



jBASE Files



Copyright

Copyright (c) 2007 TEMENOS HOLDINGS NV

All rights reserved.

This document contains proprietary information that is protected by copyright. No part of this document may be reproduced, transmitted, or made available directly or indirectly to a third party without the express written agreement of TEMENOS UK Limited. Receipt of this material directly from TEMENOS UK Limited constitutes its express permission to copy. Permission to use or copy this document expressly excludes modifying it for any purpose, or using it to create a derivative therefrom.

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

jBASE, jBASIC, jED, jSHELL, jLP, jEDI, jCL, jQL, j1, j2 j3 j4 and jPLUS files are trademarks of TEMENOS Holdings NV.

REALITY is a trademark of Northgate Solutions Limited.

PICK is a trademark of Raining Data Inc.

All other trademarks are acknowledged.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Windows, Windows NT, and Excel are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names used in this publication may be trademarks or service marks of others.

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 Peoplebuilding

Hemel Hempstead

Hertfordshire

HP1 1JY

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

Documentation Conventions	1
JBASE FILES	3
CLEAR-FILE	3
COPY	4
CREATE-FILE.....	6
DELETE.....	7
DELETE-FILE	8
jbackup	9
jrestore	13
jgrep.....	17
jrf	19
jstat	21
SEL-RESTORE.....	25
DISTRIBUTED FILES.....	27
OVERVIEW	27
PART FILES.....	28
CREATING DISTRIBUTED FILES.....	29
ATTACHING AND DETACHING PART FILES	30
PARTIONING ALGORITHM.....	31
System Partition Algorithm.....	31
User-defined Partition Algorithm.....	32
DISTRIBUTED FILE COMMANDS.....	35
CREATE-DISTRIB.....	35
DELETE-DISTRIB	38
LIST-DISTRIB.....	39
VERIFY-DISTRIB.....	40
DISTRIBUTED FILE CONSIDERATIONS	41
DISTRIBUTED FILES EXAMPLE.....	42
JBASE TRIGGERS	44
OVERVIEW	44
SUBROUTINE PARAMETER DESCRIPTION	44
filevar.....	44
Event.....	44
prerc.....	45
flags	45
RecordKey.....	45
Record	45
userrc	45
ASSIGNMENT OF TRIGGER SUBROUTINE ARGUMENTS.....	46
CREATE-TRIGGER.....	47

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 <i>Italic</i>	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.

JBASE FILES

jBASE can handle data in a variety of forms. Support is built in for access to hashed data files (the default) as well as directories and sequential files. Data held in other forms can be accessed through the use of an appropriate driver. Regardless of where the data is stored, the access mechanisms are the same. This document covers the different file types and the commands used to manipulate files within jBASE.

CLEAR-FILE

The CLEAR-FILE command allows the user to clear all records from the dictionary file or data section files.

COMMAND SYNTAX

```
CLEAR-FILE {DICT|DATA} filename{,section}
```

SYNTAX ELEMENTS

filename is the name the file to be cleared. The file type must be one of the supported jBASE file types. If the file type supports separate dictionary and data files, the DICT or DATA keywords may be used to clear either the dictionary file or the data file. DATA is assumed by default. section is the name of the multiple data section to be cleared.

EXAMPLES

```
CLEAR-FILE File1
```

or

```
CLEAR-FILE DATA File1
```

Clears the data portion of file File1.

```
CLEAR-FILE DICT File1
```

Clears the dictionary portion of file File1 (File1JD).

COPY

The COPY command copies specific or selected records from a specified file to the terminal, printer or another file.

COMMAND SYNTAX

```
COPY {DICT} filename{,section} {recordlist} {(options)}
```

PROMPT

```
TO: {(DICT) filename{,section}} {targetrecordlist}
```

SYNTAX ELEMENTS

filename is the name of a valid file. The file type must be one of the supported jBASE file types. The DICT keyword can be used to specify that the record or records should be copied from or to a dictionary file.

section is the name of the multiple data section, if specified.

recordlist is the list of records (keys) to be copied. If recordlist is omitted, the active SELECT list is used, if present.

Options;

A	Force ASCII mode. Newline becomes field mark and vice versa.
B	Force binary mode. No newline conversion.
D	Delete source record after it is copied.
F	Output new page between keys. Used with T or P option.
I	Suppress or display record keys as records are copied. Emulation dependant.
N	Suppress pagination. Used with T or P option.
O	Overwrite record in target file, if it already exists.
P	Direct record contents to the spooler.
T	Direct record contents to the terminal.
S	Suppress line numbers, for T or P option.
X	Hexadecimal display, for T or P option.

targetrecordlist is a list of record keys to copy the records to - effectively renaming the copied records. Each key in the targetrecordlist is applied in sequence to the copied records. If targetrecordlist is not specified or contains less keys than there are copied records, the original record keys will be used.

NOTES

If you enter <RETURN> at the TO: prompt, the records will be copied to the terminal screen or spooler, depending on the options chosen. In this case, the D(elete) option will have no effect.

EXAMPLE

COPY File1 Record1 (T

Copies Record1 from File1 to the terminal.

COPY File1 Record1 (OD

TO:(DICT File2 Record2

Copies Record1 from File1 to dictionary file File2JD overwriting Record2. Once copied, the original Record1 is deleted from File1.

CREATE-FILE

The CREATE-FILE command allows the user to create a new file for use within the jBASE system. The command will allow creation of any file type known to the jEDI libraries - unless the concept of a file is redundant to that target system.

COMMAND SYNTAX

```
CREATE-FILE {DICT|DATA} filename{,section} {HASHMETHOD=nnn} {TYPE=tname}
  {PERM=nnn} {LOG=YES|NO} {TRANS=YES|NO} {BACKUP=YES|NO}
  {NumBuckets{, BucketMult{, SecSize}} } {NumBuckets{, BucketMult{, SecSize}} }
```

SYNTAX ELEMENTS

DICT	An optional keyword used to specify that the command should create a dictionary file only for the filename.
DATA	An optional keyword used to specify that the command should create a data section file only for the filename.
filename	The name of the file that is to be created.
section	The name of a new multiple data section to be created.
HASHMETHOD	Used when a hash file is to be created. The numeric parameter nnn specifies the hashing method to be used when accessing the file. The default method of 2 works very well with all sorts of key types. However if the record keys will be perfectly uniform numeric keys then there may be a slight advantage to using method 1 on the file.
TYPE	Specifies the type of file to be created. Supported types include <ul style="list-style-type: none"> • J3 – files with configurable levels of flushing • J4 – standard hash files (the default) • JP – jPLUS files • UD – Directory files used to access files in the OS
PERM	Used to set the permissions of the file in exactly the same manner as the UNIX chmod command. nnn is an octal number that will be masked by the current umask setting. The default the value of nnn is 666.
LOG	LOG=YES NO allows the file to be included in or excluded from, the record or transaction logging mechanism, if licensed on your

	system. The value is set to YES by default.
TRANS	TRANS=YES NO allows the file to be included in or excluded from any transaction boundaries that are defined by an executing program. The value is set to YES by default.
BACKUP	BACKUP=YES NO allows the file to be included automatically by the jBASE jbackup utility. The value is set to YES by default.Options

DELETE

The DELETE command deletes specific or selected records from a specified file.

COMMAND SYNTAX

```
DELETE {DICT} filename{,section} {recordlist}
```

SYNTAX ELEMENTS

filename	The name of a valid file. The file type must be one of the supported jBASE file types. If the file type supports separate dictionary and data files
DICT	May be used to delete records from the dictionary file.
section	section is the name of the multiple data section if specified
recordlist	The list of record keys to be deleted. If the recordlist is omitted the active SELECT list will be used if present

EXAMPLE

```
DELETE File1 Record1
```

Deletes record Record1 from the file File1.

```
GET-LIST DeleteList
```

```
DELETE File1
```

Deletes all records from File1 that match the record keys selected by the active select list..

DELETE-FILE

The DELETE-FILE command allows the user to delete complete file sets, the dictionary, or the data section of a file.

COMMAND SYNTAX

```
DELETE-FILE {DICT|DATA} filename{,section}
```

SYNTAX ELEMENTS

filename	Name the file to be deleted. The file type must be one of the supported jBASE file types. If the file type supports separate dictionary and data files, the DICT or DATA keywords may be used to delete either the dictionary file or the data file.
section	Name of the data section to be deleted. NOTES

NOTES

The command will detect inconsistencies in its use and issue suitable error messages.

Warning: Beware of creating a file and then immediately deleting it using the DELETE-FILE command. The DELETE-FILE command will respect the JEDIFILEPATH variable and if it finds a file of the same name in a directory earlier in the path than the current working directory it will delete that file. For this reason it is best to define the JEDIFILEPATH variable as '.' (the current working directory):

EXAMPLE

```
DELETE-FILE File1
```

Deletes the complete file set of File1, comprising the dictionary file, default data section and any multiple data sections.

```
DELETE-FILE File1,Section
```

Deletes the multiple data file named Section1 from File1.

```
DELETE-FILE DATA File1
```

Deletes the default data file named File1.

```
DELETE-FILE DICT File1
```

Deletes the dictionary file of File1 (File1]D).

jbackup

The jbackup utility provides fast on-line backup facilities and can also be used to check file integrity.

COMMAND SYNTAX

```
jbackup -Option {Inputlist}
```

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

OPTIONS

- bn set number of write buffers to n (default is 8, minimum is 1)
- 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 file (not available on windows)
- 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 (Windows only)
- N suppress compression if supported by device (Windows 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)
- E1 will make jbackup pause and give the user an option to quit if it encounters corrupt files
- E2 same as -E1 but will also pause for warnings

NOTES

This command will set the JEDIFILEPATH environment variable to “.” to prevent the backing up of incorrect files.

Use jchmod to change the characteristics of a file. For example,

```
jchmod -B filename
```

will cause jbackup to skip 'filename'. Other options of interest are +B, -O and +O.

jbackup creates a file named jbk*PID as a work file when executed. Therefore, jbackup must be run from a directory which has write privileges. If the file system or directory is not write enabled you will receive the error message ERROR! Cannot open temporary file jbk*PID.tmp, error 2

See also jrestore.

EXAMPLES

Unix

```
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 1 Mbyte.

```
jbackup -Ajbase -S/usr/jbc/tmp/jbase_stats >/dev/null
```

```
LIST /usr/jbc/tmp/jbase_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.

Windows

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

Reads all records, files and directories under the C:\users\vanessa directory provided by the jfind selection and displays each file or directory name as it is encountered. The -P option means that the files are not actually backup (print and scan only). It is useful to verify the integrity of the selected files and directories. This command should be run with jshelltype sh rather than jsh.

```
jfind D:\data -print | jbackup -f C:\temp\save20030325 -m10000 -S C:\temp\stats -v
```

The jfind command outputs the names of all the files and directories under the D:\data directory. This output is passed to the jbackup command causing it to backup every file that jfind locates. Rather than save to tape, this jbackup command creates a backup file: C:\temp\save20030325. Note that jbackup creates the save2003025 file, but the directory c:\temp must exist before running the command. The -m10000 option specifies that the maximum amount of data to back up is 10,000MB (or 10GB) rather than the default 100MB. The -S option causes file statistics to be written to the hashed file stats. This file should exist and be empty prior to commencing the backup. The -v option causes the name of each file to be displayed as it is backed up. Because of

the pipe character used to direct the output of `jfind` to `jbackup`, this command should be run with `jshelltype sh` rather than `jsh`.

jrestore

The jrestore utility provides fast on-line restores from the saves produced by the jbackup utility. The jrestore can be controlled to restore from any file type on the backup, from single records to multiple directories. The jrestore utility can also be used to verify jbackup saves.

COMMAND SYNTAX

```
jrestore -Options
```

OPTIONS

- a restore from current media position
- bn set number of read buffers to n (default is 8, minimum is 1)
- c"old new" restore old directory path as new directory path
- d"DirRE" restore directories matching regular expression
- f Device restore from device file, default stdin
- h"HashRE" restore hash files matching regular expression
- H FileList restore files using only file names from FileList file
- i"ItemRE" restore items matching regular expression
- I ItemList restore items using only item ids from ItemList file
- l"LnkdRE" restore links matching regular expression
- n control info files not restored
- N control info files restored and indexes rebuilt
- o"OfileRE" restore other files matching regular expression
- pn set priority, nice value of parent process
- u"UfileRE" restore normal file matching regular expression
- v verbose mode
- F use fixed block device. Use for QIC tapes
- B force block size to 128k, default 16k
- Cn force block size to n bytes, rounded to nearest k
- P print and scan files only, no restore
- O overwrite existing files and records
- R suppress rewind last reel

- T type restore hash files as specified file type; the original modulo and separation will be retained rather than use the 'resize' parameters.
- U update only does not overwrite existing files or records
- V verbose dot mode, displays a "." for each file

NOTES

When using jrestore ensure that you are executing at the standard shell not in jsh otherwise the double quotes and other meta characters will lose their meaning.

Windows Note: When specifying path names the backslash must be escaped with a backslash, otherwise the backslash character is removed. e.g.

```
C:\\MyApp\\new
```

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 ids containing the string SMITH.

jchmod

The jchmod command enables you to modify jBASE-specific file attributes such as the resize parameters of a hashed file.

COMMAND SYNTAX

```
jchmod {+options} {-options} file{ file...}
```

SYNTAX ELEMENTS

+options	
+B	Flag the file to be saved by the jbackup utility.
+L	Flag the file to be logged to the record or transaction log.
+T	Flag updates to be included as part of transaction boundaries.
+R{parms}	Set the resize parameters. The syntax is the same as for the CREATE-FILE command and includes spaces. The spaces should be retained by quoting the string from the shell.
-options	
-B	Flag the file NOT to be saved by the jbackup utility.
-L	Flag the file NOT to be logged to the record or transaction log.
-T	Flag updates NOT to be included as part of transaction boundaries.
-R	Remove any resize parameters from the file.
-t	Tabulate and display statistics of the files. Note this option is exclusive and will cause all other options to be ignored.
file{ file...}	A list of all the 'real' file names to be processed by the command.

NOTES

jchmod will ignore files that are not jBASE supported files, or where options do not apply to certain file types. This allows the command to be issued against an entire directory without determining which files are valid for the command. A warning message will be issued for any ignored files.

The 'real' file name should be specified to jchmod for dictionary and data sections of a file. The formats DICT file and file,section are not supported by this command.

jgrep

The jgrep utility enables pattern matching of records in one or more jBASE files.

COMMAND SYNTAX

```
jgrep {options} searchstring file {recordlist...}
```

SYNTAX ELEMENTS

- searchstring is the string to search for in the file(s). If the string contains spaces, surround it with quotes or use the -I option described below.
- file is the name of any valid file.
- recordlist is a list of record keys.

Options	Explanation
-C	Make the search case insensitive. This means that a search string of 'ABC' would match the string 'abc'.
-I	The search string(s) has been omitted from the command line and will be prompted for before searching the files in the list.
-L	List only the record keys that the search string(s) were found in.
-N	Do not wait for keyboard input between pages of output.
-P	Send all output to current printing queue. (Implies the -N option).
-S	Search all subdirectories. If the file specified is a UNIX directory, all the files in the directory and its sub-directories will be searched for the searchstring(s).
-r	Raw mode. Display record key, line number and occurrences field separated, one line at a time. Note that this option skips jBC object records located in hash files.

NOTES

Options may be specified after the file or recordlist by preceding the options with a left parenthesis:

```
jgrep searchstring file recordlist (options
```

The -S option will cause all records in a hashed file, or all files in a UNIX directory to be searched.

EXAMPLE

```
jgrep "ABC DE" SALES CUST0001
```

Searches the record CUST0001 in file SALES for the string 'ABC DE'.

```
find . -name "JAC*" -print | xargs jgrep -NS fast
```

Standard UNIX commands can be used to provide arguments to the jgrep command.

```
jgrep -ILSN .
```

Prompts for search strings and then searches the records in all files in the current directory, and all files in any subdirectories. Standard UNIX commands can be used to provide arguments to the jgrep command. Does not pause at the end of each page of output.

jrf

The jrf utility can be used to resize one or more files.

COMMAND SYNTAX

```
jrf {options} {filename{,section}}
```

SYNTAX ELEMENTS

- filename is the base name of the file to be resized. The file type must be one of the supported jBASE hash file types.
- section is the name of the data section to be resized.

Options can be one or more of the following:

Option	Explanation
-H3	Recreate files as HASH3 (j3) files. □
-H4	Recreate files as HASH4 (j4) files. □
-H5	Recreate files as jPLUS files
-B	Keep original bucket / frame size
-C	Change restore specification
-D	Allow downsize of file
-E	Resize empty files
-I	Ignore empty files. By default the user will be prompted to resize empty files.
-L	Don't journal the jRF operation
-Mn	Override default hash method, set to method n
-R	Reporting mode only. The files will not be changed.
-V	Verbose mode. Information is printed about each file size as the command progresses.
-V1	Very verbose mode. A jstat is performed on each file and the results printed.
-Sm{s,i}	Size to parameters; m – modulo, s – separation, i - ingroupmaxsz

NOTES

Badly sized files can cause severe performance problems. Often, over time internal file sizes become too small for the number of records they contain.

The jrf utility will resize files listed on the command line, selected via an active SELECT list or all hash files in the current directory. By default the jrf command will resize all valid hash files in the current directory.

Warning: This command should only be used on files that are not in use by an application. Although the files cannot be physically corrupted by jrf, they can be logically corrupted as updates by programs that have already opened the file will be lost.

jstat

The jstat command analyses a hashed file to provide statistics on the distribution of records and use of the data space.

COMMAND SYNTAX

```
jstat {options} {filename{,section}}
```

SYNTAX ELEMENTS

- filename is the base name of the file to be analysed. The file type must be one of the supported jBASE hash file types.
- section is the name of the data section to be analysed.

options are as follows:

Option	Explanation
-f	Display free space information.
-r	Display table entries for each bucket.
-v	Displays additional verbose information. When used with the r option displays each record key in the bucket.
-m	Display information in machine readable format.
-dchar	Specifies delimiter char to use as a field separator, for machine readable information supplied by the m option.

The -f option displays additional information relating to the organisation of free space within the file. The information allows a judgement to be made as to how fragmented the file has become. As records are added to the file, buckets are allocated for secondary space or to extend large groups. As records are deleted, empty buckets are added to the free space chain associated with the file. When new buckets are required to extend data space for new records, the buckets available on the free space chain will be used wherever possible. This fills up any 'holes' that may have been created in the file. If the file is very dynamic this process can create fragmentation of the free space and therefore the file. If this becomes excessive, the file can become very large in relation to the data it contains.

If there are a large number of buckets in the free space chain, in relation to the number of buckets allocated to the file, this may indicate that the file has become fragmented. Use the jrf utility to remap the file and remove fragmentation.

Two figures regarding freespace are given in the output (see notes). The Total buckets used for free space chain shows how many buckets have been used within the file to record the free space chain. If this is large, it is a good indication that the file has become fragmented. The Total unused (freed) buckets shows how many buckets have been created for extra storage space and then given back to the free space chain. If this is large in relation to the number of buckets in the file, this another indication of file fragmentation.

Bucket Information

The r option displays additional information regarding the efficiency of each bucket in the file. A table will be output showing the number of bytes allocated, and the number of bytes used, for each group in the file. This can then be used to judge the distribution of the data within file. You could also extract this information from the output stream using an awk script (or similar) and then perform a statistical analysis of the figures.

Additional Information

The v or verbose option displays the following additional information:

Restore re-size parameters Any resize parameters that have been set against the file using the jchmod command.

Last Accessed	Time and date of the last file access.
Last Modified	Latest time and date that the file was modified in any way.
File Size	Size of the file in bytes as UNIX sees it.□
Inode	UNIX inode number for the file.□
Device Id	UNIX device-id (normally the disc partition).□
Backup File	Whether the file is backed up by jbackup.□
Log File	Whether file updates are logged by record or transaction logger.
Trans Rollback	Whether file updates should be rolled back if a transaction fails or is aborted.

When the v option is combined with the r option, the key and size of each record in the buckets is also displayed.

Machine Readable Information

The `m` option displays all the information (excluding `-r` option) in machine readable format and thus allows resizing and analysing programs to extract information in a simple way. This option excludes all other options, except the `-d` option which allows the field delimiter to be changed to a character other than `<TAB>` (0x09).

Information from the `m` option is presented in the following order:

(1)	File Type	HASH1, HASH2 etc.
(2)	#Recs	The number of records in the file.
(3)	#Rbytes	The number of bytes used by the records.
(4)	FileLength	The length of the UNIX file.□
(5)	PBuckSz	The size of each primary bucket in bytes.
(6)	#TBuckets	The number of buckets contained in file.
(7)	AvgRecSz	The average size of each record.
(8)	AvgBucketSz	The average size of each bucket.
(9)	#PBuckets	The number of primary buckets.
(10)	#SBuckets	The number of secondary buckets.
(11)	#FBuckets	The number of buckets in the free space chain.
(12)	FSpace	Free space in bytes.
(13)	#Ublocks	The size of each bucket in UNIX file space blocks.□
(14)	SBuckSz	The size of each secondary bucket in bytes.

NOTES

In its simplest form (`jstat filename`) the command output will be as follows:

```
File filename
Type = HASH1 , Hash method = 2 , Created Tue Jul 14 02:58 1992
Buckets = 97 , BucketSize = 512 , SecondaryBucketSize = 512
Record Count = 5 , Record Bytes = 832
Bytes/Record = 166 , Bytes/Bucket = 8
Primary file space: Total Buckets = 97 , Total Bytes = 1996
Secondary file space: Total Buckets = 0 , Total Bytes = 0
```

Most of the fields are self-explanatory, those that are not are explained below:

`SecondaryBucketSize` is the secondary bucket size of a hashed file as calculated at file creation time by the `CREATE-FILE` command. It specifies the record size beyond which space will be allocated outside the hash bucket itself. The hash bucket will instead contain a pointer to the record.

Primary file space is the primary file space allocation. This shows how the file relates to the originally allocated number of buckets and allows the effectiveness of the file size to be judged. The total number of buckets should match or be close to the number originally allocated to the file.

When a record is to be stored in a bucket that does not have enough free space, the bucket is allocated an additional number of primary buckets to accommodate the new record. The number by which the total buckets shown here exceed the original number allocated is the number of times that this has occurred. If this number is large, you should consider resizing the file. This can be done in two ways:

1. If you have allocated enough buckets to cater for the number of records in the file, increasing the size of each individual bucket should allow the records to fit into the file better. The new bucket size should be large enough to cater for the average record size shown in the jstat output.

1. If there are many more records than the number of buckets, you should increase the number of buckets to reflect this. The new number of buckets should be equal to the number of records divided by the number of average size records that will fit into a single bucket. If the file is growing in size over time, you should allow for some additional expansion in your calculations. You may find that a combination of both the above techniques is necessary if the average record size is much greater than the current bucket size and there are many more records than allocated buckets.

If the records in your file are very large, it is usually more efficient to force all records into secondary file space rather than create a very large bucket size. In this case, recreate the file with a secondary bucket size of 0.

Secondary file space indicates how much of the file was allocated as pointers to secondary space in the original group. Where possible, this should only form a small proportion of the file.

In general, the file should be sized slightly larger than its 'perfect' size. Unless files are very badly sized, overall performance will only be affected marginally.

SEL-RESTORE

The SEL-RESTORE command restores all or specific records into a jBASE file from an ACCOUNT-SAVE or FILE-SAVE.

COMMAND SYNTAX

SEL-RESTORE targetfilename {recordlist} {(options)}

PROMPT

Name of Account on media : sourceaccountname

Name of File on media : sourcefilename

or

Number of File on media : sourcefilenumber

SYNTAX ELEMENTS

sourceaccountname	sourceaccountname is the name of the account on the media where the source file resides.
sourcefilename	sourcefilename is the name of the file on the media in which the source records reside.
sourcefilenumber	sourcefilenumber is the number of the file on the media in which the source records reside. Used with the N option.
targetfilename	targetfilename is the name of the file to which the records are to be restored.
recordlist	recordlist is a list of record keys to be restored.

options are as follows:

Option	Explanation
-A	Media is already positioned in the section containing the account where the source file is located.
-C	Restore all records from the current media position.
-F	Display file names as the media is searched for the source file.
-I	Display record keys as they are restored.
-N	Prompt for source file number.
-O	Overwrite records which already exist in the target file.
-S	Suppress the 'records already exist' messages

NOTES

The command prompts for the name of source account and file held on the media, unless the N option (restore by file number) has been used, in which case you will be prompted for the number of the source file on the media.

Before execution, any tape device should have been opened with the T-ATT command.

EXAMPLES

SEL-RESTORE NEWCODES * (I

Name of Account on media :PAYROLL

Name of File on media :TAXCODES

Restore all records from the file TAXCODES in account PAYROLL into the file NEWCODES.

The record key will be displayed as each record is restored.

SEL-RESTORE NEWCODES SINGLE (N

Number of File on media :22

Restore record SINGLE from file number 22 into the file NEWCODES.

DISTRIBUTED FILES

Overview

A Distributed file is a collection of existing files used primarily for the purpose of organizing data into functional groups. Each file within the collection is called a part file. A distributed file can contain up to 254 part files. The method for determining in which part file a record belongs is called the partition algorithm.

As a simple example, suppose your database consists of records which span 42 regions and you elect to distribute your data so that each part file contains all records for a specific region. With distributed files you would be able to process any one of the region part files independently of the others, or you would be able to process all 42 region part files collectively (i.e. as one database containing the records from all 42 regions).

Distributed files can also be used when the size of a file exceeds the size limit for the operating system (typically 2 gigabytes). This effectively permits file sizes to reach 254 times the maximum file size your operating system allows.

Part Files

The components of a distributed file collection are called part files.

Part files can have any name and can be any file type except a distributed file.

Part files can exist anywhere on the network, accessible via the JEDIFILEPATH environment variable, Q-pointers or F-pointers.

Each part file is assigned a part number when it is attached to a distributed file. The part number must be a positive integer in the range of 1 through 254 inclusive. This part number is an integral element as it is used by the partition algorithm to determine which part file the record belongs to.

Part numbers do not have to be sequential nor do they have to be continuous. It is quite valid, for example, to have 4 part files numbered 52, 66, 149 and 242.

A part file can belong to more than one distributed file although this imposes two restrictions:

1. The part file must always have the same part number for each distributed file to which it belongs.
2. All distributed files to which a part file belongs must use the same partition logic. In other words, when a record is written to the common part file, the partition algorithm for each distributed file must resolve the record's location in the same manner. This is only applicable when the distributed file uses the user-defined partition method.

The number of part files and the partition algorithm can be varied at any time throughout the life of the distributed file. Be aware that if the partition algorithm changes such that records that were normally written to one part file, using the original partition algorithm, might be written to another part file using the new partition algorithm. This could lead to unwanted duplication.

Another problem that can occur is the wrong file is accessed through the distributed file stub (i.e. the file to which the part files are attached to create the distributed file set; see Creating Distributed Files). Be aware that part files are resolved in the same manner as any other file in jBASE. For example, suppose two files exist with the same filename where one is resolved via an F-pointer (in \$JEDIFILENAME_MD) and the other is resolved via \$JEDIFILEPATH, and that the one in \$JEDIFILEPATH is our actual part file. What will happen is the actual part file will never be found because the file pointed to by the F-pointer will be found first, as indicated by the jshow -f command. To alleviate this problem, it is best to attach the files using a full explicit filepath (see here for further details on attaching/detaching part files).

Creating Distributed Files

A distributed file is created using the `CREATE-FILE` command with the qualifier `TYPE=DISTRIB`. This will create two files, a dictionary which is a Hash4 (currently fixed at mod3) and the distributed file stub. If desired, the dictionary can be resized using the `jrf` utility. For example, the following command creates a distributed file called `DISTREGION`:

```
jsh DEMO ~ -->CREATE-FILE DISTREGION TYPE=DISTRIB  
  
[ 417 ] File DISTREGION]D created , type = J4  
  
[ 417 ] File DISTREGION created , type = DISTRIB
```

The file partition table is empty at this point, and the partition algorithm is set to the default system partition method with a delimiter of `'-'` (i.e. all record IDs must be of the form "PartNumber-recordID"). These aspects of the distributed file can be changed with the `create-distrib` command.

Attaching and Detaching Part Files

Files are attached to a distributed file using the `create-distrib` command with the `-a` option. A file must already exist before it can be attached to a distributed file.

In the following example an existing file, `DISTCUST.SOUTH`, is attached to the distributed file `DISTCUST` as part number 4:

```
jsh DEMO ~ --> create-distrib -a DISTCUST 4 DISTCUST.SOUTH  
  
Part file 'DISTCUST.SOUTH', Part number 4 added
```

Note that we can also attach a file using a full explicit filepath as in:

```
create-distrib -a c:\home\myaccount\DISTCUST 4 DISTCUST.SOUTH
```

This method of attaching a distributed file is preferred to ensure the proper part file is resolved through the partition algorithm. See [Part Files](#) for further details.

A part file can be detached from a distributed file using the `create-distrib` command with the `-d` option. The synonym `DELETE-DISTRIB` can also be used for this purpose. For example, to detach the `DISTCUST.SOUTH` part file:

```
jsh DEMO ~ --> create-distrib -d DISTCUST 4  
  
Part file 'DISTCUST.SOUTH', Part number 4 deleted
```

Partitioning Algorithm

Each distributed file uses a partition algorithm to determine in which part file a record belongs. The partition algorithm is specified by using the create-distrib command. All part files belonging to a distributed file use the same partition algorithm.

There are two methods for defining the partition algorithm, the system defined method and the user-defined method. The partition algorithm uses the record ID (or part of the record ID) to distribute the record to the appropriate part file.

System Partition Algorithm

When a distributed file is created it is automatically set to use the system partition algorithm. The create-distrib command is used to set the partition algorithm. When using the system partition method, jBASE assumes that the record ID will be in the form:

```
<PartNumber> <Delimiter> <RecordID>
```

Where:

PartNumber is an integer which determines the part file to which the record is written.

Delimiter can be any character except a system delimiter (AM, VM, SVM). The default delimiter is a dash (-).

RecordID is the actual item-ID of the record. In a 'hashed' file type, this determines the group to which the record is written.

The following example sets (or changes) the distributed file DISTREGION to use the system partition algorithm. A dash (-) will be used as the delimiter between the part number and the record ID:

```
create-distrib -pSYSTEM,- DISTREGION
```

User-defined Partition Algorithm

The user-defined method provides the greatest flexibility when the need to determine which part file a record belongs is not based on an integer value (although it could be used for that purpose as well). The user-defined method is implemented by writing a jBC subroutine which determines the part number, and then setting the distributed file to use the subroutine as the partition algorithm. As an example, a distributed file called DISTCUST has 5 part files and the first character of each ID determines which part file the record belongs.

We will use the following jBC subroutine as the partition algorithm:

```
SUBROUTINE DistCustSub (Reserved, Key, PartNo)

EQU Otherwise TO 1

* Extract the first character from the Key

FirstChar = Key[1,1]

* Now determine the PartNo based on FirstChar

BEGIN CASE

CASE FirstChar = "N"; * North

    PartNo = 1

CASE FirstChar = "E"; * East

    PartNo = 2

CASE FirstChar = "W"; * West

    PartNo = 3

CASE FirstChar = "S"; * South

    PartNo = 4

CASE Otherwise ; * Invalid condition; isolate for later review

    PartNo = 99

END CASE

*

RETURN
```

Compile and catalog the subroutine. Ensure that the subroutine is accessible via the JBCOBJECTLIST environment variable.

The subroutine is called each time a record is read from or written to the DISTCUST distributed file. The subroutine must support 3 arguments:

Argument	Description
Reserved	This parameter is reserved for future enhancements and should not be altered within the context of the subroutine.
Key	This is the record ID. It must be constructed in the application program prior to READING or WRITING the record from/to the distributed file. Do not alter this argument, use it only as a source.
PartNo	This must be assigned by the subroutine and must return a valid part number.

You will notice that the part numbers consist of 1, 2, 3, 4 and 99. This illustrates an important feature. It is not a requirement that the part numbers be sequential or continuous. This could be used to allow additional part files to be added to the distributed file collection without the necessity of renumbering.

Take special care when writing this subroutine to account for all possibilities. If for any reason the PartNo cannot be determined you will receive either a READ_ERROR or WRITE_ERROR at the point of failure. Here is one such example where there are 11 part files. The part number is determined based on the last character of the key, the last character is assumed to be numeric but, if it's not, it will be placed in the 11th part file:

```
SUBROUTINE distsub(reserved, key, partno)

lastchar = key[-1,1]

IF NUM(lastchar) THEN

    partno = lastchar

    IF partno = 0 THEN partno = 10

ELSE

    partno = 11

END

RETURN
```

Can you spot the error?

The fatal flaw is if the subroutine ever encounters an item-id of 'null'. A null item-id is considered numeric, hence partno will be set to 'null'. A better way to code this would be:

```
SUBROUTINE distsub(reserved, key, partno)

lastchar = key[-1,1]

IF lastchar >= 0 AND lastchar <= 9 THEN

    partno = lastchar

    IF partno = 0 THEN partno = 10

END ELSE

    partno = 11

END

RETURN
```

This subroutine takes the 'explicit' approach and does not make assumptions about what form the data will be in.

To set (or change) the distributed file to use the user-defined partition algorithm, use the create-distrib command. For example, to set the DISTCUST distributed file to use the DistCustSub subroutine:

```
create-distrib -pUSER,DistCustSub DISTCUST
```

When compared to the system partition algorithm, the user-defined partition method incurs a small performance penalty when calling the jBC subroutine. The exact cost of this is highly dependent on how easily the part number is resolved within the subroutine.

Distributed File Commands

CREATE-DISTRIB

The CREATE-DISTRIB command is used to:

- 1) Maintain the parts of a distributed file
- 2) Set or change the partition algorithm
- 3) List or verify the component part files and the partition algorithm

The CREATE-DISTRIB command accepts a variety of options which determines its function as the following table illustrates:

Function	Syntax	Alternate Syntax
Add a part file	CREATE-DISTRIB -a FileName PartNo PartFile	CREATE-DISTRIB FileName PartNo PartFile
Delete a part file	CREATE-DISTRIB -d FileName PartNo	DELETE-DISTRIB FileName PartNo
List all part files	CREATE-DISTRIB -l FileName	LIST-DISTRIB FileName
Verify the existence of the Part Files	CREATE-DISTRIB -v FileName	VERIFY-DISTRIB FileName
Set (or change) to the System Partition Algorithm	CREATE-DISTRIB - pSYSTEM{,Delim} FileName	
Set (or change) to a User-defined Partition Algorithm	CREATE-DISTRIB - pUSER,Subroutine FileName	

Syntax Elements

Element	Description
Filename	The name of the distributed file
PartNo	An integer from 1 through 254 inclusive which associates the Part file to the Distributed File
PartFile	The name of the part file
Delim	A single character used to separate the Part Number from the record ID
Subroutine	The name of the user defined subroutine

Other options are:

- f Force Mode
- V Verbose

Examples

```
CREATE-DISTRIB -a INVOICES 24 INVOICES.MAR1999
```

Attaches the file INVOICES.MAR1999 as the 24th part file to the INVOICES distributed file.

```
CREATE-DISTRIB INVOICES 24 INVOICES.MAR1999
```

Same as the previous example. Note that the -a is assumed in the absence of any options.

```
CREATE-DISTRIB -pSYSTEM, | DISTCUST
```

Sets (or changes) the partition method to the SYSTEM partition algorithm for the DISTCUST distributed file. A vertical bar "|" will be used as the delimiter to separate the part number from the record ID.

```
CREATE-DISTRIB -pUSER,DistSub DISTCUST
```

Sets (or changes) the DISTCUST distributed file to use the user-defined subroutine, DistSub, as the partition algorithm.

```
CREATE-DISTRIB -d MEDICAL.CLAIMS 149
```

Detaches (disassociates) the 149th part file from the MEDICAL.CLAIMS distributed file.

```
CREATE-DISTRIB -l CONVENTIONS
```

Lists the component part files of the CONVENTIONS distributed file. Also lists the partition algorithm.

```
CREATE-DISTRIB -v CONVENTIONS
```

Verifies the existence of the component part files belonging to the CONVENTIONS distributed file. Also confirms the partition method. If the distributed file uses the user-defined partition method this also verifies that the subroutine can be executed.

DELETE-DISTRIB

The DELETE-DISTRIB command detaches (de-references) a component part file from a distributed file.

Syntax

```
DELETE-DISTRIB FileName PartNumber
```

Syntax Elements

Element	Description
FileName	The name of the Distributed File
PartNumber	An integer from 1 through 254 inclusive which was used to associate the Part File to the Distributed File

Notes

If the user-defined partition method is used, you should ensure that the subroutine used for the partition algorithm does access the de-referenced file.

If the system partition method is used, you should ensure that no keys are created which can read from or write to the de-referenced file.

Example

```
DELETE-DISTRIB INVENTORY 42
```

Detaches (de-references) the 42nd part file from the distributed file INVENTORY.

LIST-DISTRIB

The LIST-DISTRIB command displays all partition information pertaining to a distributed file.

Syntax

```
LIST-DISTRIB FileName
```

Syntax Elements

FileName is the name of a Distributed File.

Notes

The VERIFY-DISTRIB command is much more useful as this not only displays the same information as LIST-DISTRIB, it also verifies the existence of the component part files. If the distributed file uses the user-defined partition method, VERIFY-DISTRIB also verifies that the subroutine is executable.

Example

```
LIST-DISTRIB INVENTORY
```

VERIFY-DISTRIB

The VERIFY-DISTRIB command verifies the existence of the component part files of a distributed file. If the distributed file uses the user-defined partition method, VERIFY-DISTRIB also verifies that the subroutine is executable.

Syntax

```
VERIFY-DISTRIB FileName
```

Syntax Elements

FileName is the name of a Distributed File.

Example

```
VERIFY-DISTRIB INVENTORY
```

Distributed File Considerations

Although jBASE does not restrict you from directly populating part files, records should always be written through the distributed file stub. Be aware that if a record is placed in the wrong part file, and that record is subsequently handled through the partition algorithm, it will be placed in the part file according to the partition algorithms own relentless logic. This will result in the same record appearing in two part files.

Once part files are populated, changing the logic of the partition algorithm (or changing the partition method), could have disastrous results. If it is necessary to do this you must pass each record through the new partition algorithm so that it is placed in the proper part file. You must also remember to delete each record from its original location.

A distributed files is opened in the usual way. For example, the following statement opens a distributed file called DISTCUST:

```
OPEN "DISTCUST" TO DISTCUST_FILE ELSE ABORT 201, "DISTCUST"
```

By default, when a distributed file is opened, all component part files are opened at the same time. You can defer the opening of all part files by setting the JEDI_DISTRIB_DEFOPEN environment variable.

On versions of jBASE prior to 3.3.9, if a record ID resolved to a partition (part file) that did not exist, the process would be trapped to the jBASE debugger with an "Error 22" error message. This behavior has been changed (see patch number PN3_30268) such that a READ from a non-existent partition will take the ELSE clause and a WRITE will be trapped with an 'Error 22' unless the WRITE is supplied with the ON ERROR clause.

If you delete a part file then you must also DELETE-DISTRIB to remove the reference from the distributed file stub. You must also modify any user-defined partitioning algorithm. This is detailed in the distributed file example.

Distributed files support secondary indexes and triggers at both the distributed file level and the part file level.

Distributed Files Example

In this comprehensive example, we create a distributed file called DISTCUST using a user defined partition algorithm and attaching five part files.

Create the distributed file stub. This is the file to which all part files will be attached:

```
jsh DEMO ~ -->CREATE-FILE DISTCUST TYPE=DISTRIB
[ 417 ] File DISTCUST]D created , type = J4
[ 417 ] File DISTCUST created , type = DISTRIB
```

Define the partition algorithm. If the distributed file uses the default system partition method, this step would not be necessary unless you wanted to change the delimiter separating the part number from the record ID. For this example we will use the user-defined method by assigning the subroutine DistCustSub as the partition algorithm:

```
jsh DEMO ~ --> create-distrib -pUSER,DistCustSub DISTCUST
```

Create the five files to be attached as part files. If the files already exist then this step can be omitted:

```
jsh DEMO ~ -->CREATE-FILE DISTCUST.NORTH 1 101
[ 417 ] File DISTCUST.NORTH]D created , type = J4
[ 417 ] File DISTCUST.NORTH created , type = J4
```

```
jsh DEMO ~ -->CREATE-FILE DISTCUST.EAST 1 101
[ 417 ] File DISTCUST.EAST]D created , type = J4
[ 417 ] File DISTCUST.EAST created , type = J4
```

```
jsh DEMO ~ -->CREATE-FILE DISTCUST.WEST 1 101
[ 417 ] File DISTCUST.WEST]D created , type = J4
[ 417 ] File DISTCUST.WEST created , type = J4
```

```
jsh DEMO ~ -->CREATE-FILE DISTCUST.SOUTH 1 101
[ 417 ] File DISTCUST.SOUTH]D created , type = J4
[ 417 ] File DISTCUST.SOUTH created , type = J4
```

```
jsh DEMO ~ -->CREATE-FILE DISTCUST.ERRORS 1 41
[ 417 ] File DISTCUST.ERRORS]D created , type = J4
[ 417 ] File DISTCUST.ERRORS created , type = J4
```

Attach the five files to the distributed file:

```
jsh DEMO ~ -->create-distrib -a DISTCUST 1 DISTCUST.NORTH
Part file 'DISTCUST.NORTH', Part number 1 added
```

```
jsh DEMO ~ -->create-distrib -a DISTCUST 2 DISTCUST.EAST
Part file 'DISTCUST.EAST', Part number 2 added
```

```

jsh DEMO ~ -->create-distrib -a DISTCUST 3 DISTCUST.WEST
Part file 'DISTCUST.WEST', Part number 3 added

jsh DEMO ~ -->create-distrib -a DISTCUST 4 DISTCUST.SOUTH
Part file 'DISTCUST.SOUTH', Part number 4 added

jsh DEMO ~ -->create-distrib -a DISTCUST 99 DISTCUST.ERRORS
Part file 'DISTCUST.ERRORS', Part number 99 added

```

Now, let's list the part files we have just added.

```

jsh DEMO ~ -->LIST-DISTRIB DISTCUST -or- create-distrib -l
DISTCUST
Partitioning Algorithm is USER Subroutine 'DistCustSub'
Part file 'DISTCUST.NORTH', part number 1
Part file 'DISTCUST.EAST', part number 2
Part file 'DISTCUST.WEST', part number 3
Part file 'DISTCUST.SOUTH', part number 4
Part file 'DISTCUST.ERRORS', part number 99

```

The distributed file system is now complete. At this point, the DistCustSub subroutine is called each time a record is written to or read from the DISTCUST file.

Let's say, for example, that we find that the DISTCUST.ERRORS part file is no longer needed.

The actions we must take to remove this file from the distributed file are:

Detach the part file from the distributed file:

```

DELETE-DISTRIB DISTCUST 99
-or-
create-distrib -d DISTCUST 99

```

Modify the user-defined partition subroutine DistCustSub by removing the lines which allocate records to part number 99. Recompile and catalog.

Optional: Delete the DISTCUST.ERRORS file.

```

DELETE-FILE DISTCUST.ERRORS

```

JBASE TRIGGERS

Overview

The mechanism provided to define the action that takes place when a database trigger event occurs is a jBC subroutine. The name of the subroutine is specified in the create-trigger command. A different subroutine can be defined for each of the nine database trigger events, however it is usually more convenient to use one subroutine for each file that has a trigger defined, distinguishing between the different events in the subroutine.

The subroutine can be used to define ancillary updates that need to occur as a result of the primary update. The seven parameters passed to the subroutine allow interrogation and (where applicable) manipulation of the record being updated.

Subroutine Parameter Description

filevar

The file variable associated with the update. For example, you can do:

```
WRITE var ON filevar,"newkey"
```

however you must then be very careful of calling this subroutine recursively.

Event

One of the TRIGGER_TYPE_XXX values to show which of the 9 events is currently about to take place. ['TRIGGER_TYPE_XXX' values are defined in \$JBCRELEASEDIR/include/JBC.h (Unix) and %JBCRELEASEDIR%\include\JBC.h (Windows).

Type	Event
TRIGGER_TYPE_PREWRITE -	before a WRITE occurred
TRIGGER_TYPE_POSTWRITE -	after a WRITE occurred
TRIGGER_TYPE_PREDELETE -	before a DELETE occurred
TRIGGER_TYPE_POSTDELETE -	after a DELETE occurred
TRIGGER_TYPE_PRECLEAR -	before a CLEARFILE occurred
TRIGGER_TYPE_POSTCLEAR -	after a CLEARFILE occurred
TRIGGER_TYPE_PREREAD -	before a READ occurred
TRIGGER_TYPE_POSTREAD -	after a READ occurred
TRIGGER_TYPE_POSTOPEN -	after an OPEN occurred

prerc

The current return code (i.e. status) of the action.

For all the TRIGGER_TYPE_PRExx events, it will be 0.

For all the TRIGGER_TYPE_POSTxx events, it will show the current status of the action, with 0 meaning that the action was performed successfully and any other value showing the update failed. For example, if a WRITE fails because the lock table is full, the value in prerc is 1.

flags

Flags to show whether a WRITE or WRITEV was requested. Currently not implemented.

RecordKey

The record key (or item-id) of the WRITE or DELETE being performed.

For CLEARFILE, this is set to null.

Record

For the WRITE actions, this is the record currently being updated. For the DELETE or CLEARFILE actions, this is set to null. This variable can be modified within the Subroutine if required. However, the modification will be discarded unless the create-trigger command was executed with the -a option.

userrc

This variable can be set to a non-zero value for the TRIGGER_TYPE_PRExxx actions in order to abort the current action. However, unless the -t option was used with the create-trigger command, it will be meaningless.

There are two options to setting this value:

Any negative value will cause the action to be terminated. However, nothing will be flagged to the application, and it will appear to all intents and purposes that the action performed.

Any positive value is taken to be the return code for the action.

For example, when a WRITE completes it will normally give a return code of 0.

If this variable is then set to say 13 (which is the Unix error number for "Permission denied") then the application will fall into the jBASE debugger with error code 13.

NOTES

Processing carried out inside a trigger should be as economic as possible. They are not designed to allow copious application logic. Any files opened inside a trigger should be opened once and the file descriptor stored in named common such that files are not opened in every iteration. Care should be taken that logic inside a trigger will not result in overhead or in an infinite loop of trigger calls.

Assignment of Trigger Subroutine Arguments

The arguments of a trigger subroutine are generally assigned by the database management system at the time the subroutine is invoked, but there are exceptions. The subroutine can in turn assign or reassign argument values if the trigger was created with the -a option.

The table below summarizes the state of each argument at the time the subroutine is invoked, according to each trigger type. Note that there are three cases where record is null even though the record key is assigned, i.e., pre- and post-delete and pre-read. This is so for the read event because there is no need to read a record before reading a record, and in the case of the delete events, because the attempt to delete a non-existent record warrants no further action.

If an application requires a record to be verified prior to deleting it, then that operation that should be performed at a higher level.

Trigger Type	filevar*	event	prerc	flags	recordkey	record	userrc
Pre-Write	YES	YES	YES	N/A	YES	YES	UD
Post-Write	YES*	YES	YES	N/A	YES	YES	UD
Pre-Delete	YES*	YES	YES	N/A	YES	NULL	UD
Post-Delete	YES*	YES	YES	N/A	YES	NULL	UD
Pre-Clear	YES*	YES	YES	N/A	NULL	NULL	UD
Post-Clear	YES*	YES	YES	N/A	NULL	NULL	UD

Pre-Read	YES*	YES	YES	N/A	YES	NULL	UD
Post-Read	YES*	YES	YES	N/A	YES	YES	UD
Post-Open	YES*	YES	YES	N/A	NULL	NULL	UD

YES means that the variable is assigned in this trigger. UD means that it is user definable, N/A means that the variable is not used. Null means that the variable is assigned a null value.

Note that filevar is not the name of the file, but rather the system-level file unit, ie the 'OPEN' file descriptor. It can be treated as such for file operations within the subroutine, but cannot be treated as a typical variable, e.g., it cannot be used with a PRINT or CRT statement.

CREATE-TRIGGER

The CREATE-TRIGGER command is used to specify the database events for which the trigger subroutine is called.

COMMAND SYNTAX

```
CREATE-TRIGGER -Options FileName {triggername|*} subroutine
```

SYNTAX ELEMENTS

FileName can reference either a jBASE hashed file or a directory.

triggername must be * or one of the nine database events: POSTOPEN, PREREAD, POSTREAD, PREWRITE, POSTWRITE, PREDELETE, POSTDELETE, PRECLEAR, POSTCLEAR. If * is specified then the trigger subroutine will be called for each of the nine database events.

subroutine is the name of a jBC subroutine. Also see Trigger API.

The valid options for CREATE-TRIGGER are:

Option Name Explanation

- a or (A) amend flag subroutine can amend the record
- d or (D) debug flag subroutine can be debugged
- t or (T) terminate flag subroutine terminates update

-o or (O) overwrite flag overwrite any existing definitions

NOTES

CREATE-TRIGGER can be run multiple times for the same file. If a trigger has already been defined for the specified event then the overwrite flag must be used to effect the change.

EXAMPLES

```
CREATE-TRIGGER BP POSTOPEN SUBBOPEN
```

The subroutine SUBBOPEN will be called immediately after the BP file is successfully opened by any jBASE process.

```
CREATE-TRIGGER -o PAYROLL * SUBBP
```

The subroutine SUBBP will be called for every database event to the PAYROLL file. Existing trigger definitions will be overwritten.

Comment Sheet

Please give page number and description for any errors found:

Page	Error

Please use the box below to describe any material you think is missing; describe any material which is not easily understood; enter any suggestions for improvement; provide any specific examples of how you use your system which you think would be useful to readers of this manual. Continue on a separate sheet if necessary.

Copy and paste this page to a word document and include your name address and telephone number. Email to documentation@temenos.com