jBASE Overview

What's New in Release 4

© 2006, 2016 Zumasys, Inc.

Copyright

Copyright (c) 2006 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. TEMENOS UK Limited, makes no warranty of any kind with regard to the material contained in this manual, including but not limited to the implied warranties of merchantability and fitness for a particular purpose.

The information contained in this manual is subject to change without notice.

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

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

Documentation Conventions

This manual uses the following conventions:

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

⇒	A right arrow between menu options indicates you should choose each option in sequence. For example, “Choose **File ⇒Exit**” means you should choose **File** from the menu bar, and then choose **Exit** from the File pull-down menu.

Syntax definitions and examples are indented for ease in reading.

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

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

JBASE RELEASE 4.1 OVERVIEW

The release of jBASE 4.1 undoubtedly marks a major new software development for jBASE Software. This release introduces a number of new concepts and features to the jBASE product model as well as several minor, although no less important, enhancements.

An incredible amount of time and effort has been put into remodelling the jBASE core system and the complementary surrounding components to produce a truly world beating, leading edge platform and tool set, capable of catapulting legacy as well as mainstream applications to the fore front of tomorrows computing solutions.

The major new features of the jBASE 4.1 releases are as follows:

- Threads
- Internationalization
- New Query Engine and Conversion Processor
- New Administration and Maintenance Tool
- New Remote File Handling
- New Platform Support
- Additional Compiler support
- New CALLJ and CALLdotNET interface
- New JavaOBjEX functionality
- New jQL functionality
- New jPlus files
- New jBASE BASIC Functions

New Features

Threads

JBASE 4.1 has been substantially recoded to provide for multiple thread capability and functionality. This capability is provided in two models. The Execute/Perform Thread Model and the Advanced Thread Model.

A thread is one part of a program that can execute independently from other parts of the program. The basic technical difference between threads and processes is that processes have their own memory space, whereas threads are in the same memory space. Sometimes called “sub-processes”, threads can operate in the same data and text space as normal processes. As a result, there are substantial gains to be made in performance by utilizing threads. It is infinitely faster to create a new thread. Processes are expensive to create.

Many environments like Java or COM and some Web Services work much better when using a new thread rather than having to fork a new process. These environments are designed to work efficiently with threads.

Multithreading allows multiple streams of execution to take place concurrently within the same program, each stream processing a different transaction. Therefore, multithreaded programs are of most value when run in multitasking or multiprocessing environments, which allow multiple operations to take, place concurrently.

Execute/ Perform Thread Model

The Execute/Perform Thread Model means that when a process ‘executes’ a program using either the PERFORM or EXECUTE statements in jBASE BASIC, in place of ‘fork’ing a new process the program to be executed can be started in a thread of the same process. The initiating program then just waits for the new program to complete. The cost of a process ‘fork’, which can be relatively substantial due to process duplication by the underlying operating system, can be saved and the system resources preserved. This has the benefit of reducing the overall system requirements to run the application, while at the same time massively improving the execution performance of the application.

Advanced Thread Model

The Advanced Thread Model provides the capability to multi thread complete applications and/or their subcomponents. In the Advanced Thread Model, several threads can be simultaneously executing the same or different components of an application. As is possible with shared libraries and jBASE today, there is only the requirement to have one copy of the application in memory at any one time. However, with the advanced thread model, only the private data for the thread is duplicated rather than the full process data space. This model is most beneficial for the OS390 and AS400 platforms as the model drastically reduces the requirement for multiple processes address spaces. However, the model can also be deployed to reduce system requirements on both the UNIX and the Windows platforms.

jBASE Thread Option and Functions

Start Asynchronous threads in the same process using the –Jt option in the PERFORM/EXECUTE statement.

```
EXECUTE "MyProgram -Jt "
```

jBASE 4.1 also introduces three functions for jBASE BASIC programs, which can be used to start asynchronous threads in the same process.

JBASECOREDUMP

Use as a diagnostic tool for applications which allows a snapshot of the application to be dumped to an external file for later analysis.

JBASETHREADCreate

Use the JBASETHREADCreate command to start a new thread.

JBASETHREADStatus

The JBASETHREADStatus command shows the status of all running threads.

Internationalization

The jBASE 4.1 release incorporates modifications to the underlying jBASE library functions and components such as to provide the tools and mechanisms whereby applications can be internationalized and localized for global communities. The internationalization functionality provides applications with the capability for correct handling of locale dependent collation sequencing, along with processing of unique character properties, code page data import/export and terminal/printer data input and output. The jBASE library functions when used in International Mode process internally using character rather than byte orientated values and properties such that applications can be easily coded or converted by minimum change for the international market.

Query Engine and Conversion Processor

JBASE 4.1 incorporates a completely new Query engine (jQL) along with a complementary conversion processor. Several additional syntax and keywords provided by other legacy systems have been provided for in the new Query engine, thus providing a superset of legacy reporting tools. The new Query engine has been coded to provide an extremely high performance-reporting tool, to the effect that appending additional selection criteria to further refine the query has a minimal impact on the query processing performance. The jBASE conversion processor has been recoded to provide extensive support for all conversion formats as well providing additional support for internationalized date, time, currency and number formats.

Remote File Handling

JBASE 4.1 incorporates a new Remote File Service capability. The new jRFS has been coded such that the service can take advantage of standard mainstream JINI services. The required client and server configuration to properly locate remote files has now been substantially reduced. In addition, the jBASE Remote File Service can be configured to handle backup system takeover in the event of a remote server system failure. The new jRFS also incorporates the capability to handle jQL queries correctly on remote files such that the query is executed on the remote server, hence optimizing the query process and minimizing network traffic.

Additional Platform Support

The jBASE 4.1 release provides platform support for IBM iSeries (AS400) and zSeries (OS390) systems. The jBASE installations on these systems provide native system support by way of a Unix System Services (USS) implementation for the zSeries (OS390) and a native ILE implementation for the iSeries (AS400). These two native implementations have been developed with extensive corporation from the IBM development laboratories.

Additional Compiler Support

JBASE 4.1 introduces support for new or complementary compilers. With jBASE 4.1, the latest GNU Compiler is now supported and can be used in place of the native system compiler. In addition, support for the .NET framework compiler has been added to the Windows installations of jBASE 4.1. Refer to the samples directory provided with the jBASE 4.1 releases for examples of how to use the GNU compiler.

CALLj and CALLdotNETinterface

JBASE 4.1 introduces additional support Java and .NET technology via the CALLJ (Call Java) and CALLdotNET (Call into .NET) interface statements. You can use these statements from jBASE BASIC to excellent effect, such that the multitude of packaged solutions available on the market today for Java and/or .NET framework can be seamlessly integrated with jBASE BASIC programs. Refer to the samples directory provided with the jBASE 4.1 for examples of how to implement this functionality.

JavaOBjEX functional extensions

The jBASE 4.1 release introduces new functionality for JavaOBjEX, such that client programs can remotely communicate with a javaOBjEX server program and hence the application subroutines and data, without the requirement for jBASE to be loaded on the client system. This modification to javaOBjEX provides for an easily installable lightweight client installation. In addition javaOBjEX has been further extended to provide jQL statement objects for jQL execution and result retrieval similar to the jQL extensions for jBASE BASIC, along with the terminal IO object, which provides for control input and output events and an XML object, which can utilize the extended dictionary information to

generate XML documents from the jQL object result sets. Refer to the samples directory provided with jBASE 4.1 for examples of how to implement some of this functionality.

jQL functional extensions

JBASE 4.1 introduces new functions to jBASE BASIC to enable execution of queries and retrieval of the query result set within the jBASE BASIC program. These set of functions provide an extremely powerful tool for manipulation from within jBASE BASIC such that only data fields specified by the query need be retrieved with optional formatting ready for further processing by the application. The query can be efficiently compiled and executed, and then the results retrieved directly, thus obviating the need to execute/perform a SELECT command, which generates the select list, which in turn must then be processed by a readnext and a read statement in order to access the data field in the record.

jPLUS files

JBASE 4.1 introduces jPLUS files which provide large file support on the majority of Unix and Windows platforms, such that Hash files can extend beyond the normal 2 GB operating system limit. In addition, the jPLUS files provide configurable levels of data flushing to ensure file integrity stays intact in the case of system failure. Enabling support for large files is for the underlying Operating System file system when creating the OS files system. System administrators usually create the OS file systems and the options required to enable the use of large files differs between platforms. Refer to System Administration Guide re file system creation for your specific platform.

New jBASE BASIC Functions and Statements

The following functions and statements, which are new to the jBASE BASIC language, are now defined as keywords. Any use of these keywords in user source code can be safely converted to a capitalized form using the jBASE 'PortBas' portation tool.

BYTELEN	CALLJ	CALLdotNET
CHANGETIMESTAMP	DEFCE	JQLCOMPILE
JQLEXECUTE	JQLFETCH	JQLGETPROPERTY
JQLSETPROPERTY	LATIN1	LENDP
LOCALDATE	LOCALTIME	MAKETIMESTAMP
TIMEDIFF	TIMESTAMP	UNIQUEKEY
UTF8 XMLTOXML	DYNTOXML READXML	XMLTODYN WRITEXML

New Formatting syntax for FMT, ICONV and OCONV

JBASE 4.1 introduces additional formatting syntax for the FMT, ICONV and OCONV functions. This formatting extension provides support for field width formatting. For example, the following command would cause the supplied string to be formatted in a left justified text field of 10 characters. Any characters exceeding beyond the 10-character limit will be "wrapped" using text characters to separate each 10-character string. Each field of 10 characters can then be extracted from the dynamic array using the REMOVE statement.

MyString = "Greg the slow fat oaf tripped over the very unlucky dog"

MyFormattedString = MyString "10#L"

Additional Date and Time formatting have also been introduced with jBASE 4.1 to provide extensions for internationalization.

The “DG” date format option provides a useful global date format as YYYYMMDD, where YYYY is the year, MM is the month and DD is the day.

A new format type “W” has been provided to signify World Date or World Time for use with timestamp values. For example:

TimeStamp "W{Dx}{Tx}"

Where:

W A new conversion code for World Date/Time, used with Timestamp
D Date
T Time
x Format option: S = Short, M = Medium, L = Long, F = Full

The format of the short, medium, long and full is exampled below:

"WDS" or "WTS" SHORT is completely numeric. 12/13/52 or 3:30
"WDM" MEDIUM is longer. Jan 12, 1952

"WDL" or "WTL" LONG is longer. January 12, 1952 or 3:30:32pm

"WDF" or "WTF" FULL is completely specified

Tuesday, April 12, 1952 AD

3:30:42pm PST

The normal TIMEDATE function now returns the date format using the configured locale when in International Mode. The time portion of the string is always in the format hh:mm:ss, however the date format can vary according to locale and will provide the short version similar to the WDS format.

jBASE BASIC Statements for use with jQL

The following statements have been created to enable jBASE BASIC programmers to deal directly with jQL statements, thereby eliminating the need to parse the output of commands such as EXECUTE. Equivalents of these statements will also be made available in Java OBjEX.

JQLCOMPILE

JQLCOMPILE compiles a jQL statement.

JQLEXECUTE

JQLEXECUTE starts executing a compiled jQL statement.

JQLFETCH

JQLFETCH fetches the next result in a compiled jQL statement.

JQLGETPROPERTY

Receives the requested property value from the system or “” if the property is not set

JQLPUTPROPERTY

JQLPUTPROPERTY sets a property in a compiled jQL statement.

JQLCOMPILE

JQLCOMPILE compiles a jQL statement.

JQLEXECUTE

JQLEXECUTE starts executing a compiled jQL statement.

XML Functions

JBASE is incorporating new XML capabilities built into jBASE BASIC based on the Xalan and Xerces libraries.

DYNTOXML

Convert the array to XML using the optimal xsl to transform

XMLTODYN

Converts the XML to a dynamic array using the optional XSL to transform

XMLTOXML

Transform the XML using the XSL

XML Statements

READXML

Read XML from a dynamic array using a style sheet from the DICT

Reads a record from a file using the style sheet held in DICT->@READXML to transform the data into xml format

WRITEXML

Writes a dynamic array in xml format using a style sheet from the DICT

Use WRITEXML to write an XML record to a hash file

Transforms the XML into a dynamic array before being written to the file

The transform takes place using the style sheet in DICT->@WRITEXML

DESIGN CHANGES

Single Point of Failure

JBASE 4.1 removes the possibility of a single point of failure when operating the application. This is mainly by the removal of the reliance on the jPML, (Process Manager and Licensing) demon, this process was mandatory for Unix installations and hence posed a possible single point of failure, i.e., If the jPML demon failed for any reason then jBASE processes could hang forcing disconnection. With the jBASE 4.1 release the jPML demon is no longer required, hence removing this possible point of failure. In addition to the jPML demon, the background-processing demon (jBTP) is also no longer required.

Root/Administrator Privileges

JBASE 4.1 removes the requirement for some programs to execute with root/administrator privileges. This requirement was mainly associated with the jPML and jBTP demon processes with have now been dispensed with on jBASE 4.1.

Program Execution

JBASE 4.1 introduces a major change to the manner in which jBASE controls program execution. Program execution includes statements such as CHAIN, ENTER and PERFORM/EXECUTE.

Prior to jBASE release 4.1, each of these statements would involve the creation of a new operating system process. In the case of PERFORM/EXECUTE, the parent process would wait for the child process to terminate. In the case of CHAIN and ENTER, the parent process would terminate at the same time the child process.

This earlier model gave jBASE a real native feel to it, as all the programs were genuine executable programs. The reasons for change were two fold.

Performance: The overhead of creating new processes is a relatively high overhead, although this depended upon the application.

Thread safety: Modern environments need to have applications working in a thread-based model, java being a prime example.

The new model, provided by jBASE 4.1, now has the following properties:

A user is simply a data object: More than one user can exist in each program.

Compiled programs are now compiled into a shared object: A program is still compiled into an executable program such that if can also be run direct from the native command line or shell scripts.

To run a program an object is simply associated with a thread: The thread then runs the program using the object as its data instantiation.

Program Execution Comparison

Consider a user starting a jBASE program, “MAIN”, which does an EXECUTE “DIARY”. Prior to jBASE 4.1 the steps to run this program would be as follows:

a) Start an executable with the name MAIN (or MAIN.exe for Windows)
b) Use the operating system to create another process
c) The original MAIN program would wait for the child process to terminate.
d) The child process would execute program DIARY.
e) The child process (DIARY) would terminate.
f) The parent process (MAIN) would terminate.

JBASE 4.1 now does something radically different. While there appears to be more steps in the following example, this is purely for explanatory purposes. Overall, the overhead is substantially lower than previously describe for prior releases.

a) Load an executable with the name MAIN (or MAIN.exe for Windows)
b) Start a 'listening' thread for internal jBASE use.
c) Instantiate an object to describe the user
e) Start a thread and associate the object then execute program MAIN.
f) Load the shared object for DIARY and execute the shared library function.
g) Return from executing DIARY back to the code in MAIN.
h) Terminate the thread, which was associated with the MAIN program.
i) Deconstruct the object for the user.
j) Terminate the process.

Performance is improved because the execution of PERFORM "DIARY" only involves steps f) and g) which is very fast, in addition as the shared object for DIARY is cached, then f) is essentially only required to be done the first time around.

CHAIN and ENTER both use very similar mechanisms however the important point is that the same thread is used as an execution thread for all the programs that are PERFORM'ed, CHAIN'ed or ENTER'ed. All that is now required is control of the point at which the thread is currently executing.

In this new mechanism, as the user is simply an object, we can use any executing thread to run the program for that user. This is particularly important for environments like web servers and java processes, whereby thread pooling is required, such that different threads might be used for each execution of a program or subroutine.

The execution of subroutines has not changed from previous jBASE releases, in that the subroutines (shared objects) are still loaded from shared libraries and program execution is branched to the entry point of the subroutine.

jPML

Under jBASE 4.1 there is no requirement for the jPML (Process Manager and Licensing) demon process. The jPML process was responsible for synchronizing communication between executed processes. However, with jBASE 4.1, the PERFORM/EXECUTE statements no longer create or fork new processes and so the requirement for process communication via the jPMLWorkFile and synchronization via the jPML message queues is no longer required.

jbcconnect

The jBASE 4.1 release also does away with the ‘jbcconnect’ program, as it is no longer required. The reason for the ‘jbcconnect’ command, on prior releases, was to inform the jPML demon that this user connection was to be associated with a consistent port number and connection id, which was then set in the JBCCONNECT environment variable. This environment variable information was then used, in conjunction with the jPML demon, to ensure that all subsequent programs executed by the user connection were consistently allocated the same port number and environment.

jBTP

Under jBASE 4.1 there is no requirement for the jBASE Background Task Processing demon. This demon was required on previous releases to be able to start up programs in the background as different operating system users and as such required root privileges. Background processing programs such as PH-START and Z have been modified and no longer require the jBTP demon and will only be able to start background processes using the same operating system user id.

Locking and jRLA

The use of the jBASE record Locking Arbiter demon from a user perspective under jBASE 4.1 does not change. However internally the lock owner is now resolved as a combination of process id and object id to cater for multithreading processes. In previous releases, only the process id was used. The jRLA demon is not required to run as root/administrator user. Although still optional, the use of the jRLA demon is recommended for jBASE 4.1 as it provides an optimized parallel locking facility with performance far in excess of that provided by OS based file locks. The jRLA locking facility is now also available for the Windows platform. The jRLA configuration file is no longer required.

By default, earlier versions of jBASE would release all locks taken by a subroutine upon exiting the subroutine. On jBASE 4.1, the locks remain until explicitly released.

jspprint

Under jBASE 4.1 there is no change to the use of the jspprint demons. These demons are started in background using the SP-STATUS menu or the restart spooler commands.

CATALOG Programs and Subroutines

jBASE 4.1 changes the way main programs are executed. The CATALOG command creates main programs as executable programs but also as loadable-shared libraries duplicated as loadable-shared libraries similar to subroutines.

What this actually means is that main programs are created, using the CATALOG command, as both main executable programs as previously but also duplicated as loadable shared libraries similar to subroutines.

These main program shared libraries are stored in the bin directory along with the normal main executables. The reason for the shared library version of the main program is such that these programs can be executed in the same process space by a thread, similar to the mechanism for a subroutine. This execution behavior then fits with the perform thread model. The normal main executable is also generated so the program can still be started from the Unix/Windows command line or scripts/bat files, rather than from the jBASE jSHELL. The majority of jBASE program executables such as SELECT are also duplicated as shared library modules to integrate with the perform thread model. The format and usage of subroutines is unchanged however, it should be noted that if the main programs are modified and re cataloged that program changes will not be 'seen' by other processes that have previously executed those programs. This is because the main program is already 'attached' to the process as a shared library and hence the original copy preserved in memory.

SELECT LISTS

The jBASE 4.1 release with the new perform model introduces a subtle difference to previous releases with regards to the use of external SELECT LISTS. External select lists are used to pass lists of record keys between two programs. The jBASE 4.1 release will now only make select lists available to programs executed by the perform thread model. This change also effects the execution of the jSHELL when used in 'sh' mode. In this mode, the jSHELL will execute programs in new process address spaces and hence any external select list in force will not be available to the new program.

Program do_select.b;

```
CRT "Normal PERFORM, i.e. PERFORM 'check_selected' After a SELECT ."
PERFORM "SELECT ." CAPTURING Output
PERFORM "check_selected"
CLEARSELECT
CRT
CRT
CRT "External PERFORM, i.e. PERFORM @IM:'check_selected' After a SELECT ."
PERFORM "SELECT ." CAPTURING Output
PERFORM @IM:"kcheck_selected"
CLEARSELECT
Program check_selected.b
CRT "Selected Items=":@SELECTED
```

If you compile and catalog both these programs, and run do_select, you will get the following output under 4.1

```
Normal PERFORM, i.e. PERFORM 'check_selected' After a SELECT .
```

Selected Items=62
External PERFORM, i.e. PERFORM @IM:'kcheck_selected' After a SELECT .
Selected Items=0

Environment Variables

JBASE 4.1 has reduced the overall number of environment variables but has also introduced some additional variables for new functionality.

Installation/Configuration

JBASE 4.1 introduces a new installation procedure coded in Java to provide better control of the jBASE installation and initial configuration. In the unlikely event that you may wish to remove the jBASE 4.1 release, an uninstall option has also been provided. When installing jBASE 4.1 the user will be guided through the installation process by a number of prompt and response events. Installation on the desktop will also provide options to install and configure client systems as well as the option for custom installation of certain components.

User Source Code Changes

Some of the concepts introduce with the jBASE 4.1 release mean that some user programs may need to be modified to take account of these changes. The majority of these changes are automatically provided for by the jBASE 4.1 compiler, such that all that is required for existing programs to work with jBASE 4.1 is a recompile of the source code and a re catalog of the resulting object programs.

However if users have coded programs which interface to jBASE programs, i.e. the programs are called from jBASE BASIC programs by CALLC or a DEFC for a C function, then the programs will need to be modified to take account of the additional argument for the thread data pointer. This is also the case for C programs which call jBASE BASIC subroutines, such that a data pointer must first be obtained before calling the subroutine. Any user coded jEDI driver source will also be affected, as the jEDI API's have been modified to pass around the thread data pointer. In addition user coded jEDI drivers will need to be modified to ensure they can support multithreaded application. User make files may also need to be modified to cater for the new jBASE 'jcompile' command.

New jBASE Compiler Command

JBASE 4.1 introduces a new general-purpose compiler command that can be used for different sources and objects. The required compiler for the source file is invoked according to the source file extension, for instance '.g' will assume an 'antlr' source file and invoke the 'antlr' tool to convert the source file into a '.cpp' source file for use in a lexer program. These commands replace the 'jbc' command that was available with previous jBASE releases and the 'jBuildSLib' command, used for building shared libraries. Refer to the samples directory supplied with jBASE 4.1 and see the jcompile -H option for a description.

Note: that End Users who only use BASIC and CATALOG will be unaffected by this change

Changes for Calling C functions

User coded C functions called from jBASE BASIC programs need to be modified to either expect the additional Data Pointer argument or redefined to use the DEFCE statement, which specifies that this C function is an external C function and does not require the Data Pointer argument.

NOTE: that all the jBASE macros defined in the jsystem.h include file now have a reference to the data pointer argument. If using external C functions that are not using jBASE macros then the only argument types, which can be used with the DEFCE function declaration, are FLOAT, INT or STRING. Refer to the samples directory supplied with jBASE 4.1 for an example on how to implement this functionality.

Changes for calling Subroutines from C/C++

User coded C or C++ functions that call directly into jBASE BASIC subroutines will need to be modified to obtain a Data Pointer before calling the subroutine. Refer to the samples directory supplied with jBASE 4.1 for an example on how to implement this functionality.

Changes for jEDI drivers

User coded jEDI drivers will need to be modified to take account of the Data Pointer that is now passed to all functions, in addition all jEDI drivers should be recoded to ensure that they are thread safe and use thread safe functions. This is a change to the jEDI driver API introduced by the jBASE 4.1 release. Refer to the samples directory supplied with jBASE 4.1 for an example on how to implement jEDI drivers for jBASE 4.1.

Database Q Pointers

Enhanced jBASE 4.1 Q pointers provide the capability to access files in accounts on different databases. Attribute 4 of the MD/VOC Q pointer is now used to specify the database. This is optional such that if attribute 4 is null then the current database configuration (JEDIFILENAME_SYSTEM) will be assumed.

E.g. a DataBase THISDB contains an Account THISACC, which contains an MD file, which contains a Q pointer THATFILE.

e.g.

	THATQFILE
01	Q
02	THATACC
03	THATFILE
04	THATDB

When the THATQFILE is opened the database reference THATDB will be looked up in the jAdminServer profile/DATABASE entries to cross reference to a system file for the database THATDB, if the system file is found and opened then the account entry THATACC will be resolved from the system file. The path for the account used to prefix the target file THATFILE, is the filename and path that is then used for the open.

DataBase Entry THATDB in profiles/DATABASES/THATDB contains system path /thatdb/SYSTEM].

Account Entry THATACC in the system file /thatdb/SYSTEM]D contains home path of /thatdb/thatacc.

In short, Q pointer THATQFILE resolves to /thatdb/thatacc/THATFILE

Q pointers without a reference on attribute 4 are unaffected, however users should ensure that spurious or white space characters like tabs or spaces are not present on attribute 4 otherwise Q pointer resolution may fail.

F pointers are unaffected by this change. Note Q pointer to Q or F pointer resolution is not recommended and will only be enabled if the JEDIENABLEQ2Q environment variable is set as per previous jBASE releases.

Persistant Variables

Earlier versions of jBASE allowed the notion of persistant variables through the use of the V option when compiling. This option is no longer supported. A suggested workaround is to use COMMONs rather than persistant variables as illustrated in the examples below.

Subroutine using 'Persistant Variables' :

```
Command->
0001 SUBROUTINE dictsub
0002 INCLUDE qbasiccommonpick
0003  IF UNASSIGNED(openflag) OR openflag = '' OR openflag = @FALSE
THEN
0004 OPEN 'MYFILE' TO filevar THEN
0005 openflag = @TRUE
0006 END ELSE
0007 ABORT 201, 'MYFILE'
0008 END
0009 END
0010 .
0011 . rest of subroutine code
0012 .
0013 RETURN
```

Converted subroutine using named-COMMON :

```
0001 SUBROUTINE dictsub
0002 INCLUDE qbasiccommonpick
0003 COMMON /dictsub/ filevar, openflag
0004  IF UNASSIGNED(openflag) OR openflag = '' OR openflag = @FALSE
THEN
0005 OPEN 'MYFILE' TO filevar THEN
0006 openflag = @TRUE
0007 END ELSE
0008 ABORT 201, 'MYFILE'
0009 END
0010 END
0011 .
0012 . rest of subroutine code
0013 .
0014 RETURN
```

jQL Syntax Keywords

JBASE 4.1 provides support for the following syntax keywords for jQL. The keywords can be further expanded via the MD or VOC entries such that jQL syntax can be localized for specific languages where appropriate.

NOTE: that imported MD or VOC files must be updated before using the jQL syntax using the UpdateMD command.

!	&	*
+	-	%
/	<	=
>	<>	><
#	<=	=<
=>	>=	~
A	AFTER	ALL
AN	AND	ANY
ARE	AS	ASSOC
ASSOCIATION	ASSOC.WITH	ASSOCIATED
AVERAGE	AVG	BEFORE
BETWEEN	BREAK-ON	BREAK.ON
BREAK-SUP	BREAK.SUP	BSELECT
BY	BY-DSND	BY.DSND
BY-EXP	BY.EXP	BY-EXP-DSND
BY.EXP.DSND	CALC	CALCULATE
CAPTION	CNV	COL-FILLER
COL.HDG	COL.HDR	COL-HDR-SUPP
COL.HDR.SUPP	COL-SPACES	COL.SPACES
COL.SPCS	COL-SUPP	COL.SUP
CONV	COUNT	COUNT.SUP
COUNT.SUPP	COUNT-SUPP	DATA
DBL-SPACE	DBL-SPC	DBL.SPC
DEFAULT	DET-SUPP	DET.SUP
DICT	DISPLAY.LIKE	DISPLAYLIKE
DISPLAY.NAME	DISPLAYNAME	EACH
EDELETE	ENUM	ENUMUMERATE
EQ	EQUAL	ESEARCH
EVAL	EVERY	FILE
FIRST	FMT	FOOTER
FOOTING	FOR	FROM
GE	GRAND-TOTAL	GRAND.TOTAL
GT	HDR-SUPP	HDR.SUP
HEADER	HEADING	I-DUMP

ID-SUPP	ID.SUP	ID.SUPP
ID.ONLY	INQUIRING	IF
IN	ITEMS	JACSELECT
LE	LIKE	LIST
LIST-ITEM	LIST.ITEM	LIST-LABEL
LIST.LABEL	LPTR	LT
MARGIN	MATCH	MATCHING
MATCHES	MAX	MIN
MULTI.VALUE	MULTIVALUE	NE
NI.SUP	NI-SUPP	NO
NO-INDEX	NO.INDEX	NO.NULLS
NOPAGE	NO.PAGE	NO.SPLIT
NOT	OF	ONLY
OR	PAGE	PCT
PERCENT	PERCENTAGE	PG
REFORMAT	REQUIRE-INDEX	REQUIRE-SELECT
REQUIRE.INDEX	REQUIRE.SELECT	RETRIEVE
S-DUMP	SAID	SAMPLE
SAMPLED	SAVING	SELECT
SELECT-ONLY	SELECT.ONLY	SINGLE.VALUE
SINGLEVALUED	SORT	SORT-ITEM
SORT-LABEL	SORT.ITEM	SORT.LABEL
SPOKEN	SREFORMAT	SSELECT
ST-DUMP	STAT	SUBVALUE
SUM	SUPP	T-DUMP
T.DUMP	T-LOAD	T.LOAD
TAPE	THE	TO
TOTAL	TRANSPORT	UNIQUE
UNLIKE	USING	VERT
VERTICALLY	WHEN	WITH
WITHEACH	WITHIN	WITHOUT
WITHOUTEACH		

jQL I-TYPE Support

The jBASE 4.1 release provides I-TYPE support for the following functions and @variables.

Note: that for support of the I-TYPE function from jBASE BASIC programs you MUST configure the @FILENAME variable to use the file name of the file dictionary.

ABS	ACOS	ADDS	ADD_MONTHS
ALPHA	ANDS	ASCII	ASIN
ATAN	CATS	CHAR	CHARS
COL1	COL2	CONVERT	COS
COUNT	COUNTS	DATE	DCOUNT
DELETE	DIVS	DOWNCASE	EBCDIC
EQS	EXP	EXTRACT	FIELD
FIELDS	FMT	FMTS	GES
GTS	IFS	INDEX	INSERT
INT	ISNULL	ISNV	ISNVS
ISNULLS	LAST_DAY	LEN	LENGTH
LENS	LES	LN	LOWER
LTS	MATCHFIELD	MOD	MODS
NEG	NEGS	NES	NOT
NOTS	NUM	NUMS	ORS
PWR	QUOTE	RAISE	REPLACE
REUSE	RND	SEQ	SEQS
SIN	SOUNDEX	SPACE	SPACES
SPLICE	SQRT	STR	STRS
SUBS	SUBSTRINGS	SUM	SUMMATION
SYSTEM	TAN	TIME	TIMEDATE
TRIM	TRIMB	TRIMF	TRIMS
UPCASE	ICONV	ICONVS	OCONV
OCONVS	TRANSLATE	TRANS	

@AM	@FM	@VM
@SVM	@SM	@DAY
@MONTH	@YEAR	@SYS.BELL
@USER.NO	@WHO	@LOGNAME
@SYSTEM.RETURN.CODE	@USER.RETURN.CODE	@APPLICATION.ID
@FILENAME	@FILE.NAME	@TERM.TYPE
@UID	@DATA	@TTY
@SELECTED	@LPTRHIGH	@LEVEL
@PARASENTENCE	@PATH	@DATE
@TIME	@ID	@RECORD

jBASE Licensing

With jBASE release 4, the licensing model, and licensing components have been completely redesigned to give greater flexibility in deployment options and to offer further expandability for future licensing paradigms.

Since the introduction of the internet, applications have evolved from clients which retain a constant stateful connection to the server, to a world where connections are stateless and only maintained just as long as is absolutely necessary. Future licensing schemas will need to license products deployed in this way in a fair and equitable manner.

In the first instance, licensing in jBASE offers concurrent licensing in a similar fashion to previous releases. Over time, this licensing will be extended to offer more sophisticated licensing models which take into account how much use is made of jBASE itself.

License components

jLicence Server

The license server is the component into which license keys are entered. The license server validates these keys, stores them, and answers enquiries from jSLIM servers which have been configured to use the license server. There is typically one license server per installation of jBASE. There can be any number of jSLIM servers.

jLicence Client

The license client is a graphical tool for administering the license server. It is used to show the status of any licenses installed on a particular server and allows the entry of license keys. Everything that can be accomplished using the client can also be done through a command line console.

jSLIM Server

The jSLIM (System License Information Monitor) server needs to be started on every jBASE server. It provides an efficient mechanism for individual jBASE components to check their licensing status without needing to communicate with the license server every time. This allows the abstraction of individual jBASE servers away from the license server or in a simple installation, the jSLIM server and the license server can coexist on the same machine.

Administration

Installing the servers as services (jBASE 4.1.5 / Windows only)

On jBASE 4.1.5 and above, both the license server and the slim server are designed to be able to run as services on Windows platforms. The installation of these services should happen automatically at install time. Should the services need to be installed manually,

Installing jLicenseServer

```
jLicServer install
```

Installing jSlimServer

```
jSlimServer install
```

Removing jLicenseServer

```
jLicServer remove
```

Removing jSlimServer

```
jSlimServer remove
```

Starting the Servers

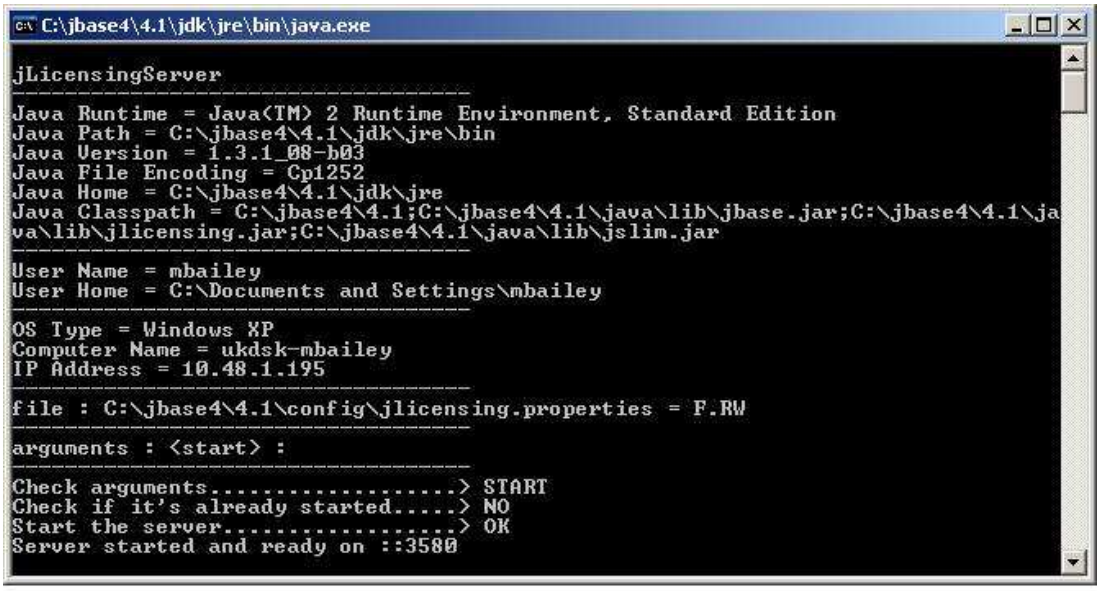
Scripts are provided to start up both the license server and the slim server. These can be found in the $JBCRELEASEDIR/bin folder on the machine where jBASE has been installed. There are four such scripts, two for Unix / Linux installations and two for Windows. The syntax for starting the jLicenseServer is as follows;

Starting jLicenseServer (Windows)

```
jLicServer [start | run] {-v}
```

Starting jLicenseServer (Unix / Linux)

```
jLicServer [start | run] {-v}
```



Using a start option will start the server in another process. If the run option is specified, the server will start in the same process. In either case, the -v option is for verbose output to the console. It is recommended that the license server is started prior to starting the jSLIM server.

Once the license server is started, the jSLIM server can be started in a similar fashion.

Starting jSlimServer (Windows)

```
jSlimServer [start | run] {-v}
```

Starting jLicenseServer (Unix / Linux)

```
jSlimServer [start | run] {-v}
```

Stopping the Servers

Scripts have been provided to stop both the license server and the slim server. The syntax for stopping the servers is similar to starting them. Once the jSLIM server has been stopped, jBASE processes on that machine will no longer be able to function and will report that no license is available.

Stopping jLicenseServer (Windows)

```
jLicServer stop
```

Stopping jLicenseServer (Unix / Linux)

```
jLicServer stop
```

Licence Server Console

A command line interface to an active license server can be activated using the following syntax.

Start the console (Windows)

```
jLicServer console
```

Start the console (Unix / Linux)

```
jLicServer console
```

The following commands are available in the console;

Command	Description
Help	List all console commands
InputKey	Input a license key
Status	Shows the status of licenses installed
View	Show details of licenses in use
Exit	Leave the console

Starting the Client

The license client can be started on a graphical console (will not work over telnet) using the following command;

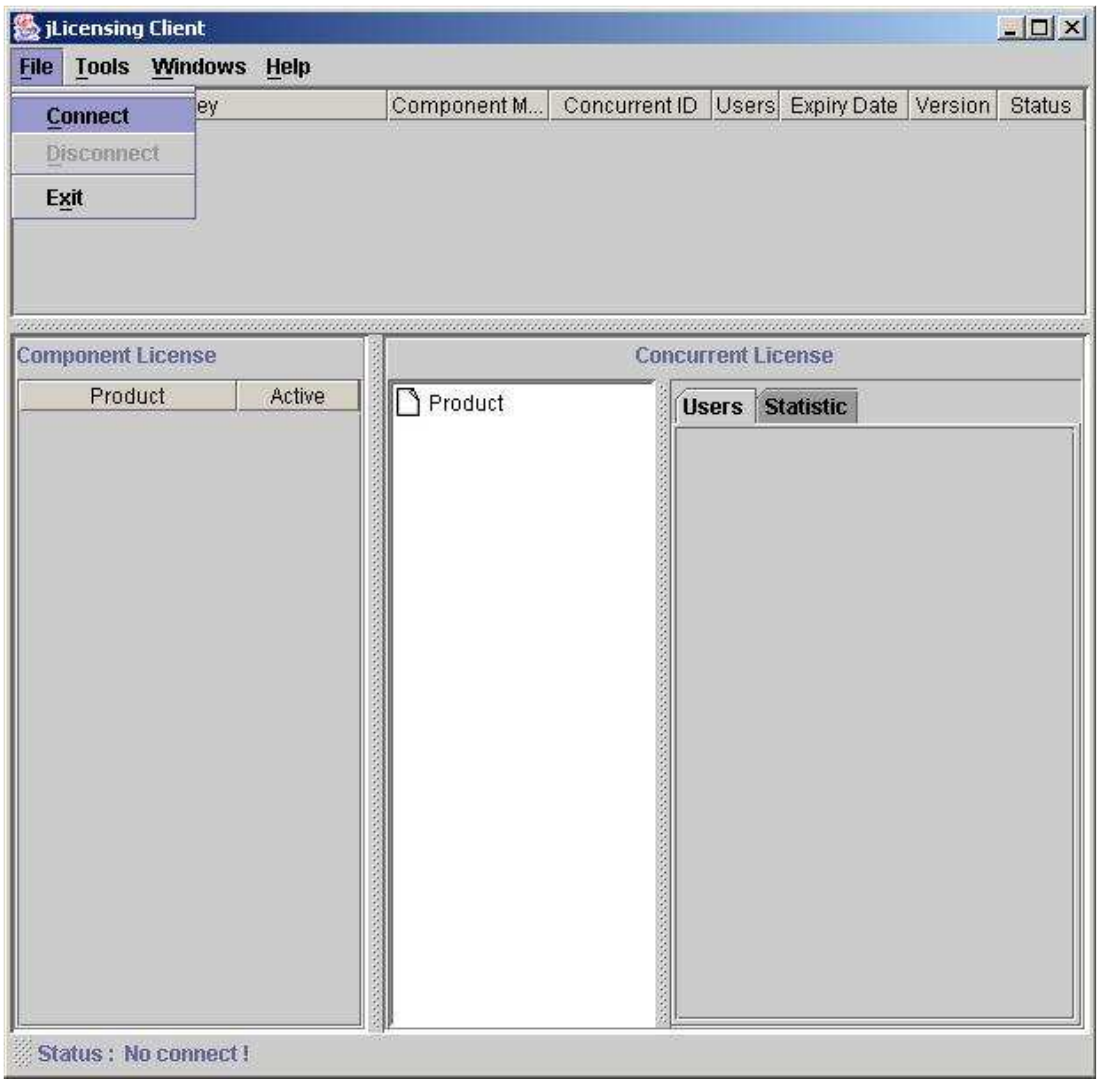
Start the client (Windows)

jLicClient

Start the console (Unix / Linux)

jLicClient

When the graphical client is started, it is not connected to a license server by default, even if the client is started on the same machine as the license server. In order to administer a license server, one must first connect to it using the connect option under the file menu.



There are three panes in the client view. The topmost pane displays details of any license keys that have been installed. The component license pane in the bottom left displays details of which components are licensed. The concurrent license pane in the bottom right displays details of any licenses that are currently in use.

In order to install a license key using the client, use the add key option under the tools menu, and then enter the key together with the supplied expiry date.

To administer a server that exists on a different machine to the license client, use the settings option under the tools menu, and enter the IP address together with the port that the license server is listening on.

Configuration

License server configuration

All the configuration settings for the license server are held in a properties file called jlicensing.properties, located in the %JBCRELEASEDIR%/properties folder. The following settings are available;

Property	Description
jLincensingSrv.rmiport	Port number to listen on. If not specified, defaults to 3580
jLincensingSrv.log.filename	Any logging information goes to this file
jLincensingSrv.log.maxsize	Maximum size for the log file to grow to
jLincensingSrv.log.typeconsole	Where to send logging information 0=logfile, 1=console
jLincensingSrv.log.level	Level of information to log from least (0) to most (7) verbose.

jSLIM server configuration

All the configuration settings for the jSLIM server are held in a properties file called jlicensing.properties, located in the $JBCRELEASEDIR/properties folder. The following settings are available;

Property	Description
jslim.server.port	Port number to listen on. If not specified, defaults to 3561
jslim.licensing.server	Where is the licensing server. Defaults to 127.0.0.1 if blank.
jslim.licensing.port	Port number that the licensing server is listening on, defaults to 3580
jslim.log.filename	Where to send logging information. Default is jslim.log if left blank
jslim.log.maxsize	Maximum size for the log file to grow to
jslim.log.typeconsole	Logging mode, 0=logfile, 1=console, 2=window
jslim.log.level	Level of information to log from least (0) to most (7) verbose.

jLicense Console configuration

Property	Description
server.servername	Location of the license server, defaults to 127.0.0.1
server.rmiport	Port number to connect on, defaults to 3580

MIGRATION OVERVIEW

Use the following steps to migrate jBASE release to jBASE 4.1:

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 are not required, as they will be recreated in a later step. Note the location of the MD and SYSTEM files if they exist

3. Install and configure the jBASE 4.1 release

Both jBASE 4.1 and jBASE 3.4 will run on the same machine, however, there are a few points to be aware if:

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 (either 3.4 or 4.1)

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 (c:\jbase30\bin and c:\jbase4\4.1\bin) in the 'PATH' variable will dictate the order that the system will search for the commands in

iv. Under certain circumstances, jBASE 3.4 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 (possibly c:\jbase30\lib)

4. Configure a 'jbaseadm' user for system administration

This should be done at operating system level.

Note: After installation of jBASE 4.1, an 'Environment' script (or 'Environment.bat' on windows) is created it the 'jbase4/4.1/' directory. This simple script contains the environment variables used by 4.1, which can be copied into any logon script created for an administrator. However, for a 'normal' user of the account (not an administrator), the 'iju' facility should be used (see step 8 later)

5. Start the jRLA processes

The **jRLA** (jBASE Record Locking Arbiter) is used to control record locks on files

6. Configure any database / VOC / MD etc required.

8. 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.

9. 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

10. Logto each account and do step 11-17

Issuing the command ‘LOGTO <*accountname*>’ in the jSHELL will connect the user to the selected account that was created in step 7 above. 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)

11. 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.

12. 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.

13. 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

14. 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

15. 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 4.1 standards. The program automatically creates a back up copy of the original file and makes any changes necessary to the file.

16. Optionally create ‘OBJECT’ data sections for source dir/files

New functionality in jBASE allows the OBJECT files to be created in a different location to the actual source code (to prevent ‘clutter’). This is accomplished by creating a ‘datasection’ to hold the data files.

17. 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> *

18. 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.

19. Test Application

Test Migration Area

jBASE 4.1 has been designed to enable users to run previous releases of jBASE on the same machine without either release interfering with the other.

The jRLA, record locking daemon, has a different layout for jBASE 4.1 and is not compatible with previous jBASE releases. However, a jBASE 4.1-jRLA process is completely independent of any previous jBASE release jRLA demon processes.

By having two different jRLA, systems, users working on the same machine can work on a previous jBASE release as well as the jBASE 4.1 release. However, if a user takes a lock using a jBASE 3.4 jRLA it will not be recognized by a program executing under the jBASE 4.1 release. Hence when testing the migration to 4.1, users should be sure to use a discreet set of test data such that locks taken for different releases do not cause confusion.

Migrating Code

Optionally, convert source files for International Mode using the ‘jutf8’ utility. This only applies to source intending to execute in International Mode.

Delete all old dollar object files. This step ensures that the catalog procedure does try and link elderly object files from previous jBASE releases into shared libraries.

Execute PORTBAS against all program files

Changes reserve word variables to initial caps

Insures that name of a subroutine and the name used on the SUBROUTINE statement are the same

Compile and Catalog program files

Test

Note: Utilities ‘jconvertfile’ and ‘jcompilefile’ may be of use to automate and generate reports for the PORTBAS, BASIC and CATALOG procedures.

UPDATEMD

Use the ‘updatemd’ utility on all migrated accounts as the MD/VOC entries are not compatible with previous jBASE releases. The environment variable JEDIFILENAME_MD should be configured to point to the MD or VOC for the account.

The ‘updatemd’ utility is used to copy the standard jBASE master dictionary entries (jQL commands and modifiers) to the master dictionary referenced by the JEDIFILENAME_MD environment variable. The ‘updatemd’ utility should only be required when upgrading from a previous jBASE release or importing new accounts into jBASE.

NOTE: the ‘updatemd’ utility will overwrite the existing MD/VOC entries with jBASE versions, hence any user entries that conflict with jBASE entries will be overwritten.

Migration Notes

Miscellaneous

Removal of requirement for root privileges

As jPML is no longer required, therefore single point of failures has been removed. The JEDI_SHM_WORKFILE and jPMLWorkFile is no longer required.

Reduced use of jBASEWORK file

Removal of any set user-id programs.

Spooler now reliant on operating system security

jBASE j1 and j2 files no longer supported.

Port number allocation changed and still work in progress.

Background jobs can now be started using the Unix & operator, as the ttyname no longer used as the base for the port number.

the -Jb option on Unix and Windows is supported to start the processes in background.

Port disconnection now occurs on exit from the jBASE process and the port then becomes available for other users.

Mixed case commands more consistent

jRLA now available on Windows.

jRLA utilizes different resources from previous jBASE releases, hence cannot use jBASE 4.1 and jBASE 3.4 programs on the same data files and expect locking to work. Remote file must be used to access from jBASE 3.4 to jBASE 4.1 and vice versa to respect locks.

Mixed case commands have been made more consistent. In general, both lower-case and upper case versions exist. For example, jRLA is actually jrla, and UpdateMD is updatemd.

PUTENV

The use of the PUTENV statement should be unaffected. Normally the PUTENV is used to change environment variables then PERFORM another program; this should still work as the environment variables are exported for external program execution.

Compiling and Cataloging

When you BASIC and CATALOG a program, and should you subsequently try a PERFORM in the same program, it will no longer work, due to changes caused by the new thread model.

If you run a program called myprog while you are in the jshell, the shared object myprog is attached to the jshell. When myprog is recataloged, it creates a new-shared object. When you run it, the old version of myprog runs, because that is already attached to the jshell. Therefore, when cataloging programs, it is necessary to exit the jshell and then go back in again in order to access the newly cataloged program.

New Compile Command

A single jcompile command has replaced the jBASE BASIC and jBuildSLib command.

Debugger Information

The ! and .d files no longer exist - the debugger information is built into the object itself.

Executables as Shared Libraries

When an executable PROGRAM is built with CATALOG or jcompile, jBASE not only creates an executable, but also a shared library version (.dll or .sl or .so) for thread usage. When a PERFORM statement attempts to execute a program, the statement first looks for a shared library version of the program and if found executes the program as a thread. If the shared library version of the program cannot be found then the program is executed as an external program in a child process.

LOGOFF versus Killing Processes

If an Application is using the advanced thread model, where multiple ports are working in the same process: for example as a Web thread pool or using Connection Manager, it is not advisable to simply 'kill' the process as all ports in that process will disappear. Use the LOGOFF command.

Select Lists and PERFORM/EXECUTE

Normally, the PERFORM statement starts a new program as a thread. Parameters like PASSLIST, RTNLIST, PASSDATA, and RTNDATA will all work. However if PERFORM is forced to use a new process, rather than a new thread, this functionality will no longer work as expected. This is because PERFORM starts an entirely new process, and is allocated a new port number, and because parameters such as PASSLIST, RTNLIST, PASSDATA, RTNDATA are only applicable while running in the same port number that information is not passed from one process to another. This also applies to named commons as it's similar to starting a new user. The new user doesn't share anything with the old user. Once you PERFORM a program outside the jBASE process, this information is no longer available.

For example:

```
PERFORM CHAR(255):"k":"UnixScript"
```

The statement will create a new child program in a different process and hence data is no longer passed between them.

Application Performance

Due to changes in the efficiency of various jBASE components for performance reasons applications may need to be programmed differently. Some components, such as jQL and the PERFORM statement, are substantially quicker, while International Mode enabled accounts will run slower as a result of additional overhead associated with the LEN function

Shared Library naming convention changed

The jBASE jLibDefinition file, used to determine how to name and create shared libraries when using the CATALOG command, has been changed. The %a format is no longer provided as the default format for UNIX installations. The default format is now just %n, meaning that the shared libraries should be named in an ascending numeric sequence.

E.g

Eld jLibDefinition lib%a%n.so

This format means that shared libraries created when CATALOGing subroutines would use the user id as part of the library name. For instance user id ‘mike’ would cause shared libraries to be created as libmike0.so, libmike1.so, libmike2.so and so on.

New jLibDefinition lib%n.so

This format means that the shared libraries will now just be created as lib0.so, lib1.so and lib2.so and so on, irrespective of the user id.

Users are still able to modify the default jLibDefinition file, which is taken from the config directory of the jBASE release directory.

Note: all library directories should be deleted and the programs re CATALOGed to avoid potential confusion if this file is modified.

JBASE INTERNATIONALIZATION

What is Internationalization?

More and more applications are crossing international and cultural boundaries and as such need to be localized to provide support for the local language of the user. Internationalization is the development of software that can be localized for user communities without changes to executable code.

jBASE Environment Variables for Internationalization

The following environment variables are used for Internationalization. Refer to the jBASE Internationalization documentation for details as to how to configure them.

JBASE_I18N

JBASE_CODEPAGE

JBASE_LOCALE

JBASE_TIMEZONE

JBASE_I18N=1

JBASE_CODEPAGE=iso-8859-1

JBASE_LOCALE=fr_FR