every nn input records
 -v             verbose output , 1 line per record
 -x             eXclusive use of the database , no group locks taken
 -H             Display the verbose help screen
 -V             Very verbose output , 1 line per record
Where 'input_spec' and 'output_spec' can be one or more of the following:
 blockmax=nnn (S)          The maximum size, in blocks, of a serial device
 blocksize=nnn             The block size to read or write data to/from a TTY or SERIAL device or file
 device=file|device (S) Provide the name of a file or device for a SERIAL device. Can be more than one of these
 encrypt=true(O)           Output transfer is to be encrypted
 end=timespec (I)          Specify a time specification (see later) to end the input data at
 hostname=host(IOK)        Host for socket transfers to / from
 key=encryptkey            The key to use for encryption
 noflush{=true|false} (O) End of transaction causes the output buffer to be flushed immediately
 notrans{=true|false} (O) All transaction boundaries will be ignore if true (default false)
 port=portnum (IOK)        Socket port to use for socket transfer
 prompt{=true|false}       When switching serial devices or files, always prompt user first
 rename=fromdir,todir (O) Path names will be converted from 'fromdir' to 'todir'
 renamefile=filename (O) Specifies a file with a list of renames in the format 'fromdir,todir'
 retry=nn (I)              When 'terminate=wait' used, the time interval between retry attempts
 scheme=method             Encryption method for socket transfers
 set=current (I) (L)       Begin using the current log set as input
 set=database (O) (D)      The output should update the database i.e. a restore process
 set=eldest (I) (L)        Begin using the log set that has the eldest entries in it
 set=n (I) (L) (N)         Begin using log set number n
 set=null (O)              Output is to NULL i.e. discarded
 set=serial (S)            The input/output is to a serial device or file - requires one or more 'device=file|device'
 set=socket (IOK)          Input/output is to a socket
 set=stdin (I) (T)         The input data comes from the terminal stdin
 set=stdout (O) (T)        The output data goes to the terminal stdout set=tty
 (T)                       The input is from stdin or the output data is to stdout
 set=logset (O) (L)        The output will be placed on the current log set as though a native update
 start=timespec (I)        Specify a time specification (see later) to start the input data at
 terminate=eof (I)         Switch to elder log sets as necessary and only terminate on very eldest update
 terminate=eos (I)         Terminate program when the log set first processed is exhausted
 terminate=wait (I)        Switch to elder log sets as necessary and wait for new updates when exhausted
 terminate=waiteos (I)     Switch to elder log sets as required and wait for new updates until logset switched, then terminate
 timeout=nnn (I)           Provides a timeout for 'terminate=wait' in seconds
 verbose{=true|false}      Display to stderr a summary of the specification
 (D) shows we assume the device type is DATABASE
 (I) shows it is only valid for an INPUT device
 (L) shows we assume the device type is LOG SET
 (N) shows we assume the device type is NULL
 (O) shows it is only valid for an INPUT device
 (S) shows we assume the device type is SERIAL
 (T) shows we assume the device type is TTY
timespec (used in start= and end=) can be one of the following :
 nn:nn:nn                  Time of day (todays date assumed)
 DD-MMM-YYYY               Date (midnight assumed). Any date convention accepted
 nn:nn:nn,DD-MM-YYYY       Both time and date specified (or the other way around)
 filename                  Name of file create with 'jbackup -sfilename' -- the time jbackup started is used
 filename                  Any other normal file , the time the file last modified is used
```

RESILIENT FILES

Resilient files have the following characteristics: they are resistant to corruption in adverse conditions and they have the ability to auto-resize themselves as the population of such files increase.

Resilience

For standard jBASE hashed files, the writing of an item may cause one or many physical disk writes, depending on the size of the item being written. If the series of writes is interrupted (by say, a power failure), then the structure of the file may be compromised as the item may be partially written to disk.

The resilience (for Resilient files) is provided by running in SECURE mode where any update resolves down to a single disk write, any dependent writes having been flushed to disk beforehand. Fundamentally, the body of the item is written to and then flushed to disk. If a power failure occurs at this time, the "before image" of the item is still in existence on disk with the integrity of the file being maintained. The intended update is abandoned (because of the power failure). Upon power being restored to the system, the database may not be in a consistent state if the failed update was part of a transaction. This does not present a problem as the entire transaction will have been written to the Transaction Journal prior to attempting any database disk writes of the transactional data. The transaction will thus be replayed in its entirety, thus maintaining database consistency, (via a roll-forward – this will be described later in the document)
In the normal course of events the final write/ of the item pointer on disk will not be interrupted, the pointer will be switched to the new version of the item thus completing the item write.

Autosizing

With the increase in 24 hour operation there has been a corresponding decrease of available time for system maintenance of hashed files. Standard hashed files become less efficient as the data population exceeds the original creation sizing, resulting in slower retrieval and updates, so an expanding hashed file requires regular resizing.

Resilient files need no resizing as there is no concept of overflow. When the data within a frame exceeds the available disk space it is split into a pointer frame pointing to child data frames. The individual items within the frame are rehashed according to the split level and reallocated to the appropriate child frame. The hashing algorithm base changes according to the split level to avoid common hashing paths.

Where standard hashed files have a linear expansion of search path (the number of data frames read according to population), resilient files have a logarithmic expansion of the order *Modulo*, so where an undersized hashed file may require 5

disk reads a resilient file may require 3. A *properly* sized hashed file may require only one disk read, but that is assuming regular system maintenance.

The logarithmic search path may imply an exponential file size expansion, but this doesn't happen in practice as data frames which are not required, are not allocated.

SYNTAX

CREATE-FILE TYPE=JR [Modulo] [INTMODS=x[,y[,z]]] [SECURE=YES] [MINSPLIT=m] [HASHMETHOD=h] [SECSIZE=n]

SYNTAX ELEMENTS

Parameter	**Description**
Modulo	A comma separated list of the modulo of split frames, default 31. When a data frame overfills it will change to a pointer frame of the order *Modulo[level]* with a maximum of *Modulo[level]* child frames where items are rehashed according to the split level and hashing algorithm. There are a maximum of 32 modulo and each must be prime between 3 and 509.
HASHMETHOD	The internal hash method used in internal and external hashing, default 5 (FNV-1a variant, recommended).
INTMODS	Up to 3 prime numbers defining the internal hash table modulo, default 3, 7, 19. The cumulative product cannot exceed 485, i.e x + x * y + x * y * z.
MINSPLIT	Minimum split level of the file. The file will be preallocated to a minimum level of split frames from the Modulo list. This can have extreme adverse affects on performance and excessive file size, so its use is not recommended.
SECSIZE	Secondary record size, default 2048. Items exceeding this size are stored out of group, i.e. the item retains its own data frame(s), referenced by a pointer.
SECURE	The file is flushed at critical junctures such that any file update will rely only on a single disk write. This maintains the file structure in the event of system failure.

Modulo

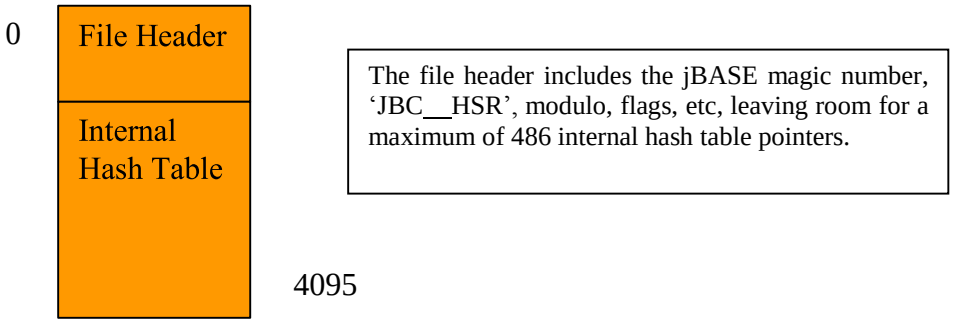
Up to 32 comma-separated prime numbers specifying the external modulo for this file. Only one modulo is usually provided, default 31.

HASHMETHOD

The hash method as used with all hashed files. The default method of 5 is recommended.

INTMODS

A newly created resilient file consists of a single 4096 byte header containing, amongst other things, an internal hash table up to three levels deep.



The size and depth of the internal hash table is specified by the INTMODS parameter and by default take the values 3, 7 & 19. The INTMODS values must be prime and ascending, and the table must fit in the available space in the file header.

MINSPLIT

The MINSPLIT value forces a table to be created with a minimum split level & would normally be used only where the future data population is known to be large and will remain large throughout the lifetime of the file. In general resilient files control their own sizing and MINSPLIT is not required.

MINSPLIT can create extremely large files as it is an exponential sizing parameter. Assuming default parameters (3, 7, 19 & 31) this table show the resultant filesize:

MINSPLIT value	Empty file size
0	4096
1	1,638,400
2	50,667,520
3	1,570,570,240
4	48,687,554,560

If the current data profile is not known or the future profile not predictable then the use of MINSPLIT is not recommended.

SECSIZE

As with all hashed files, if an item size exceeds SECSIZE then the record data is given its own linked chain of data frames and only the record key and a pointer to the data are stored inline. This is known as out of group (OOG) storage. Storing data OOG saves resources when searching or updating a group.

SECURE

When SECURE=YES is specified updates are flushed to disk where necessary to maintain the structure of the file in the event of a system failure. This will affect file performance.

Internal Hash Table Limits

Assuming three modulo x, y & z then the total size of the internal hashed table would be:

x + x * y + x * y * z

By default

3 + 3 * 7 + 3 * 7 * 19, or 423 *This is the number that cannot exceed 485, above.*

And the number of level 0 external frames would be

3 * 7 *19, or 399.

Hashing

Hashing is simply a method of deriving a seemingly random number from the record key and applying a modulo to the result. A given key will always produce the same hash value for a given hash method.

A good hash method will:

1. Produce very different hash values for similar keys
2. Produce a wide range of hash values
3. Produce a flat distribution of hash values

To hash into a modulo 3 table for key FRED where the hash value is 11, the remainder when divided by 3 is 2 so the key FRED hashes to the last group (0-2). In reality the hash value is a very large number.

File Size

A newly-created, empty file will only contain 4096 bytes. A populated and subsequently empty file may contain much more as the following data frames are not released until a resize:

1) Overflow frames.
2) External level 0 frames.
3) Internal data frames.

This is the same process as is followed with any hashed file.

The maximum file size is determined by the addressing limits of 64 bits.

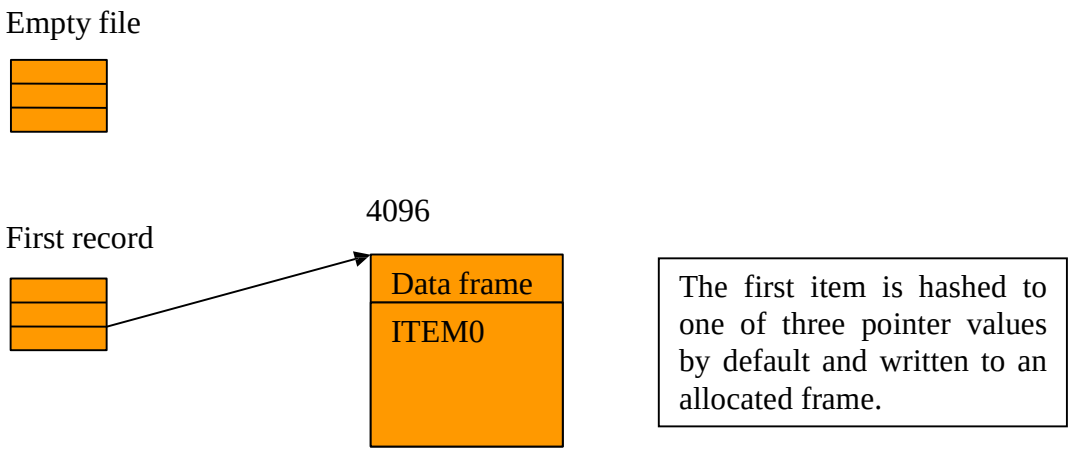
Writing Data

When the first item is added to the file it is hashed on the first internal modulo (default 3) a data frame is added to the file to contain the new item making a minimum file size of 8192.

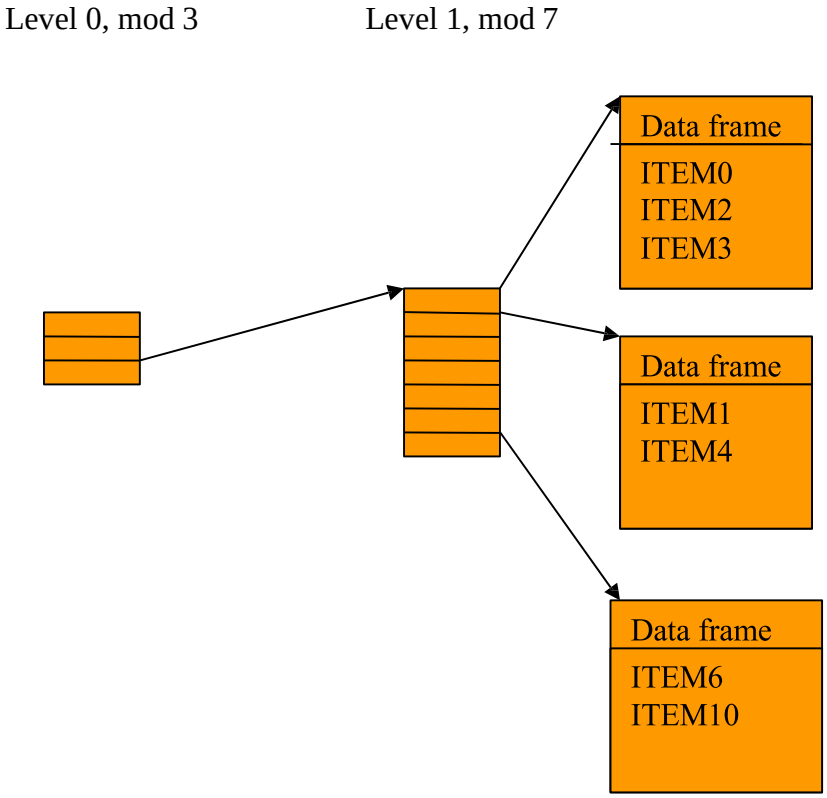
The internal hash table consists of up to three ascending prime numbers, default 3, 7 & 19, that configure the initial search path for the record id. The key is hashed on the first modulo (3) which contains one of:

0	The group is empty, nothing has ever been written to it.
< 4096	An internal pointer to the next modulo (7)
>= 4096	A pointer to a data frame.

In the case of an empty file the value will be 0, so for a write a data frame is allocated and the pointer changed to reference the frame – on the first record this will always reference 4096 as this was end of file at file creation.



When an internally referenced data frame overflows, all items within it are rehashed on the next modulo and reallocated to their respective newly allocated data frames. The original data frame is released to the free list.



Deleting Data

When data is deleted the data frame may become empty in which case it may be released to the free list and its parent pointer zeroed. A reference count in the parent frame is decremented which may in turn cause the pointer frame to be empty, to be released to the free list.

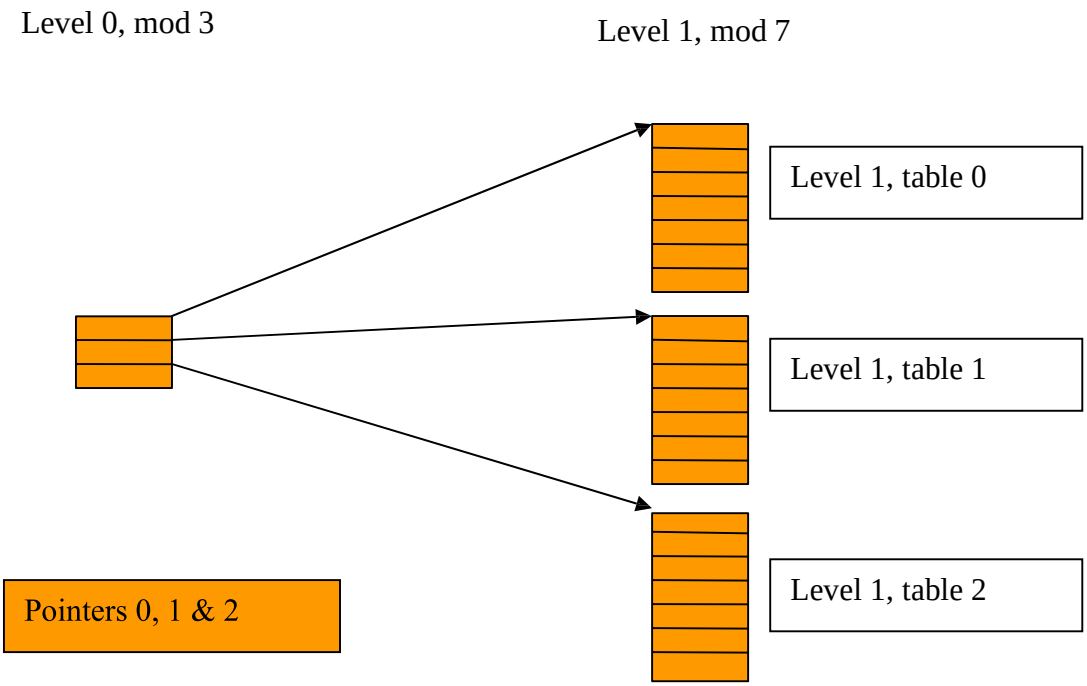
A frame is *not* released, even if empty, if:

1. It is pointed to by the file header (i.e. external level zero or an internal data frame).
2. It is less than the MINSPLIT value for the file.

As records are deleted, a pointer frame may point to a few frames the data within which would fit in the pointer frame if it was changed to a data frame. No check is made for this eventuality as checking all the parent pointer's children is too expensive. Instead a pointer frame is only released when the last pointer is zeroed.

Internal Pointers

Given a set of internal modulo the internal pointer values are known at file creation, i.e. a given pointer can have only one internal value, rather than zero or a data frame reference. In the diagram below the level 0 pointers can only point to their *respective* level 1 tables, hence the internal pointer values are predictable.



Similarly all 21 level 1 pointers each have a respective level 2, modulo 19 table.

Once an internal pointer has been set to an internal table position the pointer will never lose this value throughout the lifetime of the file. When the top level internal pointer (default modulo 19) is allocated a data frame, this will never change throughout the lifetime of the file. If the data frame overflows it becomes a pointer frame, but this doesn't require a change in the internal hash table.

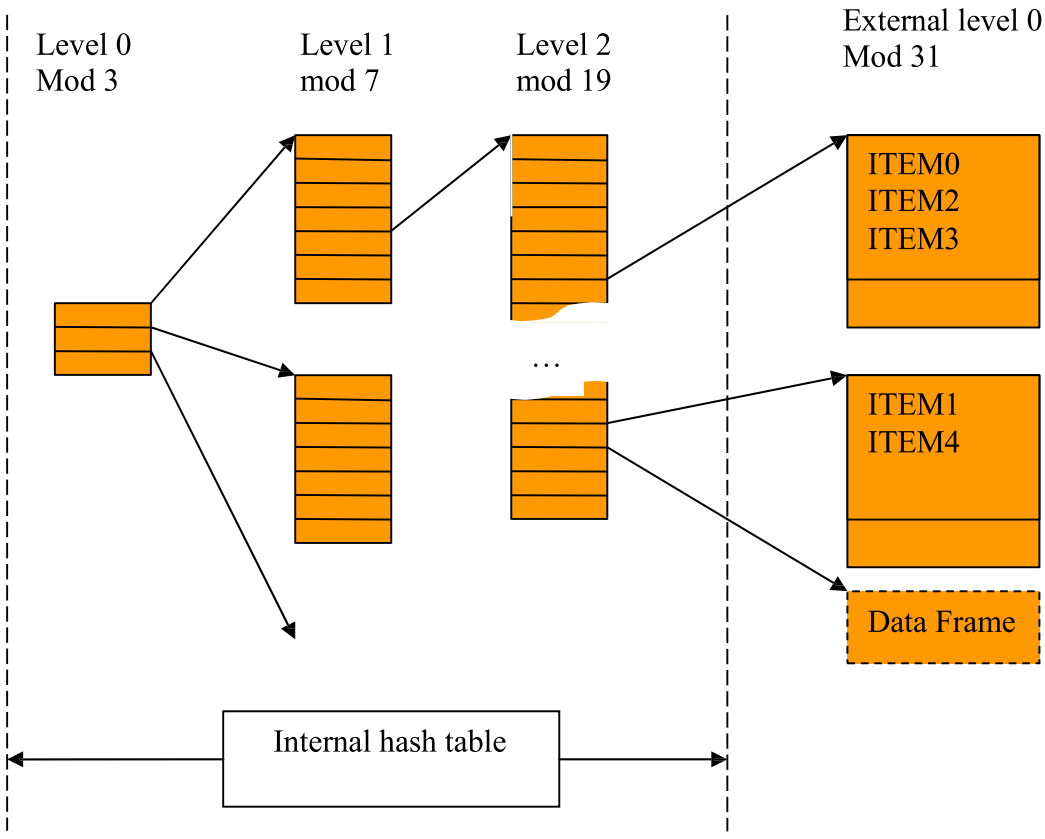
When the internal hash tables are full, no values will ever change, so processes opening the file do not require a re-read of the file header. This avoids expensive locking. Sparsely populated files may require much more locking as pointers to data frames are liable to become internal pointers.

External Frames

Data or pointer frames referenced at the highest internal pointer level of internal hash table or beyond are referred to as external frames. Internal data frames (they cannot be pointer frames) are relatively few, by default a maximum of 24 can ever exist. Once a frame is allocated to the highest level internal reference, it will never be released, even on a CLEAR-FILE.

When an external data frame overflows it is rehashed on the modulo appropriate to the external level, in the same way as internal modulo but the data frame itself becomes a pointer frame.

In internal or external hashing, if no item hashes to a pointer then no data frame is allocated.



External Hashing

If an external data frame overflows, it will split the records according to the modulo and hashing method for the split level. The hash method needs to change as if only one external modulo is used then all items that hashed to group n at level m would also hash to group n at level m + 1 causing a further split, etc.

Given the set of parameters at file creation it is always possible to predict the path a given item id will take, what cannot be predicted is the level within the path (internal and external) at which the item will exist.

jrscan

As the internal structure of Resilient files differs from hashed files so much a new utility, *jrscan* has been written to complement the functionality that jcheck provides to other hashed files, although without the destructive recovery.

The syntax, from jrscan –h, is:

```
Usage: jrscan {options} filepath

Options:
        -a      Show header values.
        -b      Bitmap scan - verify frame use.
        -h      Display help.
        -i      Display the internal hash table.
        -k      Display record keys
        -ln     Set split level to n.
        -on      Offset n (0xnnnn hex, 0nnnn oct or  nnnn
dec).
        -v      Verbose output.
```

-a	Provides a comprehensive breakdown of the header values in the file header.
-b	Uses a bitmap to map the file structure ensuring all frames are referenced once and once only.
-h	Help text.
-i	Display the internal hash table. Used for support purposes.
-k	Display record keys. Used for support purposes.
-ln	Set split level to n. Used for support purposes.
-on	Set internal file offset to n. Used for support purposes.
-v	Verbose output. Display offsets, keys and block sizes.

RECOVERY

Database recovery can take several forms depending on the nature of the state of the system.

For a simple power failure or an O/S reboot while the database is being updated, the database should be recoverable by a system warmstart procedure. This warmstart will use Transaction Journal(s) which are being used by the database(s) to roll forward all complete database transactions from the last checkpoint to the point of failure.

For a media failure whereby the database itself has been lost then this data must be restored from the last backup taken. If the Online Backup facility has been used, then the restore process can restore the system to a consistent state. Providing the Transaction Journal has not been lost during this media failure (journal is held on other media), then is should be possible to recover the system to a position just prior to the media failure. Again the system will be recovered to a consistent state.

For disaster recovery situations where there is likely to be some lengthy/permanent disruption to the live site, it may be possible to continue operations at a site which has been functioning as a hot-standby site.

Warmstart recovery

DB-WARMSTART

This command is restricted to administrative use only and there are no optional parameters.

This command will inspect the "databases-defined" file and determine whether each of the databases defined therein require to be recovered following a power failure. Any defined database which has a status of "active" will cause a recovery process to begin – all databases which have been stopped will be in a consistent state. The recovery process takes the form of a roll-forward of the database from the Transaction Journal logfiles defined for that database. The format of the recovery command is :

jlogdup –V input set=eldest start=CHECKPOINT output set=database

As this command suggests checkpointing must be configured for this to be effective. A checkpoint is defined as a point in time when all transactions have completed in

their entirety – no partial transactional updates are pending. When checkpointing is used, the database is deemed to be in a "consistent" state at the point at which the "checkpoint" record appears in the Transaction Journal. This being the case recovery is only required from the last checkpoint time. No user intervention is required in determining this time. At the completion of the recovery all transactions which were completed in their entirety will be applied to the database and transactions which are incomplete (i.e. no TRANSEND or TRANSABORT entry found in the log files for this transaction), will be discarded.

Syntax
DB-WARMSTART

For all computer types, the "WARMSTART" utility should be run with the JBASE_DATABASE set to "warmstart". It is not possible to predict which database is active (all may not be including "default". As the access to jBASE databases is determined very early on in the life of a process, the "databases_defined" file cannot be interrogated to find a usable database. The database "warmstart" also must be started. This will not be added to the "databases_defined" and as such is a special case. This ensures that recovery is not attempted for this dummy database. Once the recovery of all required databases has completed, the dummy database entry is deleted. Note: As DB-START is only possible by a system administrator, misuse of the dummy database is prevented.

Media/Computer Failure and Recovery

This recovery mechanism relies on three components within jBASE: transaction boundaries; Transaction Journaling and jBackup. Firstly, database integrity cannot be guaranteed unless transaction boundaries are utilised. By encapsulating related database updates within transaction boundaries, jBASE will either perform all of the related updates or none. Transaction boundaries are identified by the TRANSTART, TRANSEND and TRANSABORT instructions. Transaction Journaling is required in order to provide a chronologically-

Saving and Restoring the System

Regular, usually daily, backups are essential to the good housekeeping of any system. There are two mechanisms for performing backups.

You can use existing UNIX commands, such as 'tar' or 'cpio' (or Windows Backup), which work well, but should not be run while a jBASE application is updating files. 'Tar' and 'cpio' and Backup perform a binary dump of the file data, and do not obey any locks, which may have been set to indicate that an update is in progress. In addition, these saves can be limited because they cannot be restored correctly on a system, which has a different architecture to the original system. The preferred mechanism is to use the jbackup and jrestore jBASE utilities. The jbackup program will back up normal UNIX/Windows data files and directories as well as jBASE data files, and

will respect any locks set by jBASE applications. Bear in mind though that if you choose to run jbackup concurrently with other active online jBASE applications, your saved files will not be corrupt, but the continuity of any data saved from an active system cannot be guaranteed. (We shall see later that an Online Backup facility is available which overcomes this caveat.)

jbackup - jBASE Backup Utility

jbackup provides fast on-line backup facilities, which can be used to check file integrity.
jbackup -Option {Inputlist}

Where inputlist is a file containing a list of files, default stdin

Option	Explanation
-bn	Set number of write buffers to n
-c	Dump control files such as indexes as binary files
-f Device	Save to device file, default stdout
-l	Link files to be saved as separate UNIX or hash files
-mn	Maximum data capacity of media in Mb, default 100 Mb
-pn	Set priority, nice value of parent process
-s	Save summary of statistics to UNIX/NT file
-v	Verbose mode
-L file	Save from List file
-B	Force blocksize to 128k. Default 16k
-Cn	Force blocksize to n bytes, rounded to nearest k
-F	Use fixed block device. Use for qic tapes (nt only)
-N	Suppress compression if supported by device (NT only)
-S Statfile	Save statistics of all saved objects in jBASE, file Statfile. The dictionary for this file is JBCRELEASEDIR/jbackup] D.
-O	Override no backup file option, save all
-R	Suppress automatic rewind at end of backup
-P	Print and scan files only, no save
-V	Verbose dot mode, displays a “.” For each file
-A Acc	Save from user name home directory (UNIX only)
-W	indicates an online backup is to be performed

EXAMPLES

```
find /home -print | jbackup -P
```

Reads all records, files and directories under the /home directory provided by the find selection and displays each file or directory name as it is encountered. This option can be used to verify the integrity of the selected files and directories.

```
jbackup FILELIST -f /dev/rmt/floppy -m1 -v
```

Reads all files and directories listed in the UNIX file FILELIST and writes the formatted data blocks to the floppy disk device, displaying each file or directory name as it is encountered. The jbackup utility will prompt for the next disk if the amount of data produced exceeds the specified media size of one Mbyte.

```
jbackup -AjJBASE -S/usr/jbc/tmp/jJBASE_stats >/dev/null
LIST /usr/jbc/tmp/jJBASE_stats USING /usr/jbc/jbackup NAME TOTAL SIZE ID-SUPP
```

Reads all files and directories in home directory of user-id "jBASE" Generates statistics information and outputs blocks to stdout, which is redirected to /dev/null. The statistics information is then listed using the jbackup dictionary definitions to calculate the file space used.

```
jfind C:\users\myhome -print | jbackup -P
```

Reads all records, files and directories under the C:\users\home directory provided by the find selection and displays each file or directory name as it is encountered. This option can be used to verify the integrity of the selected files and directories. This command should be run with jshell type sh rather than jsh.

jrestore - jBASE Restore Utility

The jBASE jrestore utility uses two processes together with shared memory to provide an efficient mechanism for restoring records, files and directories saved by the jbackup utility. One process is used to read data blocks from the restore media and the other creates and writes records, files and directories. This two process approach keeps data movement to a minimum while making good use of any multiprocessing capabilities provided by your system.
jrestore provides a powerful selective restore capability. Records, files and directories can be selectively restored by specifying relational expressions with one or more of the available options.
jrestore is capable of resynchronisation so that the restore procedure can begin from any position on the restore media. However, note that this capability can be limited by a lack of positioning options available with the specific restore device. For example, a streaming cartridge tape cannot be backspaced.
jrestore will continue to restore from the specified device until the end of volume label is detected. You will then be prompted to mount the next device or you can select an alternative device if required.

jrestore command

```
jrestore <options>
```

options are:

Option	Explanation
-a	Restore from current media position.
-b*n*	Set number of input buffers to *n*, default is 8.
-fdev	Restore from *dev* file, default is stdin.
-c"*o n*"	Restore old directory path (*o*) as new directory path (*n*).
-d"*dir*"	Restore directories matching regular expression.
-h"*file*"	Restore hash files matching regular expression.
-i"*key*"	Restore record keys matching regular expression. Usually used with the 'h' option.
-l"*lnk*"	Restore UNIX link files matching regular expression.
-o"*o*"	Restore other UNIX files matching regular expression, e.g. named pipes.
-u"*u*"	Restore regular UNIX files matching regular expression.

Option	Explanation
-p*n*	Set priority/nice value of parent process, default is 1.
-q	Causes the P option to execute in quiet mode. Only a summary is displayed.
-v	Verbose Mode. Display files and directories before they are restored. Output is directed to stderr.
-Ttype	Restore hash files as specified type.
-B	Force buffer block size to 128K, defaults to 16K.
-C*n*	Force block size to n bytes, rounded to nearest 1024 bytes.
-U	Update only, existing files or records are not overwritten.
-O	Overwrite existing files and records.
-P	Print and scan files only, no restore.
-V	Verbose dot mode. Display a '.' for each file.
-W	Roll forward the database following the restore using the saved logfile data and configuration
-G	Roll forward the database using the logfile data and configuration which are already in use. This will follow the data restore and roll forward specified by the -W option.

jrestore Examples

```
jrestore -f /dev/rmt/ctape -P
```

Reads formatted files and directories from a streaming cartridge device, displaying each file or directory as it is encountered. This option can be used to verify that the tape does not contain any parity or formatting errors and so can be restored at a later date.

```
jrestore -f /dev/rmt/floppy -v
```

Reads and restores formatted files and directories from a floppy disk device, displaying each file or directory as it is encountered.

```
jbackup -Ajbase | jrestore -c"/home/old /home/new"
```

Reads formatted files and directories from stdin, which is being supplied by jbackup, modifies all occurrences of path string /home/old to /home/new and then restores files and directories using modified path string.

```
jrestore -f BACKUP -d".*PAYROLL$"
```

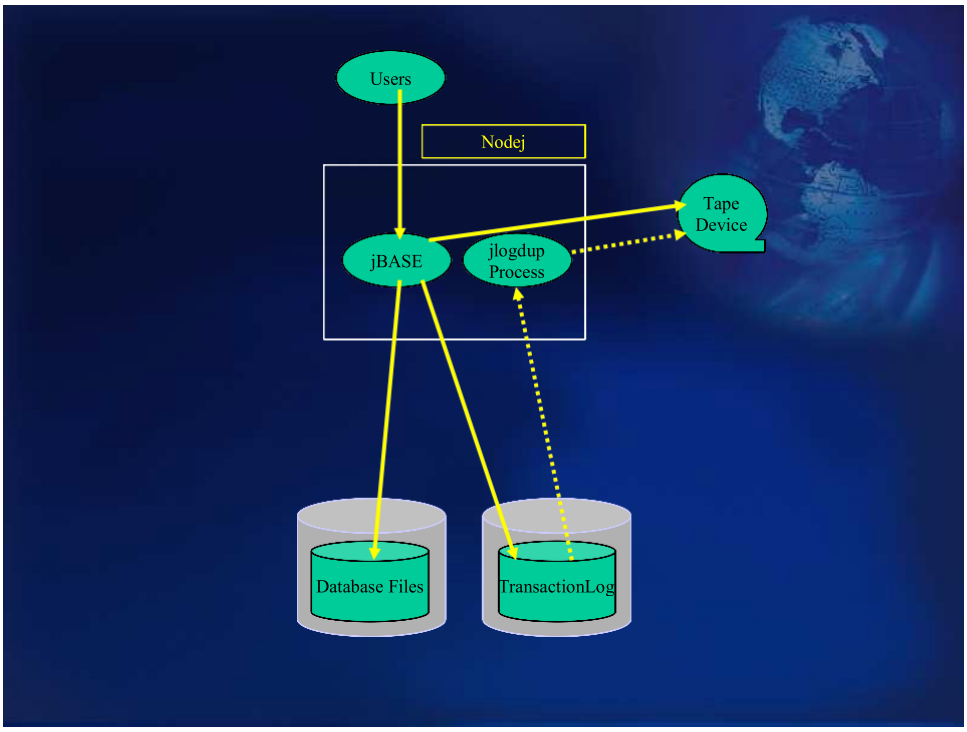
Reads formatted files and directories from UNIX file BACKUP, limits restore to any directories whose path name ends in PAYROLL.

```
jrestore -f BACKUP -h"/CUSTOMERS$" -i".*SMITH.*"
```

Reads formatted files and directories from UNIX file BACKUP, limits restore to any hash files whose path name ends in CUSTOMERS, and only restores record keys containing

Sample System Configurations

Transaction Journaling on a single system – offline backups

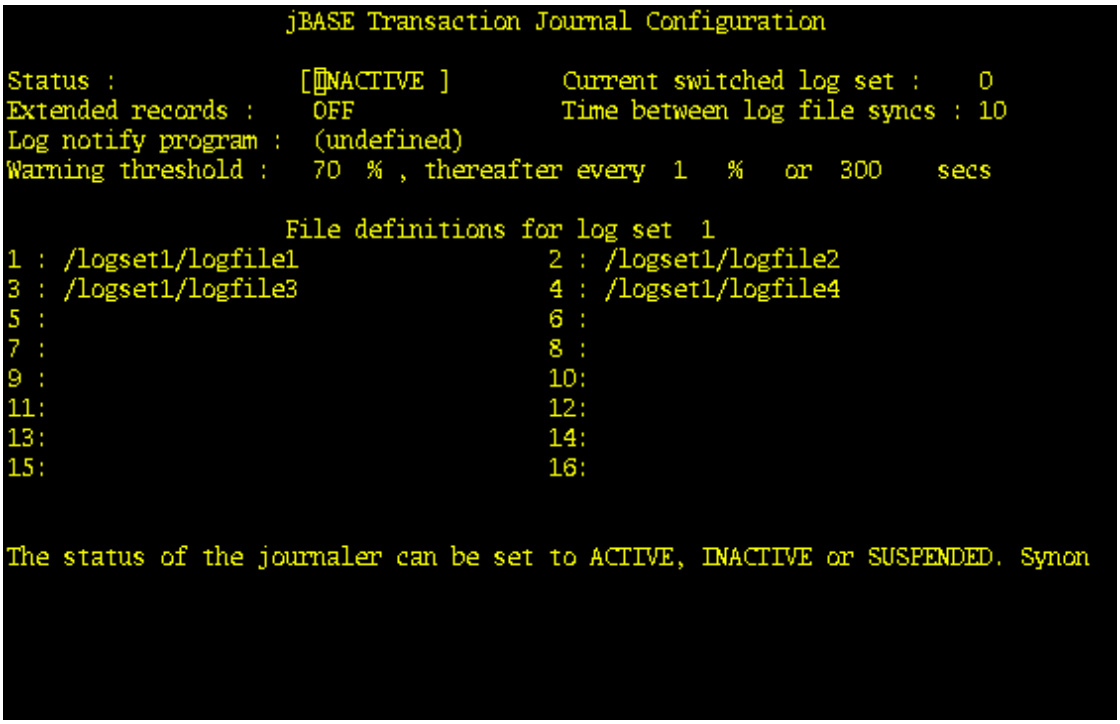


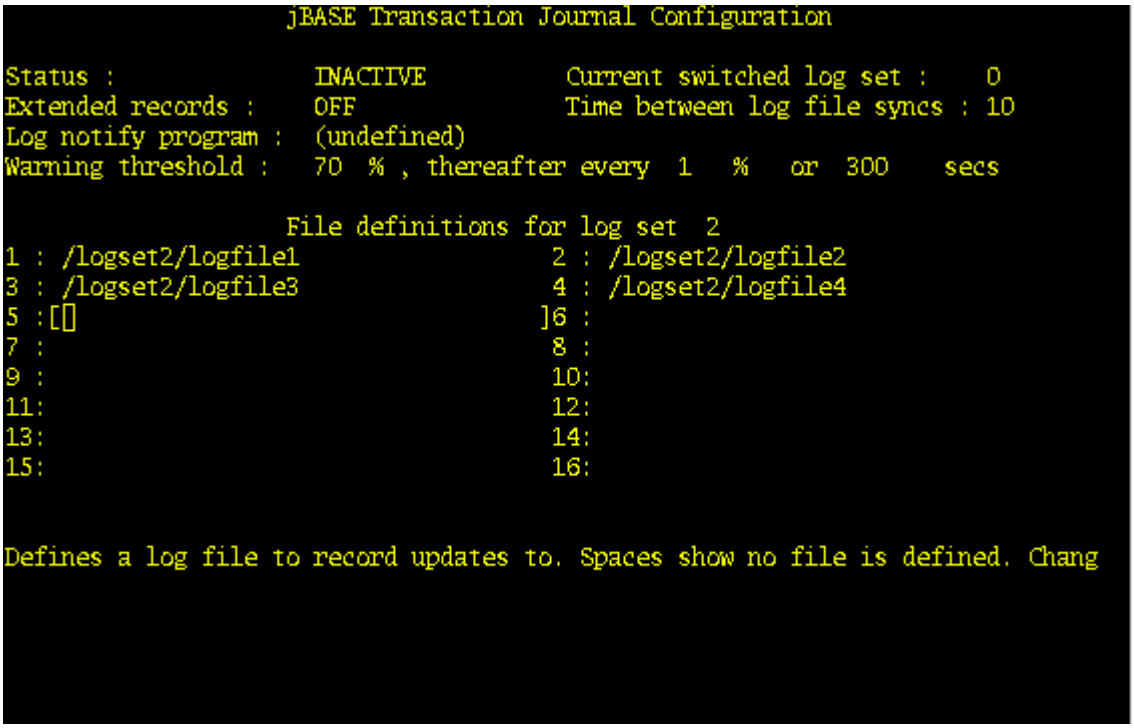
The diagram above represents the use of Transaction Journaling on a stand-alone system. In the event of system failure, the vast majority of processing which has taken place since the last system backup can be recovered and replayed. This is vital for those situations where the transaction cannot physically be repeated.

Journal Configuration

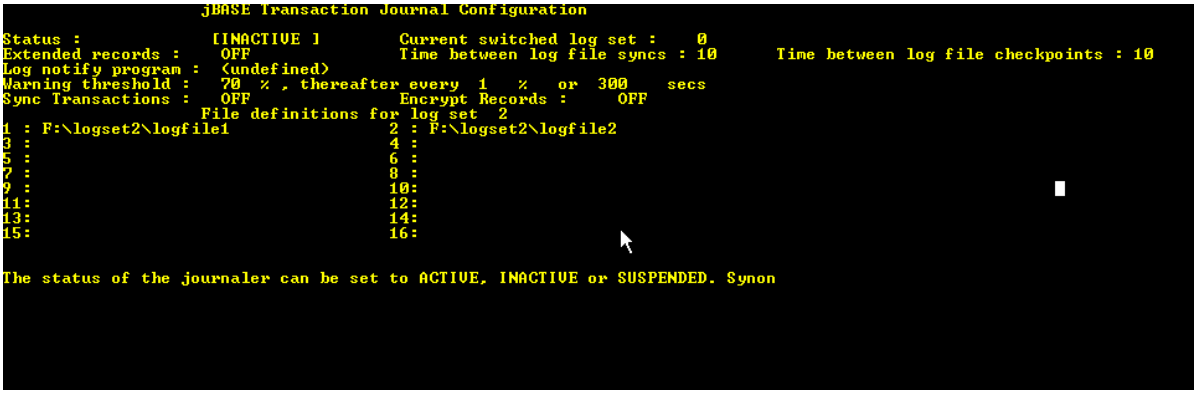
The Transaction Journal will be configured with two logsets; logset1 and logset2. Each of these logsets will occupy a separate volume partition on disk; this will allow for correct size monitoring of the logsets. The statistics of the logset usage indicated by the jlogstatus command is not at obvious 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 reflect the TJ logset usage (albeit with a 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:





For Windows :



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 (Windows: *Task Scheduler*) 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:
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.

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.

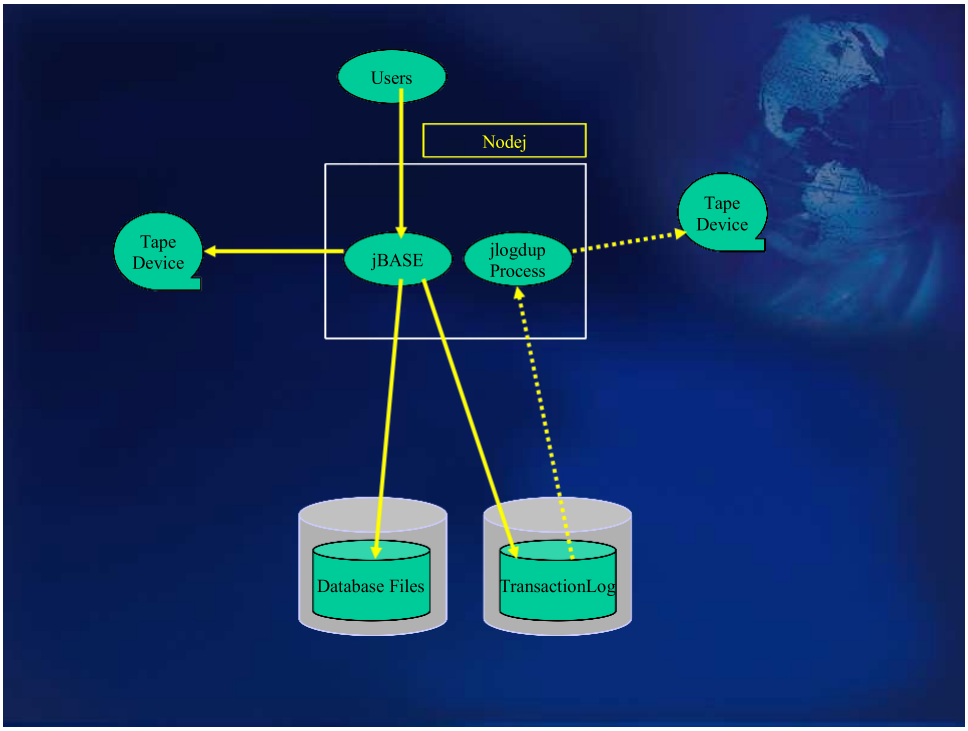
Tape device/tape failure.

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.

Failure 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 with two tape desks

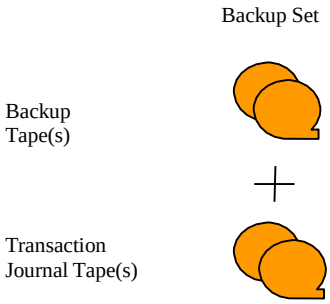


The schematic shows the same system, except this time equipped with two tape decks. The advantages of this configuration over the previous are as follows:
For the majority of the time during the day, there is a tape deck free for other uses; either for system or development usage.
This configuration 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.

Introduction of Online Backup into the Operation

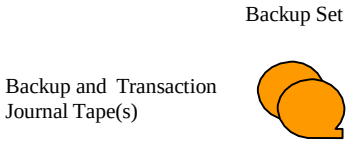
The previous scenario showed that by duplicating the number of tape decks allowed for some redundancy, plus, more importantly operational benefits. The operational benefits are very significant – the administrator may now leave users on the system whilst performing a backup. This may be achieved by carefully scripting the procedure. A “backup set” required for database recovery is therefore



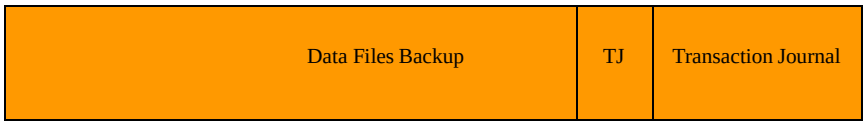
The main drawbacks with this approach is that the backup tapes and the Transaction Journal tapes have no connection between them except for any labeling which the administrator may do. In times of a system failure, a script can be written which manages the actual recovery of the machine, but this is prone to error if the management on the media is not carefully controlled.

Online Backup and Recovery solves this problem.
The Online Backup facility has been developed to enable system managers to perform necessary regular database backups while still allowing users the ability to perform updates on the database, without having to be concerned about whether the correct set of Transaction Journal tapes are restored following the database restore. The process also benefits in that it has become an automatic process, not reliant upon a script for it to function correctly.

The backup set now becomes :



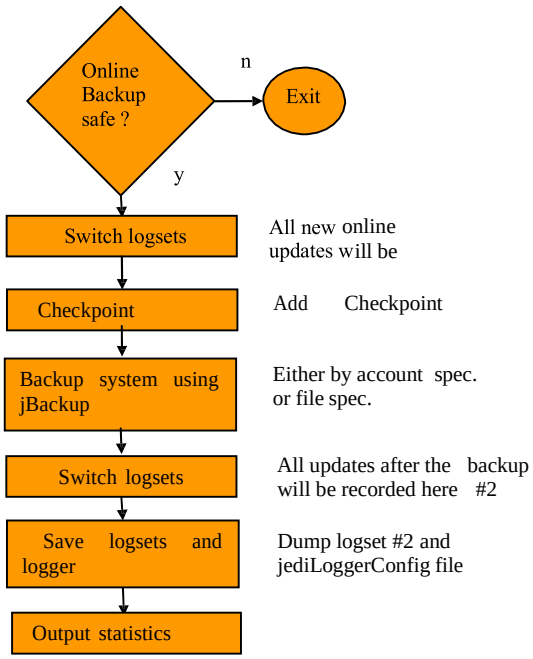
The tapes now produced are now written in the following manner :



The backup is recorded on the media, followed by the Transaction Journal configuration file (found at $JBCRELEASEDIR/config (or Windows %JBCRELEASEDIR%\config)) for the default configurations. This is then followed by a dump of the Transaction Journal logset file data which was active during the backup. It is clear that all the information to restore the database to a consistent point is now available in one set of media. During the restore process, if specified, the whole recovery process will continue unattended.

Online Backup Operational Details

The following diagram and description describe the details of an Online Backup being performed.



Prior to the commencement of the Online Backup a determination is made of whether an Online Backup is safe with regard to Transaction Logging. Two tests must be passed: firstly transaction logging must be active. Online Backups depend on the Transaction Journaling system. Secondly, a minimum of three logsets must be configured. This constraint is to prevent the loss of data contained in the Transaction Logs. If this test is passed then:

A Checkpoint is performed. The database is paused for all updates. This pause will ensure that all transactions which have entered the commit phase (i.e. by the processing of either a TRANSEND or TRANSABORT instruction) will be allowed to complete. This will ensure that all committing transactions will be entirely contained within the current logset. As all committing transactions are allowed to complete, and new transactions are held off from committing, this ensures that the database at this point of time is in a consistent transactional state.

The Transaction Logs are switched to the next (by use of jlogadmin –lnext – internally by the program).

A Checkpoint marker (record) is entered into the new Transaction Logset as the first entry. This entry will have a type specification of “CHECKPOINT” and will have a textual string “Backup Started”. The use of this checkpoint marker is not entirely essential, but is does act as information for the system administrator.

The database is then resumed – any updating processes will be allowed to continue. All updates to the database will cause an entry in the Transaction Logset to be created.

Note:
No updates will be lost during this phase of the backup. The delay in time is dependent on the transaction rate prevalent on the computer at this time. This may or may not be noticible.

The backup now begins. This backup will be a snapshot of each specified file until the end of the specified set. With online processes being allowed to update the database at the same time that the database is being backed up, it is clear that some (many) files will have been backed up prior to updates to such files - for the duration of the backup. Each update during this time will have caused the Transaction Logset to be updated with such updates. Thus the backup set will comprise the snapshot of all files specified for backup plus the image of the updates contained within the current logset. During the restore process, this Transaction Logset will be used to roll forward the database for those updates which occurred during the backup.

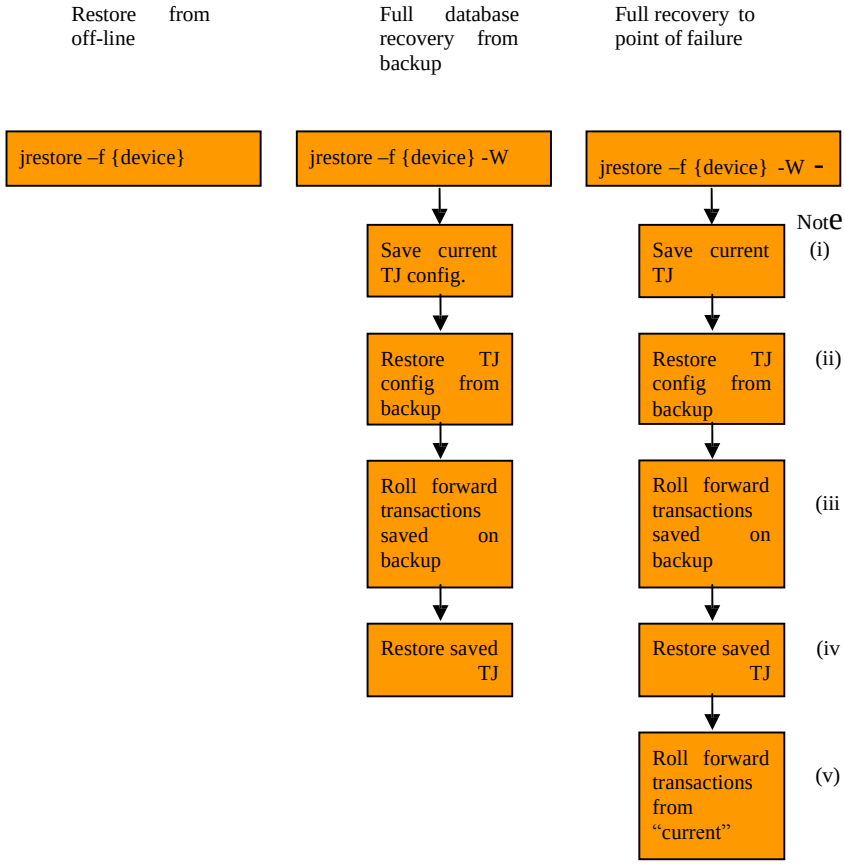
Upon completion of the backup the database will again be paused. All processes which are in the commit phase of transactions will be allowed to continue.

A Checkpoint marker is now entered into the Transaction Logset at the next position. This entry will have a type specification of "CHECKPOINT" and will have a textual string "Backup Ended". Thus it can be seen that the updates to the database during the backup will be recorded in the Transaction Logset bounded by CHECKPOINT records indicating the start and end of the backup.

Once this marker has been written, the logsets are again switched to the next available logset. Note now that this whole process has used three logsets in a very clean manner. The logset which was current prior to the commencement of the backup will be preserved (possibly for later archiving); the logset which was current at the time the backup ran is preserved and lastly a new current logset is made available for future updates. It will be seen later the relevance of this latter logset during the restore process.

The database is now resumed, allowing all updating processes to continue. Note that all transactions which enter the commit phase will be contained in the latest current logset in their entirety.

The Transaction Logset which was current during the backup and the Transaction Journal configuration file (jediLoggerConfig) are now archived to the same medium and following the database backup. These two components will be required during the restore process to return the database to a consistent state.



Recovering the database from backup media

Important: No users can be allowed on the system during a system restore.

The functionality of the restore process, jrestore, has been extended to allow for the automatic roll-forward of logsets after a database restore has completed. There are three options which an administrator may use in the recovery of the database :

A database restore from the last backup – no option required. This is the standard restore process and would be used following an off-line backup. If the backup was an online backup and updates were made to the database while the system was being backed up, then the database may (will) be left in an inconsistent state.

A full system recovery (option –W): this will return the database to the state it was at the completion of the backup. The database will be in a consistent state. Functionally, this procedure is: jrestore (all files are recovered); roll-forward of the Transaction Journal files – these files follow the standard backup on the backup media.

A full system recovery followed by a roll-forward of all transactions which occurred following the last backup to the point of failure (options –W –G). This will leave the database in a consistent state. Only complete transactions will be recovered. It is the

responsibility of the database administrator to determine which transactions have not been recovered by this mechanism. These transactions will be those which were incomplete at the point of failure.

Notes:

(i) The various operations are defined by the options chosen.

(ii) The Transaction Journal (if present) is saved. If the required recovery is to the point of failure, then this configuration will be used for the roll forward of all transactions since the backup completed.

(iii) The Transaction Journal was saved following the database on the same media. This configuration reflects the state of journaling, including journal files upon completion of the backup. The "current" logset indicated in this configuration is that logset which was in use during the backup.

(i) If a Transaction Journal configuration file existed prior to the commencement of the recovery process, then this is restored. If no configuration existed at this time (media failure perhaps – full system recovery), then the configuration is that which was in force at the end of the backup.

(v) If a full database recovery to the point of failure is specified, then a roll-forward of all completed transactions takes place using the "current" logset as defined in the configuration.

Example 1:

It is required to return the database to a consistent state following a detected problem since the last backup finished. In the case where the database and logsets reside on the same disk set and that disk has become corrupt or unusable, (and the Transaction Journal is not being copied to offline media), it is not possible to recover any transactional data entered since the end of the last backup.
The command to use in this instance could be:

```
jrestore -f /dataset/backuptues -W
```

This assumes that the database will be recreated by the standard jrestore mechanisms. Following this restore, the Transaction logfiles which were saved during the online backup are restored along with the corresponding jediLoggerConfig file. A roll-forward command of the form:

```
jlogdup  input_set=n terminate=eos output_set=database
```

is used internally to roll-forward the database. The set number specification is retrieved from the loaded jediLoggerConfig file as being the current logset. The current logset this refers to is that configured logset which was current for the duration of the online backup. The location of the backup file must be known by the administrator (in this case /dataset/backuptues).

At the completion of this internal command the database is returned to a consistent state, indeed the state of the database at the completion of the backup.

Example 2:

In this example the database and logsets are stored on different disk sets physically. In the event that the database disk is lost, it is possible, not only to restore the database to a consistent state, but also possible to roll-forward all transactions which have occurred since the end of the last backup, using the logset stored on another disk. The important configuration required for this to succeed is that the jediLoggerConfig file cannot exist on the database disk but on the disk where the Transaction Logfiles are stored. This is achieved by the use of the JBCLOGCONFDIR environment variable. This variable defines a directory/folder called "config" somewhere on the logfile-resident disk. The contents of this config directory will be created automatically when logfiles are configured by the jlogadmin facility.

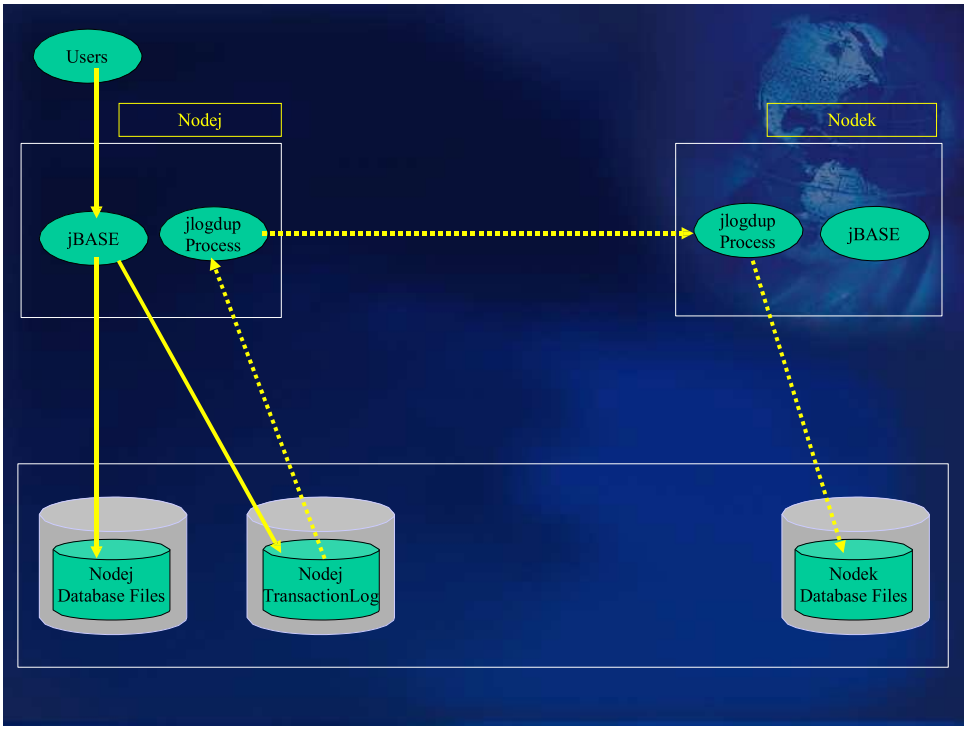
The command to run in this instance is :

```
jrestore -f /dataset/backuptues -W -G
```

The use of the "-G" option changes the restore operation thus:
Prior to restoring the jediLoggerConfig file from the backup medium, the jediLoggerConfig file which is currently in use (as specified by the JBCLOGCONFDIR variable), is saved to disk. The operations are then performed exactly as per example 1 and then the following occurs:

The saved jediLoggerConfig file is restored and is then used in a further embedded jlogdup command. This time the set to use for input is taken from the jediLoggerConfig just restored from disk. This is the set which was active at the start of the jrestore operation (in contrast to the set which was current during the backup). This roll-forward will update the database with all updates since the end of the backup, in transactions which have committed in their entirety.
This ensures that the database will again be left in a consistent state.

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 established.
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 inability to cope with the database update rate may cause the live and standby machines' database to be unacceptably out-of-sync. Too many disk-based transaction log entries on the live machine may 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).

An Online Backup is taken on Nodej. This will produce a backup set containing the database backup, the Transaction Journal configuration on Nodej and the Transaction Journal logset which was current during the backup.

e.g.

```
jfind /JBASE_APPS /JBASE_SYSFILES /JBASE_ACCOUNTS -print |
jbackup -f {device name}
```

This is now restored on Nodek to produce a consistent database as existed when the backup completed on Nodej

e.g.

```
jrestore -f {device name} -W
```

Functionally this is a database restore followed, automatically, by a roll-forward.

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 Nodek. This could be achieved with:

On Nodek the following command could be entered:

```
jlogdup input set=socket hostname=Nodek port=4089
terminate=wait output set=database
```

This will create a process which will “listen” for connection requests on socket 4089 on this computer. When such a request is fielded, then Transaction Journal records will be received, processed and updated onto the database on Nodej. This process will wait indefinitely for further data.

On Nodej:

```
jlogdup input set=current terminate=wait output set=socket
hostname=Nodek port=4089
```

This will request a connection on port 4089 on Nodek. Once accepted Transaction Journal records will be transferred from the current logset over the network to the receiving process on Nodek. This process will wait for further updates on Nodej and transfer them as and when they exist. At this point Nodek will keep in sync with updates made on Nodej (albeit slightly behind).

The status of the jlogdup processes can be monitored by running jlogstatus from a dedicated window:

```
jlogstatus -r5 -a
```

on each computer.

Online backups may be made periodically on Nodej to ensure further security. Nodek will be kept up-to-date throughout these backups automatically.

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 watermark.

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 logfile updates to a maximum of the last 5 seconds. (If the "Sync Transactions" configuration is set, then this flush period is less important)

If the same disk on Nodej is used to hold both the database and the Transaction Journal, then a failure of this disk the data loss is limited to a combination of: 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 will show the latest updates which were actually updated on the standby machine, Nodek. Obviously, the transaction update rate on the live machine governs the possible 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 (or the global setting of "Sync Transaction" 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.
An Online Backup is taken on Nodek.
e.g.

```
jfind /JBASE_APPS /JBASE_SYSFILES /JBASE_ACCOUNTS –print |
jbackup –f {device name}
```

This is now restored on Nodej.
e.g.

```
jrestore –f {device name} –W
```

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

On Nodej the following command could be entered:

```
jlogdup input set=socket hostname=Nodej port=4089
terminate=wait output set=database
```

On Nodek:

```
jlogdup input set=current terminate=wait output set=socket
hostname=Nodej port=4089
```

Ensure the jBASE demons are started.

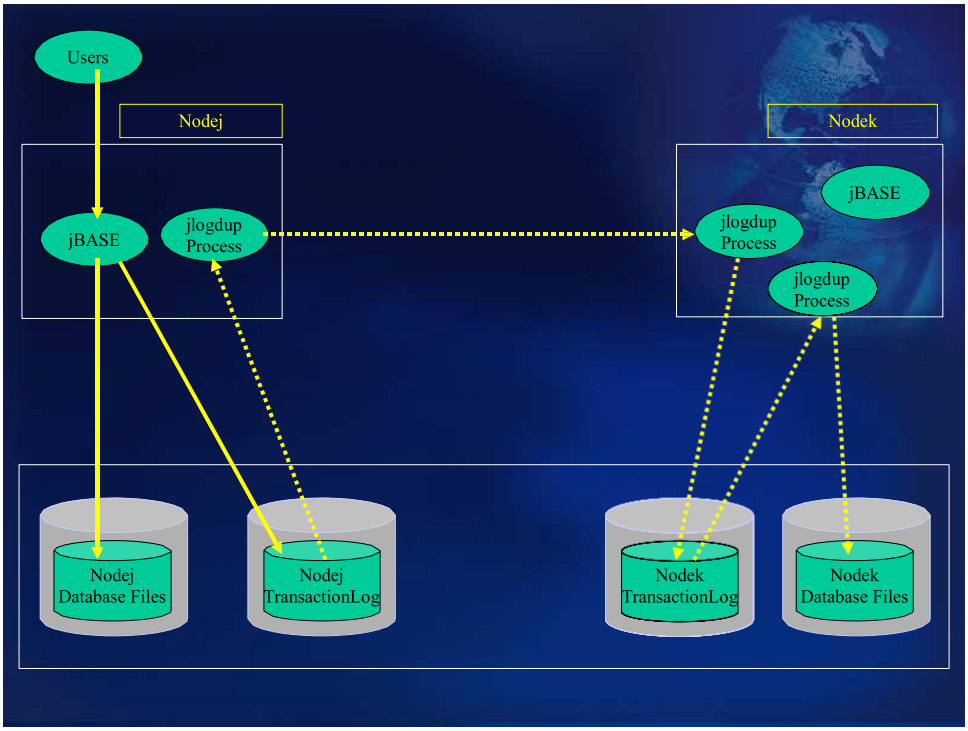
Enable jBASE logons. At this point it is safe for users to start using the system. Any updates since the start of the backup will be logged in the TJ log.

Once the 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=socket  hostname=Nodej port=4089 terminate=wait 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:

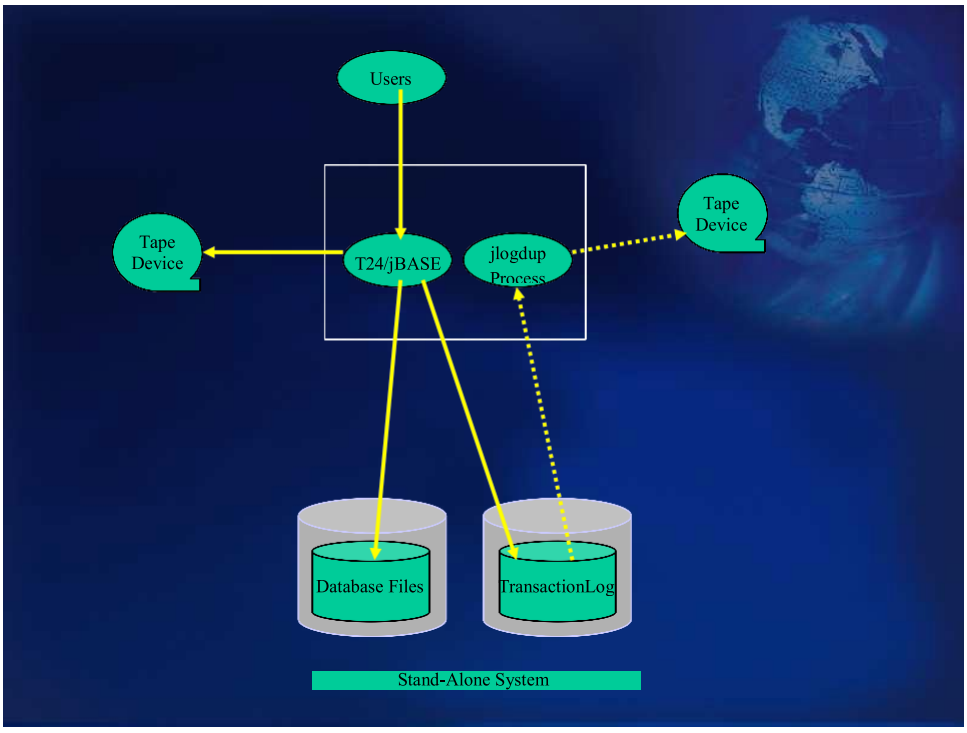
```
jlogdup input set=current output set=database
```

which 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 TRANSTART, TRANSEND and TRANSABORT programming commands. 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:
/bnk/bnk.jnl/logset1
/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.
NOTE2: 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.

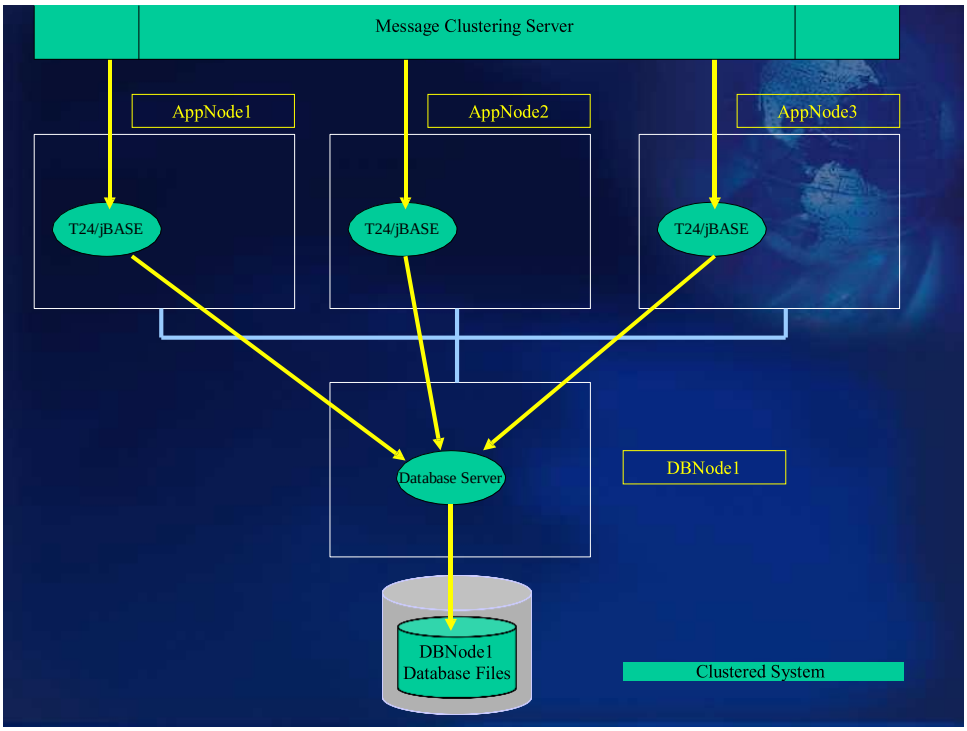
If there has been an application/database error which has resulted in the decision to perform a complete restore of the system, it is clear that if the error can be identified to have taken place at a particular time, (whether precisely or approximately), then the whole of the transaction log should 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 completed (i.e. 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 which 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.

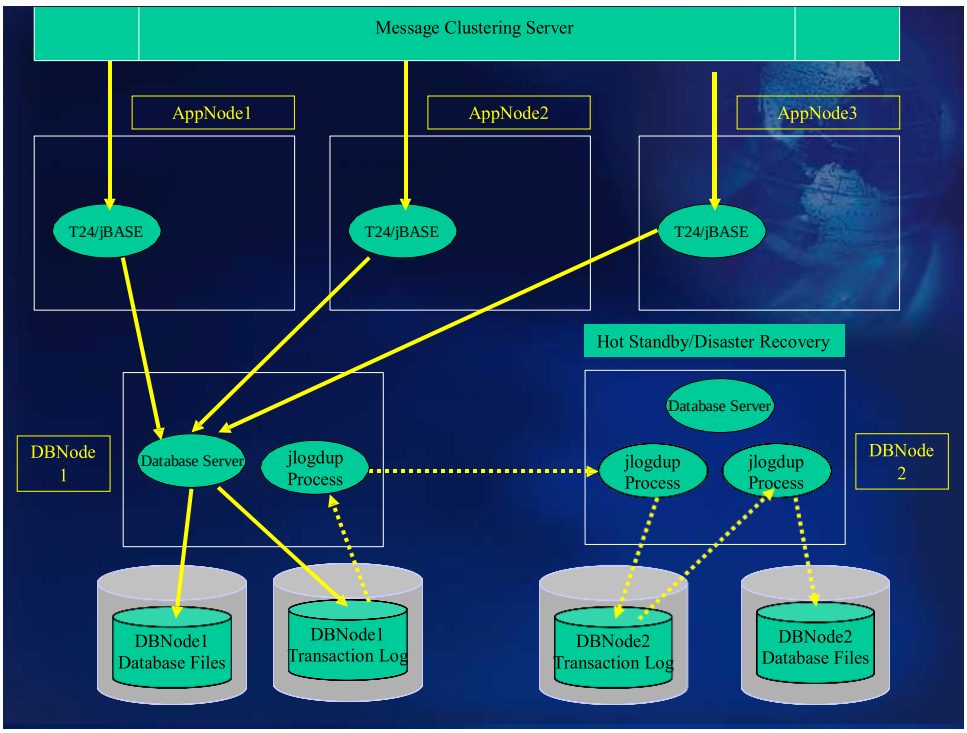
On the Database Server (1) – Application Servers communicate over NFS mounts to RDBMS stub files defined on the Database Server. The downside of this approach is that RDBMS client components (at least) have to exist on each of the Application Servers. Also there is a problem with managing database locks. 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.

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

Note 1: For Windows, each of these names should have a file type of ".cmd"

Note 2: The following are replacements for the variable LD_LIBRARY_PATH depending on platform.
LIBPATH - Pathnames of system libraries (AIX only)
SHLIB_PATH - Pathnames of system libraries (HPUX only)

warmstart

The content of the script/command for a Linux computer is:

```
JBCRELEASEDIR=/usr/jbc
JBCGLOBALDIR=/usr/jbc
PATH=$PATH:$JBCRELEASEDIR/bin
LD_LIBRARY_PATH=$JBCRELEASEDIR/lib
export JBCRELEASEDIR JBCGLOBALDIR
DB-START -nwarmstart
DB-WARMSTART
DB-REMOVE -nwarmstart
```

This script should be invoked from the /etc/rc.d/rc.local script thus :

For Windows computers the content of the batch file "WARMSTART.cmd" is :

```
set JBCRELEASEDIR=c:\jbase4.1
set JBCGLOBALDIR=%JBCDATADIR%
SET JBCOBJECTLIST=%JBCRELEASEDIR%\lib
SET JEDIFILEPATH=%HOME%;.
SET PATH=%PATH%;%JBCRELEASEDIR%\bin
SET JBASE_DATABASE=warmstart
DB-START -nwarmstart
DB-WARMSTART
DB-REMOVE -nwarmstart
```

This command should be appended to the HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run hive

setup_tj

For Unix/Linux:

```
#! /bin/ksh
```

```
export JBCRELEASEDIR=/data/reldir/jbcdevelopment
export JBCGLOBALDIR=/data/reldir/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
```

For Windows:

```
@ECHO OFF
set JBCRELEASEDIR=c:\jbase4.1
set JBCGLOBALDIR=c:\jbase4.1
set PATH=%JBCRELEASEDIR%\bin;%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
```

e.g. jlogadmin –c –f1,1,E:\logset1\logfile1 will create a logfile called logfile1 in directory E:\logset1. Note: The folder logset1 must exist.

start_tj

For Unix/Linux:

```
#! /bin/ksh
export JBCRELEASEDIR=/data/ reldir /jbcdevelopment
export JBCGLOBALDIR=/data/ reldir /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] &
```

For Windows:

```
@ECHO OFF
set JBCRELEASEDIR=c:\jbase4.1
set JBCGLOBALDIR=c:\jbase4.1
set PATH=%JBCRELEASEDIR%\bin;%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

For Unix/Linux:

```
#! /bin/bash
export JBCRELEASEDIR=/data/reldir/jbcdevelopment
export JBCGLOBALDIR=/data/reldir/jbcdevelopment
export LD_LIBRARY_PATH=$JBCRELEASEDIR/lib:$LD_LIBRARY_PATH
jlogadmin –a Off
```

For Windows:

```
@ECHO OFF
set JBCRELEASEDIR=c:\jbase4.1
set JBCGLOBALDIR=c:\jbase4.1
set PATH=%JBCRELEASEDIR%\bin;%PATH%
jlogadmin –a Off
```

start_jlogdup

For Unix/Linux:

```
#! /bin/ksh
export JBCRELEASEDIR=/data/ reldir /jbcdevelopment
export JBCGLOBALDIR=/data/ reldir /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]&
```

For Windows:

```
@ECHO OFF
set JBCRELEASEDIR=c:\jbase4.1
set JBCGLOBALDIR=c:\jbase4.1
set PATH=%JBCRELEASEDIR%\bin;%PATH%
date /t > %JBCRELEASEDIR%\config\jlogdup_to_tape_start
jlogdup input set=current terminate=wait output set=serial device=[Device Spec]&
```

e.g. jlogdup input set=current terminate=wait output set=serial device=c:\temp\logdupop

stop_jlogdup

For Unix/Linux:

```
#! /bin/ksh
export JBCRELEASEDIR=/data/ reldir /jbcdevelopment
```

```
export JBCGLOBALDIR=/data/ reldir /jbcdevelopment
export LD_LIBRARY_PATH=$JBCRELEASEDIR/lib:$LD_LIBRARY_PATH

jlogadmin -k* > discard
```

For Windows:

```
@ECHO OFF
set JBCRELEASEDIR=c:\jbase4.1
set JBCGLOBALDIR=c:\jbase4.1
set PATH=%JBCRELEASEDIR%\bin;%PATH%
jlogadmin -k* > discard
```

backup_jbase

For Unix/Linux:

```
#! /bin/ksh
export JBCRELEASEDIR=/data/reldir/jbcdevelopment
export JBCGLOBALDIR=/data/reldir/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
```

For Windows:

```
@ECHO OFF
set JBCRELEASEDIR=c:\jbase4.1
set JBCGLOBALDIR=c:\jbase4.1
set PATH=%JBCRELEASEDIR%\bin;%PATH%

0001 OPEN "","C:\jbase4.1\config" TO CONFIG_LOGS ELSE STOP 201
0002 CRT "Are you backing up the logfiles to tape? (Y/N)"
0003 INPUT tapeout
0004 tapeout = UPCASE(tapeout)
0005 IF tapeout = "Y"  THEN
0006     PRINT "Has all logging to tape finished - press any key when it has":
0007     INPUT reply
0008     EXECUTE "jlogadmin -k*" CAPTURING RESULT
0009     CRT RESULT
0010     CRT "Please remove the tape for logging and replace with the backup tape"
0011     CRT "Enter any key to continue":
0012     INPUT REPLY
0013 END
0014 backupok = "N"
0015 LOOP WHILE backupok # "Y"
```

```
0016     EXECUTE  "jfind C:\jdata4.1\bp -print" CAPTURING ALLFILES
0017     DATA ALLFILES
0018     EXECUTE "jbackup -c -f C:\temp\backup_file"
0019      *EXECUTE  "jfind C:\jdata4.1\bp -echo | jbackup -v -c -f [Device Spec]   -s
C:\tmp\jbstart"
0020     CRT "Waiting for tape to rewind"
0021     SLEEP 5
0022     CRT "echo Verify Started ": TIME() "MTS"
0023     EXECUTE "jrestore -P -f c:\temp\backup_file"
0024     *EXECUTE "jrestore -f [Device Spec] -P"
0025     CRT "Verify Finished at ": TIME() "MTS"
0026     CRT "Was the backup successful <Y/N>"
0027     CLEARDATA
0028     INPUT backupok
0029     backupok = UPCASE(backupok)
0030 REPEAT
0031 EXECUTE "jlogadmin -l next -a Active"
0032 CRT "Logsets switched and logging to disk restarted"
0033 IF tapeout = "Y" THEN
0034     CRT "Mount a new tape for logging"
0035     CRT "Enter any key to resume logging to tape "
0036     INPUT input
0037     STARTTIM =  TIME()
0038        WRITE  STARTTIM  ON  CONFIG_LOGS,"jlogdup_to_tape_start"  ON
ERROR DEBUG
0039        EXECUTE  "jlogdup  input  set=current  terminate=wait  output set=serial
device=C:\temp\logoutput"
0040       * EXECUTE  "jlogdup input set=current terminate=wait output   set=serial
device=[Device Spec]"
0041     CRT "Logging to tape restarted"
0042 END
```

recover_jbase

For Unix/Linux:

```
#!/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
```

For Windows:

```
 IF NOT (SENTENCE(1)) THEN
    REPLY = ""
    LOOP WHILE REPLY NE "F" AND REPLY NE "T"
       CRT "What is the nature of the recovery? F=Full recovery required, T=Tape logging failure"
       INPUT REPLY
    REPEAT
 END
 IF REPLY = "F" THEN
    CRT "Put the first backup tape in the tape drive"
    CRT "Enter any key when ready ":
```

```
    INPUT DONE
    EXECUTE "jrestore -f C:\temp\backup_file"
    * EXECUTE "jrestore -f [Device Spec]"
    CRT "Is a Transaction Log tape available ? ":
    INPUT REPLY
    CRT "Put the first log tape in the tape drive"
    CRT "Enter any key when ready ":
    INPUT DONE
    CRT "Enter a time to terminate the duplication process (or RETURN for all logs)
":
    INPUT ENDTIME
    IF NOT (ENDTIME) THEN
    EXECUTE "jlogdup input set=serial device=C:\temp\logoutput terminate=EOS
output set=database"
    *       EXECUTE "jlogdup input set=serial device=[Device Spec] backup
terminate=EOS output set=database"
    END ELSE
       EXECUTE "jlogdup input set=serial device=C:\temp\logoutput
end=ENDTIME output set=database"
    * EXECUTE "jlogdup input set=serial device=[Device Spec] end=$ENDTIME
output set=database"
    END
 END ELSE
    CRT "Put a new tape in the tape drive"
    CRT "Enter any key when ready ":
    INPUT DONE
    IF GETENV("JBCRELEASEDIR", RELDIR) = "" THEN STOP
    EXECUTE "jlogdup input set=current
start=RELDIR:"config":JBUILD_DELIM_CH:jlogdup_to_tape_start terminate=wait
output set=serial device=[Device Spec]"
 END
```