jBASE 5 Overview

CONTENTS

DOCUMENTATION CONVENTIONS 1

JBASE RELEASE 5 OVERVIEW 2

NEW FEATURES 3

Checkpointing 3

Warmstart Recovery 3

Resilient Files 3

Online Backup 4

Resizing files 4

Eliminated need for licensing daemons 4

New APIs 4

64 Bit Support 5

Native SQL Support 5

Native JDBC Driver 5

DOCUMENTATION 6

MIGRATION OVERVIEW 6

DOCUMENTATION CONVENTIONS

This manual uses the following conventions:

Convention	Usage
BOLD	In syntax, bold indicates commands, function names, and options. In text, bold indicates keys to press, function names, menu selections, and MS-DOS commands.
UPPERCASE	In syntax, uppercase indicates JBASE commands, keywords, and options; BASIC statements and functions; and SQL statements and keywords. In text, uppercase also indicates JBASE identifiers such as filenames, account names, schema names, and Windows NT filenames and pathnames.
UPPERCASE ITALIC	In syntax, italic indicates information that you supply. In text, italic also indicates UNIX commands and options, filenames, and pathnames.
`Courier`	Courier indicates examples of source code and system output.
`Courier Bold`	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 RELEASE 5 OVERVIEW

The release of jBASE 5 undoubtedly marks a major new software development for jBASE Software. This release builds on the successful foundations laid down by release 4 that went before and delivers a new level of resilience through the introduction of the jBASE Dataguard suite. This collection of technologies allows jBASE to be truly non-stop as a technology platform.

This release of jBASE has been re-engineered to be a true 64 bit application and only 64 bit releases are supported for production use. Much of the core of jBASE remains relatively untouched to ensure stability.

The major new features of the jBASE 5 releases are as follows:

- Checkpointing
- Warmstart Recovery
- Resilient Files
- Online Backup
- Resizing files
- Eliminated need for licensing daemons
- New APIs
- 64 bit support
- Native SQL Support
- Native JDBC Driver

NEW FEATURES

Checkpointing

JBASE 5 investment has been predominately in the area of transaction journaling which has been substantially engineered to provide an ever greater level of flexibility and robustness. A number of new structures have been introduced to allow more precision in recovery. One of the major enhancements has been the addition of a concept of Checkpointing.

Periodically, at predefined intervals, Checkpointing pauses new transaction activity to record a point in time when the database is in a known state. To record this instant in time, a checkpoint record is written to the transaction together with a timestamp.

Developments such as warmstart recovery and online backup would not have been possible to implement if it weren't for checkpoints.

Although the overhead of performing a checkpoint is minimal, the time interval between checkpoints is user configurable.

Warmstart Recovery

It has always been possible to recover a jBASE database by restoring the last backup and replaying the transaction journal but there was manual intervention required. Warmstart recovery can automate the recovery of a system that has been improperly shutdown with no user input.

This allows for automatic recovery from such things as power failures in a similar fashion to mainstream RDBMS products.

Resilient Files

In Badly sized hash files, a situation can arise where data is split across a number of different frames which all have to be read from and written to the disk when the data changes. If something went wrong during this period, there was a window of opportunity where the update could be interrupted and the structure of the file damaged. In jBASE release 5, a new type of file called a Resilient File has been introduced to eliminate this possibility.

In a resilient file, where data spans across frames on disk, it is built up in a separate area of the file and flushed to disk. Only once we know it is on disk, do we update a pointer to point to the new data structure rather than the old structure. The structure is such that all critical writes are exactly one disk block in size

which file systems guarantee to be atomic and so there is no potential for the structure of the file to be damaged in any way.

Due to the extra flushing of resilient files, there is a performance overhead when using these files.

Online Backup

Backup and restore commands have now been enhanced such that a database backup can be performed without the need to shutdown activity on the database server. The backup will be a complete copy of the database as at the point of the time when the backup process finished. This backup is guaranteed to be in a consistent state because of the checkpointing mechanism.

Resizing files

As well as being resilient, JR files also resize automatically to fit the data that is stored within them. This has been achieved by essentially divorcing the location of the data from the record key of the file. This means that the structure of the underlying data can expand without having to redistribute all the data as was the case with previous hash files.

Eliminated need for licensing daemons

Licensing is tracked in shared memory and allocated based on library use. There is no longer a need for separate license daemons to control the license counts.

New APIs

Application programming interfaces have been redesigned and streamlined with the introduction of a socket based protocol for communication between a client process and the server running jBASE. This eliminates the need for RMI and makes for faster communication between the two. As well as support for Java via Java OBjEX, there is also support for Dot Net applications through Dot Net OBjEX. This will allow applications written using the Dot Net framework to connect to jBASE on any supported platform.

64 Bit Support

Many operating system specific restrictions under 32 bit operating systems meant that jBASE was constantly having to work around them under the covers. With the move to a true 64 bit architecture, these restrictions no longer apply.

Native SQL Support

The query processor in jBASE has been enhanced to be able to accept SQL commands as well as jQL commands.

Native JDBC Driver

JDBC support allows external applications written in Java to retrieve data from jBASE via an industry standard interface.

DOCUMENTATION

Operationally, the core of jBASE works in exactly the same way as jBASE 4.1, which means that the documentation for this release carries over untouched from the previous release.

The following new features are documented in the new manual jBASE Dataguard;

- Checkpointing
- Warmstart Recovery
- Online Backup
- Resilient Files
- Resizing Files

There is no documentation for the new release being a 64 bit compliant release as this is transparent to the operator. There is an additional manual covering OBjEX and the Dot Net API is covered in a compiled help file.

MIGRATION OVERVIEW

Use the following steps to migrate from previous jBASE releases to jBASE 5:

1. Set up a test migration area on the server system

   Physically, this involves manually creating a directory (or directories) that will hold all the data files and program source code files

2. Copy existing system file, accounts and data and source files to test area

   Copy the files to the new directories.

   Note: BIN and LIB directories WILL NOT BE COMPATIBLE and can either be cleared or eliminated as they will be recreated in a later step.

3. Install and configure the jBASE 5 release

   Both jBASE 5 and previous releases will run on the same machine, however, there are a few points to be aware of:

   i. Both versions *may* share environment variables. This is only an issue if environment variables have been set up as e.g. a System or User variables on Windows. If a logon script is used on an operating system (e.g. through telnet) then the variables will be local to that instance and will not cause any problems.

   ii. The main variables that a user should be aware of are 'JBCRELEASEDIR' and 'JBCGLOBALDIR'. These two variables point to the instance of jBASE (whichever version)

   iii. The next variable to note is the 'PATH' variable. This variable is used to locate the executable programs (or jBASE commands). The order of the BIN directories in the 'PATH' variable will dictate the order that the system will search for the commands in

   iv. The other two variables typical in a MV database which would need to be set would be JEDIFILENAME_MD and JEDIFILENAME_SYSTEM.

   v. Under certain circumstances, previous jBASE releases will have created a 'libjbase.dll' file in the 'Windows\system32' directory. If this is the case then the file should be moved to 'lib' to a more relevant 'lib' directory.

4. Start any required processes

   If jDLS/jTELNET are to be used – start them using the required options.

   Note : If upgrading from jBASE 3.4 or earlier versions of jBASE 4, you will want to use the jDLS locking service instead of the jRLA service.

5. Configure the server user id and set up the profile using 'iju' utility

   The 'iju' utility is a command line program that creates a login script for users. Prior to running the 'iju' program, some users need to be set up on the operating system. Running the 'iju' facility will prompt for information pertaining to the user in question (e.g. user name, location of MD and SYSTEM files etc.), this information is then used to create a login script for the user in question. In the case of Windows, this will be a 'Remote.cmd' file. This should be placed into the relevant directory for the user. The Windows OS will automatically run this script when the user connects to via Telnet. In the case of UNIX, the script forms the basis of the users '.profile' script.

6. Execute 'jsh'

   Depending on the configuration of the user, connect to the jBASE machine. In some cases this will be via telnet and run the command 'jsh'. If the system has been set up with OS Global\System environment variables in Windows (not a telnet user) then run the command 'jSH' from a DOS command line

7. Logto each account and do step 8-15

   Issuing the command 'LOGTO <*accountname*>' in the jSHELL will connect the user to the selected account. Any variables set up in the 'Setting' parameters during the set up of the accounts should now be set. To test this, use 'echo %*VARNAME*%' (Windows) or 'echo $*VARNAME*$' (UNIX)

8. Update the MD/VOC, using the 'updatemd' utility

   The 'updatemd' command copies information from a 'Master MD template' into the MD/VOC of the account. This information will be required by jBASE.

9. Optionally convert source and data for international mode using 'jutf8'.

   The command line program 'jutf8' is used to internationalize data. Before the program can be run, the account must have internationalisation enabled.

10. Delete all old object files, i.e. dollar records.

    Search through the directory structure created in step1 and manually delete old object files. These are typically files that start with a '$' or '!' character and may end with a '.obj' or '.o' extension. You will probably want to create the optional OBJECT sub files for the BP files so that the object will no longer need to be stored in the BP file.

    i.e. CREATE-FILE BP,OBJECT 29 would create the optional OBJECT sub file.

11. Be sure to clear or delete any existing 'bin' and 'lib' directories.

    Search through the directory structure created in step1 and manually delete any 'bin' and 'lib' directories. These directories will hold the programs and libraries that were used and will need to be re-built in the next steps

12. Parse source files for any new key words using the 'portbas' utility.

    The command line program 'portbas' is used on individual source files (one at a time) to ensure that the source code conforms to the jBASE 5 standards. The program automatically creates a back up copy of the original file and makes any changes necessary to the file.

13. Compile source files using BASIC command

    The command line program 'BASIC' is the first of a two-step process to convert source code into an executable/callable form. The 'BASIC' command checks the code to ensure that it is syntactically correct. To use the command from a jSHELL, navigate to the relevant directory and call the BASIC command supplying the name of the source code:

    E.g. BASIC <SourceFile> <program\sub name in the file>

    Alternatively, the star character can be used to signify all programs\subroutines in the source file:

    E.g. BASIC <sourceFile> *

14. Catalog source files using CATALOG command

    The command line program CATALOG is run after the BASIC command has been run successfully and is the second step of the process. The command will create the executable program that can be called, or in the case of subroutines, create a '.DLL' '.SO' file. The syntax for the CATALOG command is the same as that for the BASIC command.

15. Test Application