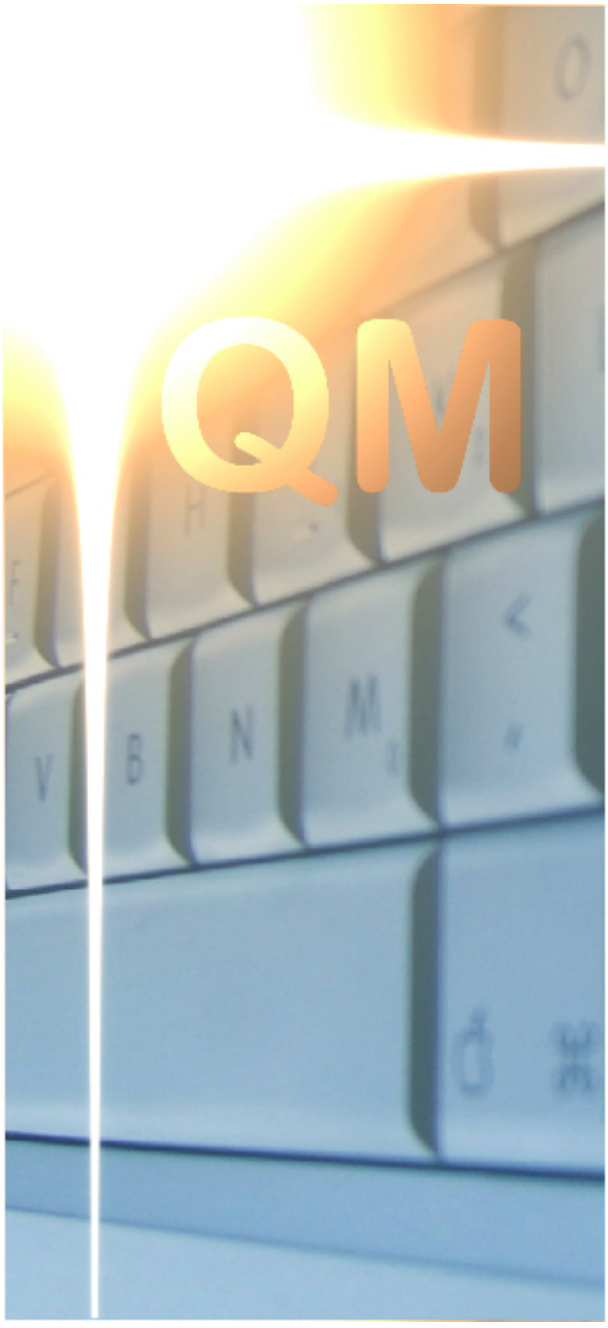

OpenQM

3.4-12

Converting Applications to OpenQM

Converting Applications to OpenQM

Publisher

Ladybridge Systems Limited
17b Coldstream Lane
Hardingstone
Northampton
NN4 6DB
England

Technical Editor

Martin Phillips

Cover Graphic

Ishimsi

Special thanks to:

Users of the OpenQM product who have contributed topics and suggestions for this manual.

Such information is always very much appreciated so please continue to send comments to support@openqm.com.

Table of Contents

1	Introduction	5
2	The Command Environment	6
3	The File System	9
4	Dictionaries	10
5	The Query Processor	11
6	Programs	12
7	Printing	16
8	Data Migration	17
9	QM Extensions	23

1 Introduction

OpenQM, more usually referred to as QM, is not a clone of any other multivalue database product. Whilst it includes many of the same features as other environments such as UniVerse, Unidata, D3, etc, there are differences. Users migrating applications will find that some features of their previous environment may not be present in QM. There may also be features in QM that provide an alternative, sometimes better, way to achieve the same thing.

Multivalue database products can be classified broadly into two types; Pick style (D3, Reality, etc) and Information style (UniVerse, Unidata, etc). As a complication in this classification, some of these environments provide mode settings to make them appear more like the other group. For example, UniVerse has six "flavours" that provide close compatibility with other products. It is classed as an Information style environment because this is its default mode of operation.

QM can be classified as an Information style product but, like some of the others mentioned above, it too has mode settings to give closer compatibility with other environments. By careful selection of modes, it is usually possible to simplify the migration process, however, migration of a realistic application will rarely be simply a case of moving the software and recompiling. There will always be work to do but this should be minimal unless the application makes extensive use of non-portable aspects of the previous environment. Ladybridge Systems would very much like to hear from users who encounter migration issues that are not discussed here.

This guide brings together hints and tips from users who have performed this migration task and should serve as a roadmap towards successful implementation. Readers should also refer to the *QM Reference Manual* for more information.

The sections in the guide are:

- The command environment
- The file system
- Dictionaries
- The query processor
- Programs
- Printing
- Data migration
- External interfaces

2 The Command Environment

The QM command processor interprets commands using the VOC (vocabulary) file. The role of this file is broadly similar to the MD file of Pick style systems but the records in it are very different. Users migrating from Information style environments will recognise the VOC as being very close to that of their previous system. The VOC always contains a Q-pointer to allow use of MD as a synonym.

A command line is considered as being made up of a number of tokens separated by spaces. Where a token contains spaces, it must be enclosed in quotes. In general, every token on the command line is looked up in the VOC file to find out what it means. Query processor commands extend this process by looking the token up in the dictionary first and only referencing the VOC if the token is not in the dictionary.

Although VOC record ids are normally case sensitive, the QM command processor gives the appearance of case insensitivity by using a two stage lookup process in which it first tries to find an item exactly as it was typed and, if it is not found, tries again in uppercase. Because the standard VOC items are all stored in uppercase, this effectively means that an item will be located, whatever case is used on entry. A similar approach is used by all built-in commands for dictionary lookups. QM also supports files with case insensitive record ids and it is possible to convert the VOC to use case insensitive ids.

Many Pick style commands include hyphens in their name (e.g. **CREATE-FILE**). In general, the equivalent QM command uses a dot in place of the hyphen (e.g. **CREATE.FILE**). To simplify migration and to avoid the need for duplicate VOC entries, the command processor will apply an automatic transformation of the first word on the command line if it is not found in the VOC as typed so that **CREATE-FILE**, for example, will actually map on to the **CREATE.FILE** VOC entry. This transformation does not apply to the remaining words of the command but synonyms are provided for many keywords and other appropriate tokens.

As mentioned in the introductory section of this document, the QM command language is based on the Information style model and its general syntax and semantics are closest to the now retired Prime Information multivalue database product but also not very different from other products such as UniVerse and Unidata. The **OPTION** command can be used to modify several aspects of command processing to give closer compatibility with Pick style environments. It is strongly recommended that developers undertaking a migration should start by reviewing the option settings available and inserting the relevant commands into the LOGIN script in the VOC file.

One command that frequently causes problems is **COPY**. The Information and Pick style syntaxes of this command are very different. The QM **COPY** command follows the Information style syntax. There is also a **COPYP** command for Pick compatibility. The **ALIAS** command can be used, typically in the LOGIN script, to map **COPY** onto **COPYP** for Pick style users if required.

QM is much more flexible than Pick style products regarding use of quotes in commands. A literal value may be enclosed in single quotes, double quotes or backslashes. In most contexts, there is no distinction between these. The role of the literal is determined by its context rather than the type of quote used. It is often possible to omit the quotes completely but beware that the token will be looked up in the VOC (and the dictionary for query processor commands), possibly with undesirable effects. In particular, the tendency for Pick programmers to define fields with numeric names in the dictionary is likely to cause problems with unquoted numeric values.

Accounts

An account is a place to work, typically corresponding to an application. A QM environment will often have several accounts, perhaps for different applications, perhaps for different versions of the same application. Accounts are identified by a case insensitive name of up to 32 characters. At the operating system level, an account is simply a directory containing the main files associated with the application. It is possible for one account to reference files in another account and for files to be positioned in other directories to aid load balancing across multiple disks.

The QMSYS Account

Every QM installation includes a special account known as the QMSYS account. By default, this is located in c:\qmsys on Windows or /usr/qmsys on Linux but it can be moved elsewhere during installation.

The QMSYS account contains all components of QM itself. It includes some additional system administration commands that are not available in other accounts.

The QMSYS account includes a file named ACCOUNTS (visible from all accounts under the name QM.ACCOUNTS) which acts as a register of account names, providing a mapping between the name and its corresponding operating system directory path.

Applications should be contained in their own separate accounts. Placing application components in QMSYS may result in data being overwritten at a subsequent QM upgrade.

Entry to QM

Users may enter QM from the operating system command prompt or directly via a telnet style connection. For the latter path, QM will prompt for the user name and password for authentication purposes and then prompt for the name of the account that the user wishes to enter. Alternatively, this account name prompt can be omitted by setting up a fixed relationship between user names and account names (see the **ADMIN.USER** and **CREATE.USER** commands for more details of this process).

When a user enters QM, the command processor looks in the VOC file of the QMSYS account for a paragraph (PA-type item) named MASTER.LOGIN. If this exists, it is executed and is typically used to perform system wide initialisation tasks. In general, developers should not perform any application specific tasks in this paragraph.

Having run (or not found) the MASTER.LOGIN paragraph, QM then looks in the VOC file of the target account for an item named LOGIN which may be of any executable type (verb, menu, sentence, paragraph, proc, etc) but is usually a paragraph. This paragraph is typically used to perform application specific initialisation tasks and to take the user into the application itself without ever seeing a QM command prompt.

During this process, the break key is disabled so that the user cannot quit out of any security checks performed in these paragraphs. The break key can be enabled if required using the **BREAK** command.

The LOGIN paragraph can use conditional processing to perform different actions dependent on user name, network address, etc.

3 The File System

QM uses a maintenance free, self-tuning file system. There are two basic file types; directory files and dynamic files.

Directory files use an operating system directory to represent the QM file (table) and a text file in the directory to represent each record in the table. This system does not give high performance but allows records to be accessed from outside QM as simple text files. Directory files need special processing modes if binary data that might include the mark characters is to be stored in them as otherwise reading data converts newlines to field marks and writing data performs the reverse substitution.

Dynamic files are represented by an operating system directory that contains two files named %0, %1. The actual database records are stored inside these files in a hashed format that allows very rapid access. Dynamic files automatically adjust their internal structure to compensate for changes in the volume of data stored in the file. Although there are various tuning parameters for dynamic files, the only one that is usually worth consideration is GROUP.SIZE which depends on the average size of records to be stored in the file. There is general guidance on dynamic file configuration in the *QM Reference Manual*. QM supports dynamic files of up to 16384Gb.

QM supports the concept of multi-files using a further layer of directory structure.

The alternate key index system allows access to data in dynamic files based on the content of data fields or calculated values (secondary keys). A dynamic file that has indices has further subfiles named %2, %3, etc. There is a limit of 32 indices in any single file.

QM files do not support use of null record ids or ids that contain the mark characters. The default limit on the length of a record id is 63 characters but this can be increased using the MAXIDLEN configuration parameter. There is a corresponding small increase in the size of the locking tables and files containing records with long ids may not be readable on systems with a lower value for this parameter.

Files are referenced by an F-type VOC entry which translates the application name of the file to the operating system pathnames of the data and dictionary portions of the file. QM also supports Q-pointers for access to files in other accounts. See the *QM Reference Manual* for further details. Note that extended Q-pointers that allow access to data on another machine using QMNet require the server to be defined using **SET.SERVER**.

The FILERULE configuration parameter also controls use of extended file name syntaxes:

account:*file*	Implict Q-pointer
server:*account*:*file*	Implicit QMNet pointer
PATH:*pathname*	Pathname

4 Dictionaries

The preferred dictionary record types for QM applications are the D and I-type items that originated in the Prime Information database and have been replicated in a number of other environments. QM supports the major features of A and S-type items for compatibility but does not include some of less used features of these record types.

There is a major difference in how QM handles A and S-type items compared to other environments though this should have little impact on application migration. In QM, correlative expressions are compiled into the same object code stream as used by I-type items to give faster execution than the traditional interpretive processing. The correlative is automatically compiled when it is first used in the query processor and the object code is stored in the dictionary record in fields 15 onwards though the standard editing tools suppress its display.

It is important to note that if one correlative uses the result of another and this second correlative expression is modified, it will be necessary to force the system to recompile the first one. This is because, although it might look like a run time subroutine call, the nested expression is actually substituted at compile time for optimal performance. The **COMPILE.DICT** command can be used to force compilation of all compiled dictionary record types and it is recommended that the entire dictionary is recompiled whenever a change is made that might impact other dictionary items.

Whilst it is recommended that users migrating from a Pick style system to QM might want to move to the more powerful D and I-type items, it is recognised that this may be better left as a task for completion after the initial migration is complete. Most correlatives should work without changes but QM does not guarantee 100% compatibility with other systems.

QM dictionaries include the concept of link records that minimise the need for **TRANS()** functions or T*file* conversions. A link (L-type) record defines how the record key of a dependent file is manufactured from the data in the original file using an expression similar to that found in I-type records. The query processor then allows the fields of the dependent file to be referenced in the form *link*%*field*.

QM also provides extensions to the capabilities of I-type items compared to most other multivalue products. In particular, a **TRANS()** function can return the value of an item which is itself an I-type and compound I-types can be nested without restriction.

Some conversion code features are inconsistent across different multivalue platforms and hence may need to be modified. QM includes the capability for users to add their own conversion codes.

5 The Query Processor

As already mentioned for the command processor, the QM query processor makes no distinction between the three styles of string quote (single quote, double quote and backslash). Quotes are often not needed at all though their use around literal values is advised to avoid a dictionary and VOC lookup that could substitute alternative text for the unquoted literal.

OPTION Settings

See the **OPTION** command for various mode settings that make the query processor behave more like its Pick style counterparts.

Some syntax/semantic differences compared to Pick style systems that are covered by option settings:

By default, QM does not insert an implied equals between the field name and a value in a WITH clause. Thus,

```
WITH field "value"
```

must be written as

```
WITH field = "value"
```

Without the intervening equals sign, the example above is interpreted as a short form for

```
WITH field # "" "value"
```

where *value* is taken as the id of a record to be included in the report.

The PICK.IMPLIED.EQ mode of the OPTION command can be used to enable the Pick style interpretation of the above example.

6 Programs

QMBasic program source is conventionally stored in directory type files. Although the compiler will process source from dynamic hashed files, the ability to access the source text from outside QM, possible only with directory files, is often useful. The compiled object code must be stored in directory files. The compiler output file is created automatically when first needed and takes the same name as the source file with a suffix of .OUT added.

There are two restrictions to bear in mind when choosing source record names. Firstly, names including spaces, whilst technically acceptable, are likely to cause problems. Secondly, when using a select list of programs to process, the compiler omits items ending with a suffix of .H or .SCR (case insensitive). These two suffices are conventionally reserved to identify include records used for general purposes and as screen definitions respectively. There are options described with the **BASIC** command in the *QM Reference Manual* to modify the list of special suffix codes.

QMBasic source programs are compiled using the **BASIC** command. The general form of this is

```
BASIC filename recordname options
```

where *filename* is the name of the file holding the source. This defaults to BP if omitted. *Recordname* is one or more record ids corresponding to the programs to be compiled. Use of an asterisk specifies that all records in the source file are to be compiled (except those with a .H or .SCR suffix as described above). A select list can be used instead of the *recordname* option.

The compiler takes several *options*. The **CHANGED** option causes the compiler to process only those records for which there is no up to date object code. Thus a command

```
BASIC * CHANGED
```

would compile all records in the BP file that have not previously been compiled or where the date/time modified on the source record is later than that of the object record. Note that because dynamic hashed files do not store a date/time modified, the **CHANGED** option cannot be used with source in hashed files. Note also that the **CHANGED** option does not detect changes to include records.

A good technique for recompiling affected programs after an include record is changed is to use the **SEARCH** command to build a select list of all programs that contain a reference to the modified record and then to use this list to steer the compiler.

Cataloguing

QM uses a three level cataloguing system to give maximum flexibility. The default mode (private cataloguing) places the program in a catalogue file that is visible only to applications running in the same account. There is no corresponding VOC file entry, the program being located by a search of the cat subdirectory of the account.

Local mode cataloguing, selected by use of the **LOCAL** keyword in the **CATALOGUE** command, does not copy the program but simply sets a V-type (verb) VOC entry to point to the compiled program. Again, the program is visible only to applications running in the same account.

Global mode cataloguing, selected by use of the **GLOBAL** keyword in the **CATALOGUE** command, copies the program to the gcat subdirectory of the QMSYS account. Programs in the global catalogue are visible to all accounts.

For compatibility with some other environments, global mode cataloguing can also be selected using one of four reserved prefix characters (*, !, _ and $) on the program call name. The $ prefix is reserved for subroutines that are built-in parts of QM and such items cannot be called from user programs.

If a program catalogued in private or global mode is recompiled, it will be necessary to recatalogue it to repeat the copy to the cat or gcat directories. If, however, the recompilation fails, active users will not be affected as they continue to execute the previously catalogued version of the program.

To simplify the development process, the QMBasic language includes a **$CATALOGUE** directive that causes automatic cataloguing after successful compilation.

Language Syntax Variants

The general syntax and semantics of the QMBasic language follows that of the Information style databases. The **$MODE** compiler directive can be used to select a variety of options to modify the language form. This directive can be included in each program individually as required or, more conveniently, the same effect can be achieved using the $BASIC.OPTIONS record.

The $BASIC.OPTIONS record is described with the **BASIC** command in the *QM Reference Manual* and in the help text. This record can appear either in the program source file from where it affects all programs in that file, or in the VOC where it affects all programs. If both locations contain a $BASIC.OPTIONS record, the one in the program source file takes precedence.

The QMBasic statement that causes the most confusion during migration is **LOCATE**. There are three different forms of this statement in use in the various multivalue database products, two of which are syntactically similar but semantically different. QM adopts the somewhat illogical Information style syntax by default but has a mode setting to adopt the more sensible form found in UniVerse, Reality, etc. QM also adds a function call variant which is very useful in dictionary I-type items. For more information, see the *QM Reference Manual*.

Other statements affected by use of **$MODE** are **READNEXT**, **FOR/NEXT**, **STOP**, **ABORT**, **ON GOSUB**, **ON GOTO**, **COMMON**, **DIMENSION**, **READ** (various forms), **PRINTER** and **SELECT**.

Some syntax/semantic differences compared to other environments and not covered by **$MODE** options:

A comment on the same line as a statement must be separated from the statement by a semicolon. For example

```
IF STOCK = 0 THEN * Out of stock
   ...
END
```

is not valid and must be rewritten as

```
IF STOCK = 0 THEN ;* Out of stock
   ...
END
```

Use of multiple statements separated by semicolons in a single line format conditioned statement is supported but not recommended as it is syntactically ambiguous and is handled differently on the various multivalue database products. A statement such as

```
IF STOCK.LEVEL = 0 THEN MSG = "Out of stock"; GOSUB EMIT.MSG
```

should be rewritten as

```
IF STOCK.LEVEL = 0 THEN
   MSG = "Out of stock"
   GOSUB EMIT.MSG
END
```

The **CONVERT()** function is syntactically different compared to D3 such that

```
CONVERT(str, old, new)
```

becomes

```
CONVERT(old, new, str)
```

The return key results in character 13 (carriage return) by default whereas some other systems use character 10 (linefeed). The **PTERM RETURN LF** command can be used to change this to character 10.

Scanning Alternate Key Indices

QMBasic does not support the BSCAN statement of UniVerse or its equivalent in other multivalue products. Instead, the equivalent functionality is provided by use of SETLEFT and SELECTRIGHT to position to the leftmost indexed item and walk step by step to the right (or SETRIGHT and SELECTLEFT to traverse from right to left).

Using UniVerse as an example,

```
bscan key, id.list from fvar using 'index.name' reset
then
   loop
      loop
         id = remove(id.list, more)
         ...process data...
      while more
      repeat
      bscan key, id.list from fvar using 'index.name' else exit
   repeat
end
```

becomes

```
setleft 'index.name' from fvar
loop
   selectright 'index.name' from fvar setting key to 1
until status()
```

```
      readnext id from 1 else exit
      ...process data...
   repeat
```

Other Syntax and Semantic Differences

QM support for the concept of a default file variable must be enabled, if required, using the $MODE compiler directive.

There must be no space between a matrix name and the left parenthesis preceding its index expression.

There must be no space between a function or subroutine name and the left parenthesis preceding the argument list.

Although QMBasic has very few reserved words, some uses that are acceptable in other implementations may fail on QM.

Portable Software

QM ships with a pre-processor that allows **$IFDEF** conditional compilation structures to be used in environments that do not support them internally (e.g. D3). This allows a single source stream to be used for multiple environments. See the **$IFDEF** compiler directive in the *QM Reference Manual* for details.

7 Printing

The QM printing system is modelled on the same general structure as is found in the Information style database products. Each user process has a set of 256 numbered print units to which it can send output. The physical destination for this output is determined using the **SETPTR** command or other related commands and subroutine calls.

This system gives flexibility in that an application has no built in knowledge of where the output will go. A single print unit may be directed to different destinations for different users. Print unit numbers may be used to select different printers, different paper types on the same printer, use of portrait or landscape mode, etc.

The **SETPTR** command defines the shape of the page (width, depth, margins) and the actual destination. If output is directed to a printer, the command sets the printer name, number of copies, etc and output is passed to the print management system of the underlying operating system. Output can also be directed to a file from which it can subsequently be printed or processed by other software.

Within the command language, many commands have an optional **LPTR** keyword to direct the output to the default printer (print unit 0). In some cases, the **LPTR** keyword can be followed by a print unit number to specify use of a non-default printer.

Within programs, the **PRINT** statement **ON** clause can be used to select a non-default print unit. This clause also applies to other printing statements such as **HEADING**, **FOOTING**, **PAGE**, etc. There are also a number of statements, functions and subroutines that allow control of print unit settings.

On Linux and FreeBSD systems, the default spooler is lp. This can be changed for a single print unit using the SPOOLER option of the **SETPTR** command, or for all print units using the SPOOLER configuration parameter.

QM includes a variant of the Pick style **SP.ASSIGN** command that should ease migration. This command uses a database set up using **SET.QUEUE** to map Pick style form queues onto the corresponding **SETPTR** options.

8 Data Migration

There are six main paths for migration of data between QM and other systems:

- Flat files
  A simple data export program and corresponding data import program can usually be written in just a few lines of Basic.

- AccuTerm host to host transfer
  The bundled AccuTerm terminal emulator includes facilities to transfer data between two systems.

- ACCOUNT.SAVE / ACCOUNT.RESTORE
  QM includes versions of these standard utilities that are closely compatible with other multivalue products. More information about use of these tools for migration appears below.

- QMSAVE / QMRESTORE
  The **QMSAVE** program is released in source form and can compiled on other systems (possibly after modification) to save data that can subsequently be restored in QM using the QMRESTORE command. These tools are described in the QM KnowledgeBase on the openqm.com web site.

- UVIMPORT / UDIMPORT
  These tools, documented online in the QM KnowledgeBase, can directly read data from UniVerse and Unidata files, converting them to QM format.

- JRESTORE
  This tool can be used to restore a JBase backup directly into QM.

Any migration to QM will require some manual work in constructing VOC file entries. Dictionary items are broadly compatible but may need inspection and some modification. There may be other system files that require conversion.

ACCOUNT.RESTORE and associated commands have been tested with images from several flavors of Pick (from R83 through AP and D3) and mvBASE. ACCOUNT-SAVE images from these systems should restore with no special attention.

To restore from FILE-SAVE tapes created by the above systems, one must use the T-RDLBL and T-FWD to read through the FILE-SAVE image and locate the required account. When the label for the desired account is found, the POSITIONED keyword to **ACCOUNT-RESTORE** can be used to restore that account. After the account has been restored, T-RDLBL can be used to check the identity of the next account on the tape.

Many multivalue systems are capable of producing a 'compatible' tape image but some are more compatible than others and will restore into QM with very little difficulty. Others require more care and there is the possibility that some may not be restorable.

The tape utilities support 'multi-reel' operation in which cascaded files emulate multiple spools of tape. Two naming conventions are supported:

The D3 style 'dash-number' format:

filename
filename-1
filename-2
... and so on ...

A filenameNN format:

filename00
filename01
filename02
... and so on ...

This format allows up to four digits appended to a base filename. The base name may be as above (filename00; filename0000; etc.) or a 'bare' file name as in the D3 style. With the numeric suffix, the base name can be suffixed with any number; this number will be incremented by the tape utilities. If a 'bare' name is used, the next reel is expected to be '1'; '01'; '001' or '0001' and all of these will be attempted.

If an End Of Tape is sensed (e.g. a read failure before an EOF or EOT mark is read), the above file names are attempted. If the 'next reel' is found it is silently attached and the restore continues. If the tape utilities are unable to find a file then the user is prompted for a file name.

Tips on Creating Tapes for Migration to QM

These notes have been contributed by users who have performed the associated migration.

Normal AP and D3 tape images created with no special options should restore directly with no special options used for the SAVE. As with all 'foreign' systems, when restoring into QM, the 'NO.OBJECT' keyword should be used to prevent Basic object code from being imported.

D3 Tape Image Creation

The sequence of D3 commands to generate a pseudo-tape is:

1. dev-make -t tape -a "drive:\directory\filename,p"

This will add a pseudo tape to the device list, typically device 3

2. set-device 3

3. t-rew

4. save (dfta

Because the media format used by the save command uses char(251) for its own internal purposes, it may be worth writing a routine that removes all "char(251)" from your data before moving it to QM.

SAVEs of data that will exceed 2gb will have to be split into chunks of less than 2gb because D3 will not write to files larger than 2gb.

Modern versions of D3 support 'cascading' files (filename; filename-1; etc.). While this is the recommended method, it also introduces a problem in that the cascading filename scheme only works when compressed files are specified. Since QM does not support compressed files, the resultant D3 files must be uncompressed before then can be used.

A simple method of D3 file preparation for Linux is to first rename the files so that they include a '.gz' suffix, then uncompress them. This can be accomplished as follows (assuming the files are in the current directory and are named 'filename'; 'filename-1' and so on):

```
for file in `ls filename*`
do
   mv ${file} ${file}.gz
done
for file in `ls filename*.gz`
do
   gunzip ${file}
done
```

After the above process, the files will exist with the original filenames but in uncompressed form and may be used by attaching the first file (in this case 'filename') in QM:

```
SET.DEVICE /path/to/filename
```

The tape utilities will take care of traversing the set of cascaded files.

After the restore process, any alternate key indices that existed in the D3 system will be created in QM.

The MD of the restored account will exist in the newly created QM account under the name 'MDRESTORE'. From this file any Q-pointers or custom verbs which will make sense to QM may be copied into the VOC. Any custom verbs that do not make sense in QM will be available in MDRESTORE file for examination.

ACCOUNT.RESTORE will create indices with the correct QM name if field 17 of the FDI on the D3 system contains the desired name prefixed with "QMI". Thus, if field 8 of the FDI contains

```
I450533A1(G1*1)V_MI4587439A1(G1*1)(TVENDOR;X;26;26)
```

and field 17 contains

```
QMIProdIDV_MQMIVendor
```

the restore process will create indices named ProdID and Vendor.

Ultimate Tape Image Creation

Use the T-ATT command to attach the tape device:

T-ATT <drive number>,<block size>

For example,

T-ATT 1, 512

Alternatively, to save to disk instead of tape, use

D-ATT <drive number>,<block size> <path to unix directory>
For example,
D-ATT 3,512 /virtsave/ult

Then use T-DUMP, ACCOUNT-SAVE, etc to create the tape:
1) T-DUMP <filename> or T-DUMP DICT <filename>
2) ACCOUNT-SAVE
3) File Saves are usually run from the SYSPROG account, menu option 1.

ADDS Mentor Tape Image Creation

On the Mentor system, attach the QIC tape with a block size of 512:

```
T-SELECT (S
T-ATT 1
T-ONLINE
SAVE SYSTEM SYSTEM (TY
```

The resulting tape may then be copied into a *nix system for use by QM by creating a small shell script named, for example, copy.tape:

```
#!/bin/bash
# script copy.tape - copy tape records from an ADDS Mentor
tape
ctr=0
while [ $? -eq 0 ]
do
  ctr=`expr ${ctr} + 1`
  cat < /dev/nst0 > adds`printf "%03d" ${ctr}`
done
```

(The above example uses a tape device name of /dev/nst0. This may need to be amended to match your system).

Make the script executable:

```
chmod 755 copy.tape
```

Create a directory to contain the files then run the script. This will create a file for each account on the tape such as:
adds01
adds02
adds03
adds04
and so on.

The first file of the set can then be attached in QM:

```
SET.DEVICE /path/to/adds01
```

The accounts must then be restored individually through a series of commands:

```
ACCOUNT-RESTORE NO.OBJECT POSITIONED
   BOT
   Block size : 512
   Illegal level (0)
T-FWD
T-RDLBL
   L 0200 16:18:41  19 MAY 2006 DATA SYSPROG FULLSAVE
  01
ACCOUNT-RESTORE NO.OBJECT POSITIONED
```

```
      ...restore messages...
   T-RDLBL
      L 0200 16:19:15  19 MAY 2006 DATA ACC FULLSAVE
     01
   ACCOUNT-RESTORE NO.OBJECT POSITIONED
      ...restore messages...
```

and so on...

The first ACCOUNT.RESTORE and T.FWD are required to position the tape image beyond the first header.

Restoring from Untested Systems

The format of ACCOUNT.SAVE tapes varies between multivalue products. A Pick style "compatible mode" (R83 format) tape commences with

Label
EOF block
Label
Descriptor block
EOF block
Label
Data.....

ACCOUNT.RESTORE therefore normally starts by skipping forwards to the third label block. On some systems, the tape commences simply with a label block followed immediately by the data. To allow for the possibility of this and other formats, the **T.RDLBL** and **T.READ** commands can be used to position the tape before the first data block. Use of the POSITIONED option in **ACCOUNT.RESTORE** will then omit all other positioning from within **ACCOUNT.RESTORE** itself.

If an **ACCOUNT-RESTORE** fails (symptoms include 'Unrecognised block type' errors as well as 'End of reel reached' errors), one may use the tape utilities to attempt to discover the best method of properly positioning the tape before using the POSITIONED keyword.

Using **T.READ** on a QM (and Pick) style tape that is positioned at BOT (Beginning of Tape; after a **T.REW**) will produce something like:

```
:T.READ
L 01F4 22:14:12  31 Mar 2006 DATA QM.TEST
    01
Block size : 500
End of file.
:T.READ
L 01F4 22:14:12  31 Mar 2006 DATA QM.TEST
    01
Block size : 500
End of file.
:T.READ
Block size : 500
End of file.
:T.READ
L 01F4 22:14:12  31 Mar 2006 DATA QM.TEST QM.TEST
    01
```

```
Block: 2
000 :_D^1^AD^0012QM.TEST^D^B9^101^^^^^SYS2^L^12^_D^2^AF:
050 :^0017MD-RESTORE^D^101^11^^^^^^L^10^_D^3^B0^0016MD-:
100 :RESTORE^D^101^                        ... and so on ... output
trimmed
```

Note the three blocks mentioned above (requiring three **T.READ** commands). Note also the '_D' beginning Block 2 - this is the D-Pointer or File Description Item for the 'QM.TEST' account and is where the actual **ACCOUNT.RESTORE** will begin.

Using a tape created by a jBase system using the command:

```
    T-ATT FILE0 DEVICE=C:\BACKUP1.VTF SIZE=512
```

Then reading from the beginning, the output of **T.READ** appears:

```
:T.READ
L 0200 22:05:32  30 MAR 2006 QM.TEST
    01

Block: 2
000 :_D^1^1^0031QM.TEST^D^20480^1^1^^^^SYS2(0)^L^10^^^(:
050 :29,1)^_I1^5000^0009MD^Q^_D^2^2^0024BP^D^20481^1^1^:
100 :^^^^L^10^^^(5,1)^_D^3
```

... and so on ... output trimmed

Note that a single T.READ command displays both the tape label and the start of account data (the '_D' bit again).

When this tape image is attached and an **ACCOUNT.RESTORE** command issued, the initial attempts to skip three blocks result in the program seeking to the end of tape data and prompting for the next reel.

Although this tape may appear to be incompatible with QM, by inspecting the image as above we can see that by issuing one **T.RDLBL** command the tape pointer will be positioned at the beginning of account data. Then by issuing the command:

```
    ACCOUNT.RESTORE POSITIONED NO.OBJECT
```

this tape will in fact restore without problem.

Using the above as a guide, if difficulty is encountered one can experiment until the correct positioning is discovered.

9 QM Extensions

This section summarises some of the features that are believed to be unique to QM or not found in most other multivalue database environments.

File System

The QMNet network file system uses an extended form of the VOC Q-pointer to reference files on remote servers. Full concurrency control is maintained to ensure data integrity across the entire network.

Extended file name syntax forms, controlled by the FILERULE configuration parameter, allow access to files without needing a local VOC entry.

account:*file*	Access a file in another account without a Q-pointer
server:*account*:*file*	Access a file on another server without a Q-pointer
PATH:*pathname*	Access a file by pathname

Dictionary Issues

Other multivalue databases require I-type dictionary items or T conversion codes to link tables together. Although these are fully supported on QM, dictionaries can contain L-type (link) records that describe the relationship between files. The query processor can navigate using these without the need for each field to be defined via a separate **TRANS()** or T conversion.

In other multivalue databases, a **TRANS()** function can only reference a real (D-type) data item, not a calculated value. With QM, the target of a **TRANS()** can itself be an I-type item. This significantly simplifies some dictionary constructs which would otherwise require duplication of expressions in multiple dictionaries.

QM can nest compound I-types (those with multiple expressions) to almost any depth.

The **GENERATE** command can be used to generate a QMBasic include record from a dictionary, guaranteeing consistency and simplifying application development.

Command Language

The command stack can be accessed directly by use of the cursor keys.

It is possible to list and delete named common blocks from the command line - a highly useful feature in a development environment.

QM can return entries from a select list, item by item, using an inline prompt construct in a loop.

The value of @-variables can easily be substituted into any command by use of an extension to the inline prompt mechanism.

Users can define their own @-variables within paragraphs, setting values as constants or the result of arithmetic expressions.

Users can add their own VOC record types and associated handler subroutines for special executable type items.

Query Processing

The query processor can produce reports in the form of comma separated files (or any other delimiter).

Reports that are wider than the terminal can pan across selected columns.

The SCROLL option allows users to scroll back through earlier pages of a report.

Query processor styles can be used to highlight some report elements (e.g. totals) using colour on displayed reports or font weights on reports sent to PCL printers.

QM extends the syntax of the SAVING keyword to work with multivalued fields.

QMBasic Programming

Programs can include directives to catalogue them automatically after compilation. Alternatively, this can be set as the default for an individual program file or the entire account.

Programs that use **EXECUTE** to launch a QM command can trap aborts occurring in that command instead of the default action of aborting the entire application.

Low level keyboard input functions are provided that can recognise cursor keys and other function key sequences, returning a device independent code.

The Hot Spot Monitor allows developers to identify where their programs spend most of their time. This is very valuable as a performance tuning aid.

The **DPARSE** statement provides a simple way to parse delimited data (e.g. comma separated items).

The **LOCATE** statement has a function variant that is particularly useful in I-type dictionary items.

The **LISTINDEX()** function can be used to search a list with an arbitrary delimiter.

The **SUBSTITUTE()** function replaces text from one list with alternatives in another list, working through each element of a potentially multivalued source item.

The QM command line parser is available for use in user programs.

All QMBasic functions are available in I-types except those that are meaningless in this context.

QMBasic supports the concept of local variables in internal subroutines. It is also possible to pass arguments into internal subroutines and to have internal functions. See the LOCAL statement in the QMBasic Programming Guide for details of these features..

QMBasic has support for object orientated programming, exception handling and arbitrarily multi-dimensional data collections.