Getting Started in OpenQM - Part 1

Version 2.0

July 2013

Brian Speirs

0 Preface

0.1 Purpose Of This Book

Multi-value databases (also known as PICK[1] databases) are one of the best kept secrets in the computing world. They began life in the late 1960s, flourished through the 1970s and early 1980s, before virtually disappearing from view in later years.

This low profile has been somewhat problematic in that it has led to a vicious circle in terms of resources available. Low visibility leads to low growth which leads to few resources being made available to broaden the market. Therefore, visibility and growth both remain low.

OpenQM could become a product to help break this cycle. It is readily available for download; there is a Personal version available at no cost; and the full product has a far lower cost than other multi-value databases.

However, there remains the problem of resources. In particular, there is no "Multi-Value Databases for Dummies" book available as there is for many other software products. Those people who do download *OpenQM* are either familiar with the multi-value model already, or are often left struggling to understand what to do once the install program has done its job.

This book is a first step towards providing a resource to help new users of *OpenQM* to learn about the system. It covers the basic issues of installation, configuration, creation of accounts and files, populating the data files, and querying the database.

Most "getting started" books show you one way to do things, and those things always work in the examples given. This book is based more in the real world.

1 They are called PICK databases because this style of database was first commercialised by Dick Pick and his company Pick Systems. That company is now part of Tiger Logic Corporation..

It shows you multiple ways to do things. It shows you things that look as if they should work – but don't. It shows you ways that work – but then asks if there is perhaps a better way to do that task. It recognises that you learn from doing, and that you learn even more when you make mistakes.

This book can't cover everything there is to know about OpenQM – not even in the restricted areas that are covered. Within QMQuery, there are various verbs, keywords, and modifiers that have not been illustrated. Likewise, not all dictionary types have been discussed.

Other areas have not been touched at all. These include use of the *QMBasic* programming language; use of *QMClient* to connect *OpenQM* to external Windows applications; and the use of *AccuTerm*'s utilities for GUI programming from within a multi-value environment. Part 2 of *Getting Started in OpenQM* covers some of these other areas, but even that does not cover everything.

It should also be noted that this book does not really cover areas such as database design – although it does make an extended digression into the related area of appropriate item-ids. Readers are encouraged to learn about database design from other sources.

Although this book primarily covers *OpenQM*, much of what is shown here applies to other multi-value databases too. In particular, *OpenQM* uses syntax that is broadly similar to that of *UniVerse* and *UniData*. So this book could be used as an alternate introduction to the *U2* Databases. While it could be used as an introduction to the other multi-value databases (such as *mvBASE* or *D3*), the differences between *OpenQM* and those databases are much greater (particularly in the format of dictionary items) that this book could only be a general guide.

0.2 Changes from Version 1.0

Changes include:

- Revision of all examples. While some filenames remain the same, the actual data and data structures are different. This was partly due to the removal of one of the source data files from the RBNZ website.
- Addition of sections on alternate key indexing, use of *QMBasic* subroutines from with *QMQuery*, and cross-tab reports.
- Updates to cover enhancements to the OpenQM product.
- Numerous revisions throughout the text and formatting.

0.3 About The Author

Brian Speirs began his career as an economist – but he used multi-value databases extensively during his economics work. Most of that experience came from the organisation that is now Beef + Lamb New Zealand – although it had a variety of names during Brian's time with the organisation.

His initial exposure was in 1986 to the *ADDS Mentor* environment. That environment evolved into the *Mentor Operating Environment*, and then to *mvBASE*. Along the way he got some exposure to *UniVerse* through a related organisation.

On leaving Meat & Wool NZ (one of the predecessors of Beef + Lamb NZ), Brian looked around for a multi-value database for personal use and found *OpenQM*. Brian quickly came to prefer using *OpenQM* to the alternatives of *mvBASE* and *UniVerse* that he also used at that time.

Version 1.0 of this book was released in February 2008. It was written in Brian's spare time while he worked at the NZ Institute for Economic Research.

Soon after this, Brian spent 4 years in the United Kingdom. During that time he worked for CACI in their Twickenham office working as a Software Engineer on their OfficeBase product. OfficeBase is based on *UniData*, and uses an extensively modified SB+ framework and a CACI developed web front-end.

Part 2 of *Getting Started in OpenQM* was written during Brian's time with CACI. That book covers how to create an application in *QMBasic* using the *AccuTerm* GUI programming environment.

Since returning to New Zealand in 2012, Brian has been working as a consultant. Once again, the revision of this book has been done in his spare time.

Brian would welcome feedback on this book. He can be contacted at brian@rushflat.co.nz

0.4 Contents of This Book

This book is organised as follows:

Chapter 1 is an overview of multi-value databases. Some of *OpenQM*'s unique characteristics are briefly covered here, along with an overview of the terminology used by multi-value databases.

Chapter 2 covers configuration of *OpenQM* and *AccuTerm*. Most installation notes have been move to the Appendix (Chapter 11).

Chapter 3 gives a first look at the way *OpenQM* appears at the file system level, and from the user perspective. The purpose of visible files is briefly covered.

Chapter 4 introduces accounts, database files, and provides a brief introduction to the editors available in *OpenQM*. Importantly, this chapter covers the creation of an account within *OpenQM* for use through the rest of this book, and populates that account with a number of data files.

Chapter 5 starts to introduce *QMQuery*. This is the database query language used by *OpenQM*. A key part of understanding *QMQuery* is the creation of dictionary items that define the data held in the data files. This chapter steps through the basic elements of querying the database including record selection, sorting, grouping and breaking, output fields, and report headings and footings. It also covers printing, report styles, and saving queries for future use.

Chapter 6 continues the exploration of *QMQuery*, starting with more advanced record selection techniques, and moves into looking up data in other files, carrying out calculations on data in the data files, writing the results direct to O/S level output files, creation of new data files from one or more source files, and working with multi-valued data.

Chapter 7 is a discussion on the choice of item-ids. In particular, this looks at the advantages and disadvantages of using meaningful data in an item-id. As part of this, it explores how to achieve the same ends using a sequential number as the item-id.

Chapter 8 contains some brief concluding comments.

Chapter 9 provides answers to some common questions.

Chapter 10 provides a quick reference to some of the common commands in *OpenQM*, and for *QMQuery*.

Chapter 11 contains detailed installation instructions for both *OpenQM* and *AccuTerm*.

0.5 Thanks

Thanks have to go to Ladybridge Systems. Without them, there would be no *OpenQM*.

Ladybridge also set up the *OpenQM* Google group, and regularly answer questions from users in that forum. Some of that information has found its way into this book, so in one sense, all of the contributors to the *OpenQM* Google group have indirectly contributed to this book.

Other information has been gleaned from the “Pick and MultiValue Databases” google group (and it’s predecessor, the comp.databases.pick newsgroup). Those resources cover all multi-value databases, and not just *OpenQM*.

This book uses icons downloaded from: http://icons.mysitemyway.com

0.6 Trademarks and Copyright

OpenQM is copyright to Ladybridge Systems. This copyright covers all aspects of *OpenQM* including source code, executable code, and documentation.

AccuTerm is copyright to AccuSoft Enterprises.

D3 and *mvBASE* are trademarks of Tiger Logic Corporation.

UniData, *UniVerse*, and *Wintegrate* are trademarks of Rocket Software.

Windows, Access, and Excel are registered trademarks of Microsoft Corporation.

0.7 Copyright of this Publication

This book is copyright to Rush Flat Consulting (2008-2013).

However, this book may be freely copied and distributed provided that the copyright to Rush Flat Consulting remains in place.

Similarly, portions of this book may be freely quoted provided that Rush Flat Consulting is acknowledged as the source of the quoted material.

0.8 Warning and Disclaimer

Every effort has been made to make this book as complete and accurate as possible, but no warrant or fitness is implied. The information provided is on an “as-is” basis. The author shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

Table of Contents

1 Introduction

1.1 What Is *OpenQM?*

OpenQM is a database environment. Specifically, it is one of a number of database products known as **multi-value databases**. Other multi-value databases and their vendors are listed in the table below:

Database	***Vendor***
Caché[2]	Intersystems
D3	Tiger Logic
jBase	Jbase International
mvBASE	Tiger Logic
mvEnterprise	Tiger Logic
OpenInsight	Revelation
OpenQM	Ladybridge Systems
Reality	Northgate Information Systems
UniData	Rocket Software
UniVerse	Rocket Software

There are also a number of superseded products still in use. However, the above list represents most of the currently available multi-value databases.

2 Caché is not strictly a multi-value database. Rather it is an object database which has gained multi-value extensions. It can thus be used using a query language similar to *QMQuery*, and a programming language similar to *QMBasic*.

1.1.1 What is a multi-value database?

Firstly, multi-value databases have been designed from the ground up as multi-user databases. While this book really only covers the use of *OpenQM* as a personal database, it is quite capable of handling hundreds of users simultaneously.

Multi-value databases have a number of characteristics that make them different from relational databases such as *mySQL*, *SQL Server*, or *Oracle*. These are:

- their data model does not (have to) conform to relational rules
- data is loosely typed
- data is stored in literal format, in variable length records
- data is stored in hashed files
- they come with their own in-built programming language
- they come with their own in-built reporting language allowing fully formatted reports to be generated from the data

Modern multi-value databases also provide interfaces to external programming languages, socket connections, and the ability to interact with the host operating system[3].

What do these things actually mean?

Non-conformity to relational rules

Take the example of a typical invoice. In a relational database, invoice data is stored in two tables. The first table represents the invoice header and contains the invoice number, date, and customer reference (amongst other things). The second table contains the line details of the invoice. This second table is linked back to the invoice header by way of the invoice number. This structure is shown below:

These tables show the data for three invoices. The first invoice has one line item, the second has two line items, and the third has three line items. The data in the second table (the line items) can be related back to the correct customer through the invoice number.

Invoice number	*Date*	*Customer number*
12345	24 Apr 2007	9854
12346	24 Apr 2007	6234
12347	25 Apr 2007	4921

3 This was not always the case. Multi-value databases originated on mini-computers, and used their own operating system. It was not until the 1980s and 1990s when multi-value databases migrated to Unix (and later, Windows) that interaction with an external operating system was required.

Invoice detail number	*Invoice number*	*Product ID*	*Quantiy*	*Price*
671245	12345	9854	2	15.00
671246	12346	6234	1	32.50
671247	12346	4921	1	23.90
671248	12347	5651	3	12.50
671249	12347	5694	2	3.50
671250	12347	6234	5	32.50

In a multi-value database, all this data could be contained in just one table (referred to as a file). The structure of the multi-value invoice file is shown below:

Invoice number	*Date*	*Customer number*	*Product ID*	*Quantity*	*Price*
12345	24 Apr 2007	9854	9854	2	15.00
12346	24 Apr 2007	6234	6234 4921	1 1	32.50 23.90
12347	25 Apr 2007	4921	5651 5694 6234	3 2 5	12.50 3.50 32.50

If you examine these two structures, you will see that the single multi-value structure contains all the same information as the two structures in the relational database. The difference is that each of the Product ID, Quantity, and Price fields have multiple entries in the field.

Note that it is much quicker to read an invoice from a multi-value database than from a relational database. Using invoice 12347 as an example, a multi-value database can read this invoice with a single disk read[4]. In comparison, a relational database would use 4 disk reads for the data, plus a few more for reading indices.

Loose data typing

In many databases, fields are defined as being of a specific data type, and the database will not allow data of any other type to be stored in that field. Multi-value databases do not follow this pattern.

Firstly, database fields do not actually need to be formally defined. Of course, well structured databases do have field definitions, but even then, the definitions are descriptive rather than prescriptive.

Secondly, even if the field definition says the data is of a certain type, the database itself places no restrictions on the type of data actually entered into the field. Therefore, string data may be entered into a numeric data field and vice-versa with no objections[5] from the database[6].

4 This assumes that the application knows the ID (primary key) of the invoice. If this is known, the database will calculate the location of the item, and read it in a single read.
5 No objections from a storage perspective. However, you may have difficulties processing the data using the programming language if you mix data types.
6 Data type integrity could be enforced by using triggers on the data file to test the data before it was written to disk.

Thirdly, in the programming language, the typing of variables is not required, and may change within the program. For example:

```
Temp = 0
...
...
Temp = 'Q'
```

These characteristics mean that multi-value databases are flexible, and make it easy to accomplish certain tasks. The flip side of this coin is that it is easy to end up with a database structure that is (a) not defined, (b) only partly defined, or (c) incorrectly defined. Likewise, it is possible to end up with unexpected data types in the database fields.[7]

Data storage

As noted above, many databases get you to define the field types. One of the reasons they do this is so that numbers can be stored as a numeric data type. What this means is that the integer 123 can be stored as a single byte representation. On the other hand, those databases will typically reserve either 4 or 8 bytes for an integer value, even though only one byte is being used.

Multi-value databases store data as literal strings. Therefore, 123 is stored as the string "123", and the field length is 3 characters. If the number changed to "12345" then the field will be expanded to 5 characters.

These variable length fields are achieved by using special characters to delimit the fields in the record. The database counts along the delimiters to find the requested field, value, or sub-value.

Consider an address database. The name and address data fields will typically be 30 characters long in a traditional database. With one name field and four address lines, the record will consume 150 bytes, regardless of much data is actually stored in that space. A multi-value database will use as many bytes as are entered – plus the delimiter characters (5 in this case). If a line is longer than 30 characters, then the multi-value database is able to store the extra characters (where a traditional database cannot) – although you will then have an issue of how to print them if you are restricted to 30 characters on an address label. If a line isn't used, then the multi-value database will only store a delimiter character.

When storing strings like an address, a multi-value database may only require a third the disk space that a relational database would use because it doesn't need to reserve space for fields – it simply uses space when it is required. When storing numbers, it is more evenly balanced, with the multi-value database using more space for large numbers and less for small numbers.

Hashed files

A hashed file consists of a series of groups or buckets. Records are assigned to a group using a pseudo-random method based on the record ID. This is how a multi-value database can find a record quickly if the ID is known. The process is:

- The ID is hashed to form a large number.

7 The phrase "Give them enough rope" is often used in discussions regarding multi-value databases. The flexibility and lack of enforcement of rules make it easy to create poorly structured databases, maintained by poorly structured code. Basically, the integrity of the system is in the hands of the developer(s).

- The large number is divided by the modulo (number of groups) of the file. The remainder from the division is the group number.
- The entire group is read from disk, and the record is found by searching through the group.

The combination of hashed files and storage of records as variable length strings make for a highly efficient storage and retrieval system. Part 2 of *"Getting Started in OpenQM"* covers the theory and practice of hashed files in greater depth.

Inbuilt programming language

Many databases (such as mySQL) do not provide a programming language to access or manipulate the data in the database. Rather, they provide an 'Application Programmer Interface' (API) which allows an external programming language (such as Perl, Python, PHP, or Visual Basic) to access the database. This means that programmers can use a language with which they are familiar – if there is an API for that language, and that the database provider does not have to put resources into developing and maintaining a programming language.

Multi-value databases have a different approach. They tend to provide an entire database environment including a programming language and reporting facilities. This is particularly useful for exploiting the multi-dimensional nature of the database.

The actual language provided is a dialect of BASIC. This has been extended to be used with multi-value data in a multi-user environment. It is simple to learn, but contains powerful data and string handling capabilities.

Modern multi-value databases often also incorporate one or more API's for use with external languages. Typical API's are for C, Visual Basic, and/or ODBC.

Inbuilt reporting language

Multi-value databases incorporate a combined query and reporting language that allows you to:

- report on data contained in one or more files
- select records to be reported on (multiple selections)
- sort the data by multiple sort criteria
- break the data into groups
- create data columns derived from other data
- calculate totals, averages, and percentages
- format the data to display in a specified format
- format the report with headings, footings and page breaks
- and more

This is all achieved through a sentence based query language. For example:

```
SORT INVOICES WITH DATE GE "01-04-2013" AND LE "31-04-2013" BY CUSTNO BREAK-ON CUSTNAME "'UV'"
ENUMERATE INVNO INVDATE TOTAL AMOUNT HEADING "'DGC'Invoices for April 2013'G'Page 'PL'" FOOTING
"Monthly invoice report" ID.SUP NO.GRAND.TOTAL LPTR
```

This would select all invoices for April 2013, sort them into Customer No order, and produce a report showing the customer name, and the invoice number, date and amount of each invoice. After all invoices for each customer have been displayed, totalled and enumerated fields will be underlined and the count of invoices, and the total amount of the invoices for that customer will be displayed. Pages have a defined heading and footing, and the report will be sent to the printer.

Summary

Overall, multi-value databases are flexible and easy-to-use. The combination of an easy to use programming language and reporting/query language that allows flexible reporting is a powerful combination.

Interfaces to external languages allow multi-value databases to be incorporated seamlessly into a Windows (or linux) environment, although this means that you forgo most of the inbuilt reporting capabilities.

1.1.2 What makes OpenQM different?

Given that there are a number of multi-value databases available, and that there is a high degree of compatibility between each of them, it is fair to ask what makes *OpenQM* different. Some of the key differences include:

- Availability:
  - A 'Personal' version of *OpenQM* is available at no charge and without registration
  - An open-source version is available for use on Linux
  - The commercial versions of *OpenQM* costs substantially less than other multi-value databases
  - No annual maintenance fee is required (unlike other multi-value databases)
  - Purchase of 10 years of upgrades is just 25 per cent of the base cost
- Bundled software:
  - An *AccuTerm*[8] licence is bundled with *OpenQM*
  - API's to C, Visual Basic, and Power Basic are included
  - A simple SQL interface is included
- Choice of Operating System (32 and 64 bit):
  - Windows
  - Linux
  - Macintosh OS X
  - FreeBSD
  - AIX
  - Raspberry Pi
  - Solaris
  - PDA

8 *AccuTerm* combines terminal emulation, file transfer facilities, a GUI development environment, and tools for automating tasks between Windows and multi-value environments.

- *OpenQM* is being actively developed:
  - Product enhancements and bug fixes come out on a regular basis
  - The development/enhancement process is responsive to user requests
- Enhancements to generic multi-value features:
  - Query/Reporting language
    - Report panning when reports are wider than the screen size
    - Report scrolling to allow previous pages to be viewed (again)
    - Report output to a delimited file
    - Report styles allow use of colour and bolding in screen reports
  - *QMBasic* programming language
    - Local variables in subroutines
    - Object-oriented programming
    - Exception handling

This just scratches the surface of the enhancements within *OpenQM*.

Overall, the combination of a powerful product, bundled software, active development, and a low price make *OpenQM* a compelling offering in the multi-value database market.

OpenQM makes it easy to try the product at no cost. A personal version is available for Windows, or the open-source version is available for Linux. A personal version of *AccuTerm* is available to complete the package. See below for further details on these products.

The commercial version of *OpenQM* costs UK£65 (from Ladybridge Systems) or US$101.27 per user. This includes one year of free product upgrades. This upgrade period can be extended to 10 years at a cost of 25 per cent of the base product cost – therefore buying *OpenQM* with 10 years of free updates costs just £81.25 or US$126.59 per user.

OpenQM allows device licensing (where a user may have multiple connections to the database without consuming additional licences) at additional cost. This cost is substantially less than the cost of purchasing additional licences.

Likewise, “standby” licences can be purchased at reduced cost.

1.2 Multi-Value Terminology

Multi-value databases have their own terminology. This section provides a quick coverage of that terminology and places it in a relational database and general computing framework.

Accounts

Information in multi-value databases is organised into accounts. An account is loosely analogous to a database in relational terms.

In *OpenQM*, an account is implemented as a folder or directory, with the database files implemented as folders or sub-directories within that account.

When logging on to *OpenQM*, a user has to log on to an account. They then have direct access to all the database files in that account. Data from files in other accounts can be accessed using a file pointer. This makes the data appear as if it is local to the current account. Users can log between accounts at will (security allowing).

Overall, an account is simply a way to group related database files.

Users

A user must be registered to use *OpenQM*. *OpenQM* largely uses the operating system authentication to validate a user name, although it is possible to add further restrictions on users once they have entered the *OpenQM* environment.

It is important to note the distinction between users and accounts. Accounts may be used by many users (simultaneously), and individual users may use multiple accounts. Individual users may have multiple concurrent sessions, in one or more accounts. On the other hand, some users will only use a single account, and may be restricted to a single account even when multiple accounts are available.

User administration is not considered in this introduction to *OpenQM*.

Database files

A database file is analogous to a table in a relational database. Whereas a relational database is made up of multiple tables, a multi-value database is made up of multiple files.

A multi-value database file normally consists of two parts – a dictionary, and a data portion – although each part can exist independently of the other, and a dictionary may be associated with multiple data portions.

A file dictionary exists to provide definitions of the data in the data file(s) *for reporting purposes*. This qualification is important because:

- the dictionary does not define or restrict the data in the manner of a relational database. The dictionary is purely descriptive
- the description is not enforced (by the database) and does not have to be correct!
- a data element can be described in multiple ways. For example, a numeric field may have three definitions to show its value in units, thousands, and millions
- the primary use of the definitions contained in the dictionary is for reporting purposes using the *QMQuery* reporting language.

In *OpenQM*, database files are implemented as operating system folders or subdirectories. Dictionary and data portions of the file each have their own folder – the data portion will take its folder name from the filename you specify in *OpenQM*, while the dictionary folder will have '.DIC' added to the *OpenQM* filename.

For example, assume that we are in an account named TEST which has been created in folder D:\QM\TEST. If we create a normal file named TESTFILE, *OpenQM* creates two sub-folders beneath the TEST folder named TESTFILE and TESTFILE.DIC.

In normal files, each of the dictionary and data folders contains two files named '%0' and '%1'[9]. These are hashed files, where *OpenQM* maintains the filing structure and indices.

9 In earlier versions (including the GPL version), these files were named '~0' and '~1'. The file names were changed because some Windows cleaner programs deleted the files on the assumption that they were temporary files.

While these files are visible at the operating system level (e.g. through a file manager), the contents of these files should not be edited using any operating system utilities.

In the case where a dictionary is associated with multiple data portions, the 'data' folder holds a sub-folder for each data portion.

Program files (or directory files)

Traditionally, multi-value databases only used hashed files as described above. However, more modern implementations such as *OpenQM* recognise that hashed files are inefficient storage mechanisms for items such as programs, and according have implemented the use of directory files for these types of items.

A directory file still consists of dictionary and data portions. However, the data portion is simply an operating system folder. The dictionary portion continues as a hashed file as described above.

Items contained in a directory file can be accessed and edited directed from the operating system environment as well as from within the *OpenQM* environment.

While directory files can be used to hold "normal" structured database data, this is not recommended. Data retrieval from directory files is much less efficient than from hashed files.

Typical uses of directory files are to store programs, images, or PDF files.

The VOC file

Multi-value databases provide a command driven environment. Accordingly, the database must be able to understand the commands issued by a user. It does this by storing command definitions in a special file known as the VOC or vocabulary file[10].

Each account has its own VOC file. This means that System Administrators can restrict actions within certain accounts by removing selected keyword definitions, or that commands can be added to the VOC that are relevant to that account.

From version 2.6-9, *OpenQM* has added a "private VOC". The idea of a private VOC is that individual users can add paragraphs and sentences to their own environment that do not impact on other users of the same account.

Records (or items)

The naming of elements contained within the database files varies according to the particular implementation. *OpenQM* uses the terminology of records, fields, values, and sub-values. PICK databases have traditionally used the terminology of items, attributes, values, and sub-values. Either terminology is generally acceptable in any of the multi-value environments, even though each environment has its own preferred terminology.

A multi-value record is loosely analogous to a record in a relational database. The key difference is that a multi-value record can be equivalent to a *group of records* in a relational environment. If you refer back to Section 1.1.1 (What is a multi-value database?), you can see that invoice number 12347 holds the data equivalent of several relational database records. This is why an item can be thought of as a group of records.

10 In PICK-style databases, this file is known as the Master Dictionary, M/DICT, or MD.

Records also refer to programs. For normal programming languages, a program is an individual file held within a folder. In the multi-value world, a program is a record within a file (a programs file). In *OpenQM*, you can actually have both views of programs, because if the programs are held in a directory file, then *OpenQM* will see them as records in a file, but the file manager will see them as files within a folder.

Fields (or attributes), values, and sub-values

Multi-value fields (attributes) are loosely equivalent to fields in a relational database. However, attributes can be divided up into several values, which in turn can be divided into several sub-values. Relational databases have no equivalent of values and sub-values, and need to use multiple tables to store the equivalent data structures.

Section 1.1.1 has already shown how the 'Product ID' field on the invoice can hold several data values. This is the core principle of the multi-value database.

Examples of the use of sub-values are less intuitive. Consider sub-values as a way of storing a bit more related data.

1.3 A Note On Capitalisation

Multi-value databases originated at a time when data entry was largely restricted to upper case characters. These early databases only recognised commands entered in upper case. Likewise, all programming was required to be in upper case.

As time went by and usage of mixed case became prevalent, different databases adapted in different ways. *OpenQM* attempts to be reasonably case insensitive:

- The programming language is not case sensitive.
- When a command is entered at the keyboard, or *OpenQM* searches for dictionary items, it firstly attempts to find the command or dictionary item in the case as typed. If the command or dictionary item is not found, then the word is converted to upper case and the search repeated. If it is still not found, then any hyphens in the name are replaced by dots and the search repeated.

To illustrate the significance of the second point, consider that you have created a dictionary item named 'myDictItem'. If you subsequently reference that dictionary item in any command or query, then you will need to type it exactly as you have named it. However, if you name the dictionary item in upper case (i.e. 'MYDICTITEM'), then *OpenQM* will find the dictionary item *whatever case you use in your query command.*

For this reason, it is recommended that you name all VOC items, dictionary items, and programs in UPPER CASE.

1.4 Command Variations

OpenQM accepts a number of command variations. These variations allow for:

- North American word spelling as well as British spelling (e.g. CATALOG is accepted as being the same command as CATALOGUE)

- compatibility with other multi-value databases (e.g. ID-SUPP is accepted as a synonym for ID.SUP).

Commands in this book are mostly expressed in the *OpenQM* “native” format – with the exception that OPTION PICK is enabled for the *QMQuery* chapters. The differences are highlighted where necessary.

1.5 Conventions In This book

OpenQM is a command-driven environment. User commands are shown in bold **Bitstream Vera Sans Mono** typeface, while the responses from *OpenQM* are not bolded:

```
SORT XRATES DATE USD GBP AUD ID.SUP
Date........   US Dollar   UK Pound   Aus Dollar
 05 JAN 2010      0.7344     0.4564       0.8046
 06 JAN 2010      0.7343     0.4591       0.8056
 07 JAN 2010      0.7378     0.4606       0.8020
 08 JAN 2010      0.7325     0.4597       0.7980
```

Where the syntax of commands is shown, curly brackets[11] denote optional components, while a pipe symbol denotes that only one of the separated options should be used:

```
SEARCH {DICT} filename {ALL.MATCH | NO.MATCH} {NO.CASE}
```

OpenQM keywords within the text will be in Bitstream Vera Sans typeface.

Where a user has to supply a list name or file name, this is usually in *italicised lower case*.

The following symbols may appear in the margin:

This indicates useful information.

Take note of this.

This is the WRONG WAY to do this – even if it looks like it is working.

This is wrong.

This is the right way to do this.

We are just going off to explore a related area. We’ll be back to the main thread soon.

This is a summary of points.

11 In many computer manuals, optional components are often shown in square brackets. However, as square brackets are used within the *OpenQM* query language and programming language, curly brackets are used both here and in the documentation to avoid confusion.

2 Installation and Configuration

2.1 Versions of OpenQM

OpenQM is available for a number of platforms, and with 3 different types of licence. Not all licence types are available for all platforms. The following table shows the combinations of platform and licensing available:

Platform	*Personal*	*Commercial*	*GPL*
Windows	✓	✓	
Linux	✓	✓	✓
FreeBSD	✓	✓	
Mac OS X	✓	✓	
PDA	✓	✓	
USB	✓	✓	

The essence of this licensing is that:

- commercial and personal licenses are available on all platforms
- the GPL license is only available on Linux

2.1.1 Commercial and personal versions

The software is identical for the commercial and personal versions. However, use of the personal license imposes some restrictions on use. These include:

- Personal licenses may only be used for personal or educational purposes. This can include development of software for commercial purposes, but the *OpenQM* licence must be upgraded to a commercial licence when the software is sold

- Personal licenses may not be used to run commercial software
- A personal license restricts the software to one interactive user, one *QMClient* user, and one phantom user.
- Remote server access is not enabled in the personal version
- The personal version cannot write items to or delete items from files beyond about 500 KB in size
- There is no free support available for the personal license
- The personal license does not come bundled with *AccuTerm*. However, *AccuTerm* also has a personal license (also subject to restrictions) which can be used with the personal version of *OpenQM*.

In contrast, the commercial licence allows you to use the software for whatever purposes you choose, and removes the restrictions on file sizes, remote server access, and bundled software.

[Note that the *AccuTerm* licence bundled with *OpenQM* is restricted to usage with the *OpenQM* commercial product. This licence does not permit usage of *AccuTerm* with the *OpenQM* personal licence, or with any other multi-value database].

You can upgrade a personal licence to a commercial licence without any reloading of software. You will simply need to pay for a licence, and then enter the licence details into *OpenQM* to remove the restrictions imposed by the personal licence.

To download the commercial or personal versions of *OpenQM*, go to www.openqm.com and click on the 'Download' link. No registration is required.

2.2 GPL version

The GPL version of *OpenQM* is only available on Linux. However, under the terms of the GPL, you may port this to whatever other platform you wish.

The GPL version remains at version 2.6-6. This was released in 2007, and is now well behind the commercial version in terms of features. Nevertheless, this was a commercial quality database at the time it was released, and remains fully functional today.

There is no automated installer for the GPL version, and some familiarity with the Linux command-line will aid in the installation. However, there are several sets of instructions available to assist you:

- For Ubuntu-based distributions, try: www.rushflat.co.nz
- For Red Hat-based distributions, try: www.geneb.org/qm/fedora_notes.txt
- Other notes at: www.billabong-services.co.uk/anji/

To download the GPL version of *OpenQM*, go to www.openqm.com and click on the link contained in the paragraph describing the open-source release on the home page. Registration is required to download the open-source release.

There is a fork of the *OpenQM* GPL database called ScarletDME. There are only minor differences between *OpenQM* and ScarletDME. You can download ScarletDME from: https://github.com/geneb/ScarletDME

2.3 Installing OpenQM

2.3.1 Installation

Full details of installing *OpenQM* on a variety of platforms are given in the Appendix.

Installing on Windows is straightforward and similar to installing any other Windows product – follow the prompts, click the appropriate buttons, and fill in the licence details at the end of the installation.

Installing the commercial (or personal) version on linux is also straightforward – set the download file to be executable, execute it, and fill in the licence details at the end of the installation.

Installing the GPL version is much more involved, and will require some familiarity with the Linux command-line. However, the instructions (referred to above) are reasonably detailed, and most people should be able to complete the task.

Before you start, consider where you will store your *OpenQM* data. By default, the *OpenQM* system will be installed into a folder named `C:\QMSYS` (or `/usr/qmsys` on Linux). This can be changed if you wish.

More importantly, consider where you will keep your user data. Most examples in this book use a path of `E:\QM\...` but any path on your computer system can be used. The data does not have to be grouped with the `QMSYS` folder. Nor do all the *OpenQM* account folders have to be grouped together.

2.3.2 Configuration

You can run *OpenQM* as it is without any further configuration. However, I would suggest the following settings be changed in or added to the `qmconfig` file:

```
ERRLOG=512
GRPSIZE=2
SORTMEM=4096
```

These settings:

- enable error logging and set the maximum log size to 512 kB (default: error logging not enabled, or log size set to 10 kB).
- set the default group size for new files to 2 kB (default: 1 kB)
- set the sort memory to 4096 kB (default: 1024 kB)

Look through the settings outlined in the help file under ‘Configuration parameters’ to see what other parameters may be changed.

To see the current values of the configurable parameters, type:

```
CONFIG
```

The `qmconfig` file will be in one of the following locations:

- the QMSYS folder
    - typically C:\QMSYS on Windows
    - /usr/qmsys on Linux
- the /etc folder (GPL)

Simply use a text editor to edit the file. Do not change anything in the LICENCE line (if it exists). Save the file once you have made your changes. The changes will come into effect once you restart *OpenQM*. (Just restart your computer).

Arguably, the GRPSIZE parameter should be set to 4. The documentation notes that: “For best performance, it (the GRPSIZE parameter) should be a multiple of the operating system disk block size”. Most modern Windows and Linux operating systems use a 4 kB block size (until disk size gets to 16TB anyway), so this would suggest that a GRPSIZE of 4 will offer the best performance.

2.4 AccuTerm

2.4.1 AccuTerm licences

When you purchase a commercial licence for *OpenQM*, you get an authorisation code for *AccuTerm* with your licence details.

In most instances[12], you need to download the *AccuTerm* terminal emulator software, from www.asent.com. Click on the ‘*AccuTerm* Standard’ product image on the left of the screen, and then download the ‘*AccuTerm* 7 30 day evaluation version’.

If you are using the *OpenQM* personal licence, then you can either get an *AccuTerm* personal licence, or purchase a full *AccuTerm* licence. However, a full *AccuTerm* licence costs more than an *OpenQM* licence, so you would be better off buying *OpenQM* in the first place. On the other hand, if you already have an *AccuTerm* licence, then you can continue to use that.

The principal technical restriction applying to the *AccuTerm* personal licence is that it will only connect to a database on your local computer (i.e. localhost or 127.0.0.1). It is also restricted to non-commercial use.

To get a personal licence for *AccuTerm*, click on the ‘*AccuTerm* personal’ product image on the left of the screen, and then fill in the form requesting an authorisation key. Once you receive this, you will be able to enter this authorisation key either during the install process, or from the *AccuTerm* help menu.

Using *AccuTerm* with the GPL version of *OpenQM* is more tricky. You will either need a full licence for *AccuTerm* so that you can use *AccuTerm* across your network connection; or if you need to use the personal version, you could try installing *AccuTerm* on your Linux machine using wine. There are reports of *AccuTerm* running OK under wine and CrossOver Office, but you need to either use a different font within *AccuTerm*, or install the *AccuTerm* fonts into wine (search http://forum.asent.com/default.asp for details).

12 If you are using a USB installation of *OpenQM*, and you have downloaded the USB demonstration file, then you do not need to download *AccuTerm* – it is already included in the demonstration file.

2.4.2 Installing AccuTerm

Full details of installing *AccuTerm* are given in the Appendix. In general terms, *AccuTerm* installs just like any other Windows product. However, to utilise many of *AccuTerm*'s features, you need to install the multi-value host programs. See the Appendix for details.

2.5 Configuring OpenQM and AccuTerm

At this stage, we have the database software installed, and the terminal emulation software installed, but they don't know anything about each other. So we need to configure both software components so they match each other.

2.5.1 Usernames and passwords

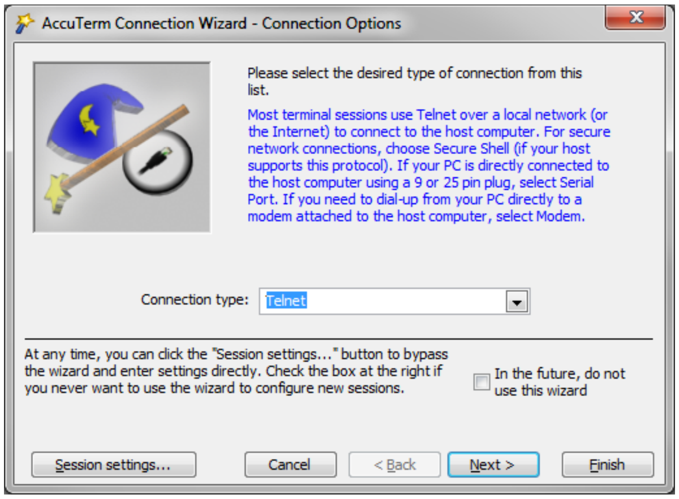
OpenQM is a multi-user database, intended to be operated in a server environment where there are multiple users. As part of normal security practice, those users are expected to have usernames and passwords.

OpenQM uses the host operating system to do user authentication. While such authentication may allow null passwords, this doesn't seem to work well with *OpenQM*.

Therefore, if you do not already have a password on your system, do it now. This can be set using the 'User Accounts' applet within the Windows Control Panel.

2.5.2 Connecting AccuTerm to OpenQM

The first step is to connect *AccuTerm* to *OpenQM*. Start *AccuTerm* (from either the desktop icon or the Start menu). The following screen should appear (over the *AccuTerm* window):


AccuTerm Connection Wizard - Connection Options

Please select the desired type of connection from this list.

Most terminal sessions use Telnet over a local network (or the Internet) to connect to the host computer. For secure network connections, choose Secure Shell (if your host supports this protocol). If your PC is directly connected to the host computer using a 9 or 25 pin plug, select Serial Port. If you need to dial-up from your PC directly to a modem attached to the host computer, select Modem.

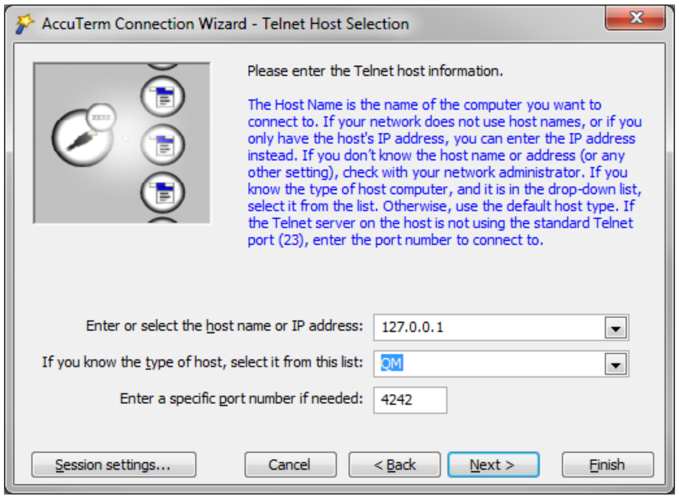
Connection type: Telnet

At any time, you can click the "Session settings..." button to bypass the wizard and enter settings directly. Check the box at the right if you never want to use the wizard to configure new sessions.

☐ In the future, do not use this wizard

If the 'Connection Wizard' screen does not start automatically, click on the 'New session' icon at the extreme left of the toolbar, or choose 'File | New' from the menu.

If you installed the personal version of *AccuTerm*, the only connection type available will be 'Telnet'. Additional options are available if a commercial version of *AccuTerm* is being used. These are: 'Serial port', 'Modem', and 'Secure shell'. We will continue using 'Telnet'. Click on 'Next'.


AccuTerm Connection Wizard - Telnet Host Selection

Please enter the Telnet host information.

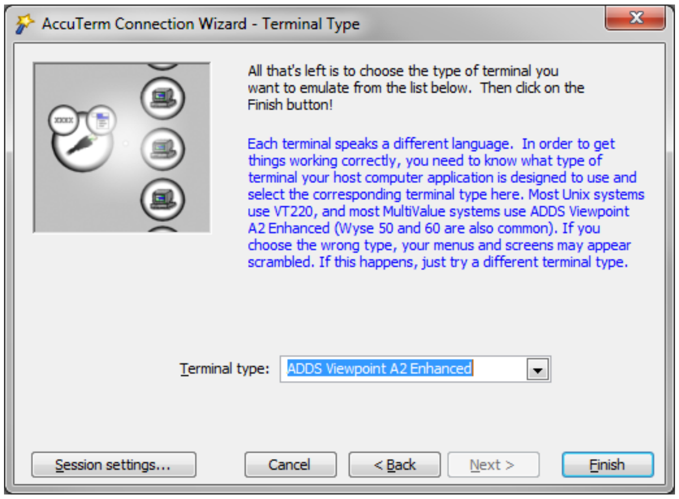
The Host Name is the name of the computer you want to connect to. If your network does not use host names, or if you only have the host's IP address, you can enter the IP address instead. If you don't know the host name or address (or any other setting), check with your network administrator. If you know the type of host computer, and it is in the drop-down list, select it from the list. Otherwise, use the default host type. If the Telnet server on the host is not using the standard Telnet port (23), enter the port number to connect to.

Enter or select the host name or IP address: 127.0.0.1

If you know the type of host, select it from this list: QM

Enter a specific port number if needed: 4242

Enter the host name of the server where you installed *OpenQM* (localhost or 127.0.0.1 if you installed on the same machine as *AccuTerm*). Select 'QM' in the type of host drop down list, and enter the port number as '4242'[13]. Click on 'Next'.


AccuTerm Connection Wizard - Terminal Type

All that's left is to choose the type of terminal you want to emulate from the list below. Then click on the Finish button!

Each terminal speaks a different language. In order to get things working correctly, you need to know what type of terminal your host computer application is designed to use and select the corresponding terminal type here. Most Unix systems use VT220, and most MultiValue systems use ADDS Viewpoint A2 Enhanced (Wyse 50 and 60 are also common). If you choose the wrong type, your menus and screens may appear scrambled. If this happens, just try a different terminal type.

Terminal type: ADDS Viewpoint A2 Enhanced

AccuTerm allows you to choose from a number of different 'terminal types'. This essentially selects the set of internal processing codes that *AccuTerm* responds to. While the choice is up to you, that doesn't help a new user choose between the options. The key point is that the settings in *AccuTerm* must match those in *OpenQM* – and we have yet to define those settings in *OpenQM*.

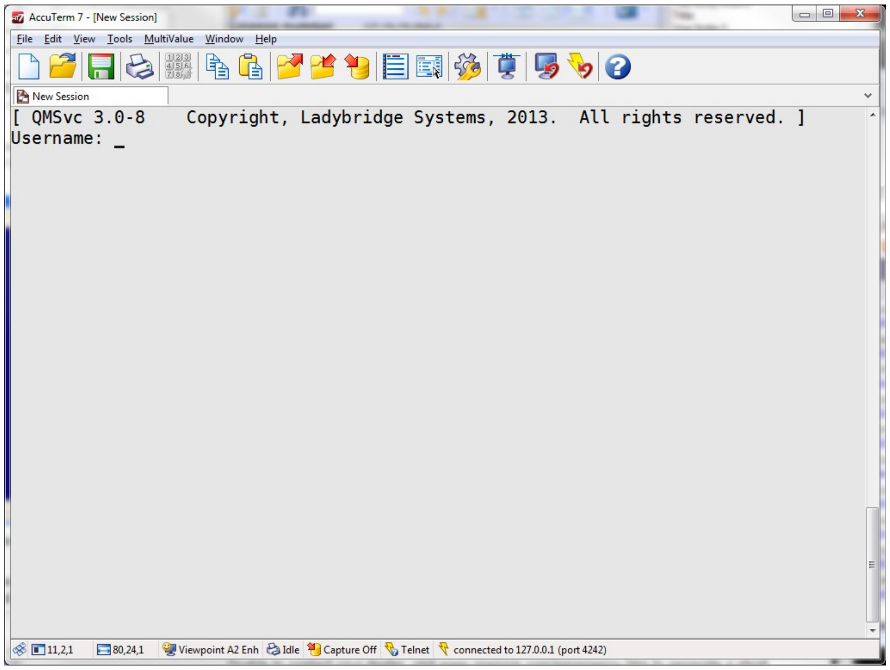
13 Note that the port number is 4240 if you installed the OpenQM on a USB stick using the USB demonstration file. The port is set using the PORT parameter in the qmconfig file. If you have difficulty connecting, check there to see if any non-default port has been set.

OpenQM has a number of *AccuTerm* specific terminal definitions. These are shown in the table below along with the terminal name used by *OpenQM*.

AccuTerm terminal type	OpenQM terminal name
ADDS Viewpoint A2	addsviewpoint-at viewpoint-at
ADDS Viewpoint A2 enhanced	vp-at vpa2-at
ADDS Viewpoint 60	viewpoint60-at
VT100	vt100-at
VT220	vt220-at
VT320	vt320-at
VT420	vt420-at
Pick PC Monitor	pcmon-at
Wyse 50	wyse50-at
Wyse 60	wyse60-at

You should select one of these. Note that the *AccuTerm* screen effectively recommends using VT220 if the *OpenQM* server is on a Unix (Linux) host, or ADDS Viewpoint A2 Enhanced otherwise.

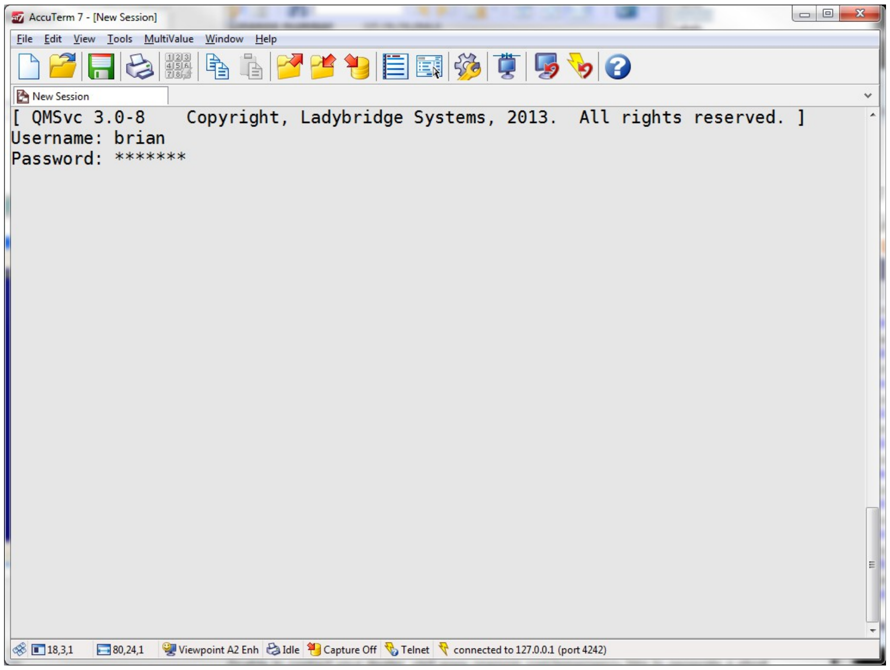
The screenshot above shows the 'ADDS Viewpoint A2 Enhanced' terminal being selected. Once you have made your selection, click on 'Finish'. The following screen should now appear:


AccuTerm 7 - [New Session]
File Edit View Tools MultiValue Window Help
New Session
11,2,1
80,24,1
Viewpoint A2 Enh
Idle
Capture Off
Telnet
connected to 127.0.0.1 (port 4242)

```
[ QMSvc 3.0-8    Copyright, Ladybridge Systems, 2013.  All rights reserved. ]
Username: _
```

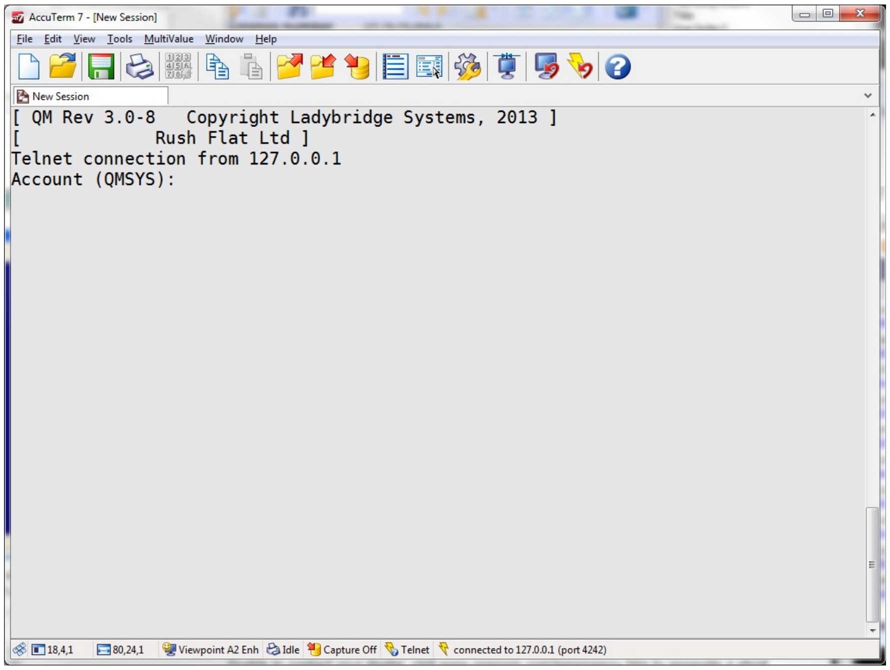
If you don't see this message, make sure that firewall software on the server computer is not blocking the attempts by *AccuTerm* to access *OpenQM*.

Type your username (in Windows XP Home installations, the default user name may be 'owner'), press enter, and then enter your password when prompted.


AccuTerm 7 - [New Session]
File Edit View Tools MultiValue Window Help
New Session
18,3,1 80,24,1 Viewpoint A2 Enh Idle Capture Off Telnet connected to 127.0.0.1 (port 4242)

```
[ QMSvc 3.0-8    Copyright, Ladybridge Systems, 2013.  All rights reserved. ]
Username: brian
Password: *******
```

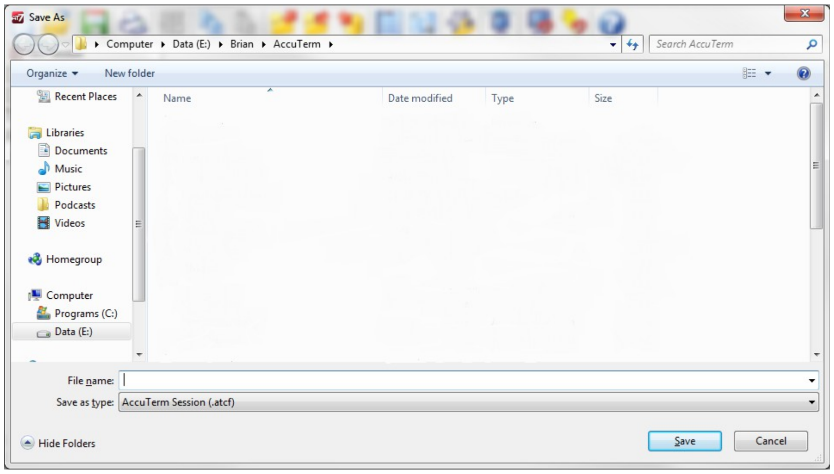
You will now see the following message:


AccuTerm 7 - [New Session]
File Edit View Tools MultiValue Window Help
New Session
18,4,1 80,24,1 Viewpoint A2 Enh Idle Capture Off Telnet connected to 127.0.0.1 (port 4242)

```
[ QM Rev 3.0-8   Copyright Ladybridge Systems, 2013 ]
[              Rush Flat Ltd ]
Telnet connection from 127.0.0.1
Account (QMSYS):
```

Press enter to accept the default account – you don't have any other accounts created, so you don't have a choice at this stage. You are now left at a command prompt, represented by a colon (:).

Before doing anything else, we need to save your *AccuTerm* setup, so that you don't have to redefine the terminal every time you want to use *OpenQM*. Click on 'File' and choose 'Save' or Save as', or click on the 'Save session settings' icon. The following screen will appear:



File name:
Save as type: AccuTerm Session (.atcf)

The folder shown in the address bar can be specified in the *AccuTerm* 'Settings' screen (under 'Files & Folders'). Type a name in the 'File name' edit box ('QM' or '*OpenQM*'), and then click on 'Save'.

Your basic terminal definition has now been saved. You can modify this further later.

For now, we are going to quit from *OpenQM*, then re-start *AccuTerm* to continue with the configuration process.

To quit from *OpenQM*, type in QUIT or OFF at the command prompt. Case is not important for these commands - Quit and quit work just as well.

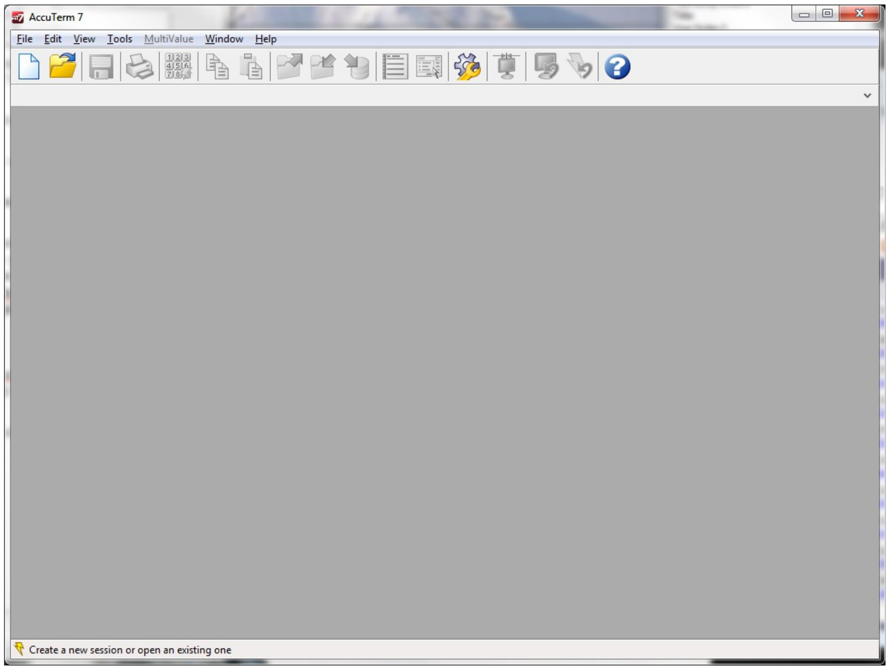
You will get an *AccuTerm* message asking whether you want to 'Reconnect', 'Close', or 'Cancel'. 'Reconnect' will present you with another login prompt; 'Close' will close the session, ,but leave *AccuTerm* running; while 'Cancel' will leave the old session visible, but you will still be logged off from *OpenQM*. Choose 'Close' to close the session. You could also close *AccuTerm* completely.

2.5.3 Initial Configuration of OpenQM and AccuTerm

At this stage, we have established communication between the *AccuTerm* "terminal" and the *OpenQM* server. But they still aren't fully matched to each other, so we need to address that now.

First of all, we need to log back on to *OpenQM*. Start *AccuTerm* (if you closed it earlier), and you should get a screen similar to the one shown on the next page.

To restart the session we saved earlier, click on the 'File' menu, and choose the session you saved earlier from the recently used file list. This will bring up the 'Username' prompt. Enter your username and password. *OpenQM* remembers which account you were last logged on to and will offer the QMSYS account as the default account to log on to. Accept the default, and you will be left at the colon (:) prompt.


AccuTerm 7
File Edit View Tools MultiValue Window Help
Create a new session or open an existing one

Setting the AccuTerm screen size

What now? Let's decide what screen size to use. A default terminal usually has a size of 80 characters wide by 25 lines deep. However, modern terminal emulators such as *AccuTerm* allow virtually any size of screen. *AccuTerm* can display up to 240 characters across the screen, and 240 lines down the screen – but on most screens, you won't be able to read the resulting font size.

AccuTerm also has two “standard” screen sizes – one is “normal”, the other is “extended”. This allows you to switch quickly between the two screen sizes.

Click on ‘Tools’ in the *AccuTerm* menu bar, and then choose ‘Settings’. You can also reach these settings by clicking on the ‘Display settings window’ icon in the toolbar. This will display a window with a tree control on the left, and a settings area on the right. Select ‘Terminal | Screen’ from the tree control. This will display the settings screen as shown on the next page.

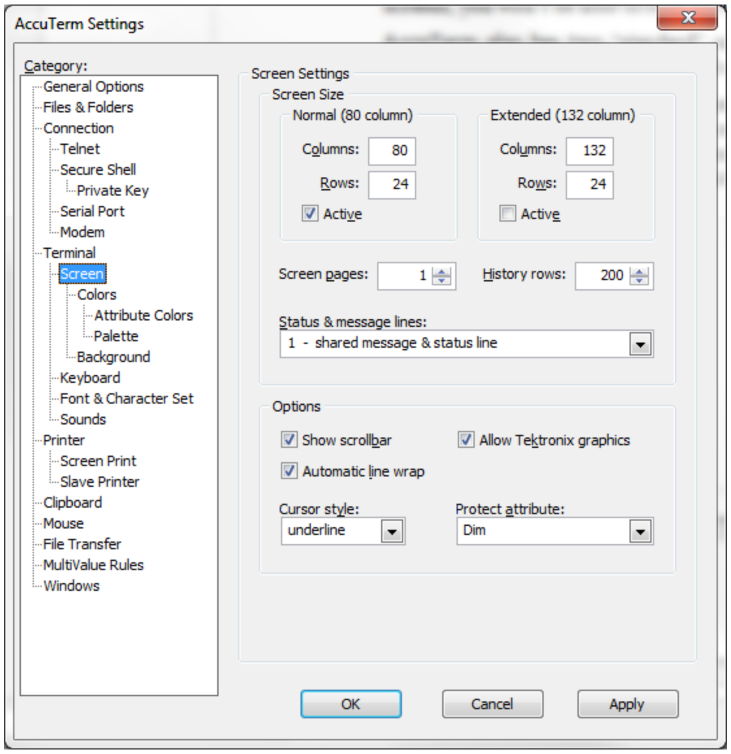
This shows the “normal” screen is set to a size of 80x24, the “extended” screen is set to a size of 132x24, and that the normal screen is active. Other settings show that *AccuTerm* will maintain a history of data sent to the screen of the last 200 lines, and that the cursor will be shown as an underline character.

Try changing these settings to find screen sizes that display sufficient data, while still maintaining font legibility. For example, you could use 132x35 as your “normal” screen, and 160x40 as your “extended” screen.

You may also find it useful to increase the number of history rows. This lets you scroll back to see what you have done. Setting this to 2000 rows should be more than adequate.

Once you have decided on your screen settings, enter them into the *AccuTerm* settings, select which screen you wish to use, and click on ‘Apply’. The current settings should appear in the

AccuTerm status bar at the bottom of the screen. Now click on the 'Save' icon in the toolbar to save the current session settings.



Screen Settings

Screen Size

Normal (80 column): Columns: 80, Rows: 24, ☑ Active

Extended (132 column): Columns: 132, Rows: 24, ☐ Active

Screen pages: 1 History rows: 200

Status & message lines: 1 - shared message & status line

Options

☑ Show scrollbar ☑ Allow Tektronix graphics

☑ Automatic line wrap

Cursor style: underline Protect attribute: Dim

Other settings

The default screen colours are fairly plain. Bring up the 'Settings' window again, and choose 'Colors' under 'Terminal | Screen'. Click on the 'Legacy colors' button, then 'OK', and save the session settings.

This gives yellow text on a blue background. You could click on the 'Advanced Colors & Styles' button to give finer control over the colours, but I would suggest waiting until after we have dealt with report styles in Section 5.6.2).

It is often useful to cut and paste text from one application to another. Let's change the default way that *AccuTerm* does this.

Open the 'Settings' window again, and select 'Clipboard' from the 'Category' tree. Tick the checkbox marked 'Do not send line-end after last (or only) line pasted)', and select 'Ctrl +C / Ctrl +V' in the 'Copy/Paste shortcut dropdown. Click 'OK'.

These changes mean that you can use the Windows standard shortcuts of Ctrl-C and Ctrl-V to cut and paste text. And when you paste the text, it is not automatically executed – you can edit it before pressing enter. If we didn't make this change, *AccuTerm* would add an <enter> at the end of the text you have just pasted – causing it to execute.

Setting OpenQM to match the AccuTerm settings

Now we need to tell *OpenQM* what settings to expect.

From the command prompt, type: `TERM <enter>`

OpenQM should respond something like:

```
TERM
Page width: 80
Page depth: 24
Device    : adds4000-at
```

Page width and depth are set to default "terminal" settings, while the terminal type has been set to ADDS 4000. This terminal type has been set by a negotiation between *OpenQM* and *AccuTerm* to set an appropriate terminal type.

If we look in the *AccuTerm* documentation ('Help | Documentation | User Manual', section 1.6.4), we can see that the ADDS 4000 terminal emulation is appropriate for the Viewpoint A2 Enhanced terminal. However, we need to change the terminal size to match our *AccuTerm* settings.

To change the terminal size:

```
TERM 132,35
```

Note: Some terminals require that the settings you apply with the TERM command be one character less than those specified in the *AccuTerm* settings window[14].

We don't need to change the terminal type, but if we did, we would type:

```
TERM term-id
```

To find the list of valid term types, use a file manager to look in the 'terminfo' folder in the QMSYS account. The term types are grouped into folders according to their first letter.

Term types with a suffix of '-at' have been modified to work well with *AccuTerm*, so you should choose these in preference to those without the '-at' suffix. i.e. choose 'adds4000-at' rather than 'adds4000'.

Typing 'TERM' now should report page width, page depth and terminal type as the values we have just specified.

Clearly, we don't want to have to set these parameters every time we want to use *OpenQM*. Fortunately, *OpenQM* provides a way of running a stored set of commands every time you log on. This is known as the MASTER.LOGIN paragraph located in the VOC file of the QMSYS account.

To set up a basic MASTER.LOGIN item, follow these steps:

From the command prompt, type: **ED VOC MASTER.LOGIN**

The computer should respond with:

```
VOC MASTER.LOGIN
New record
----:
```

To put the editor into Insert mode, type: **I**

The computer should respond with:

```
0001=
```

This means it is waiting for your input. Read the notes below, then type the following lines – or lines that have been amended to your local situation:

14 Whether the settings need to be one less than the window settings depends on the terminal type. This is a safe setting. Try setting to the full window dimensions and see if your terminal type supports this. If it doesn't, then text will wrap in some circumstances, and the page will not fit all information.

```
PA
DATE.FORMAT ON
PTERM CASE NOINVERT
TERM 132,35
```

To exit from Insert mode, press enter on a blank line, and the computer will respond with:

```
Bottom at line 4
----:
```

To save the item, type: **FI**

To exit from the editor without saving, type: **EX**

To check that your MASTER.LOGIN entry has been saved correctly, type:

```
CT VOC MASTER.LOGIN
```

This will display the MASTER.LOGIN entry on the screen. (CT means copy to terminal).

When you are satisfied with the MASTER.LOGIN entry, type **QUIT** or **OFF** to log off from *OpenQM*, then choose 'Reconnect' from the choices offered by *AccuTerm*, log back in, and type **TERM** from the command prompt. The response from *OpenQM* should match the entries you have put into the MASTER.LOGIN item.

```
TERM
Page width: 132
Page depth: 35
Device    : adds4000-at
```

Notes on 'MASTER.LOGIN' entries

The MASTER.LOGIN item is run whenever any process is started in *OpenQM*. This therefore sets system-wide behaviour. There may also be LOGIN items in each account which set specific behaviour for that account.

The MASTER.LOGIN entry outlined above may not be appropriate for all situations. These notes will let you decide how to change this entry to suit your own situation:

The first line (**PA**) tells *OpenQM* that this entry is a PAragraph. A PAragraph is essentially a series of commands.

The second line (**DATE.FORMAT**) tells *OpenQM* to use European style dates. To set American style dates, this line would read:

```
DATE.FORMAT OFF
```

The third line (**PTERM CASE NOINVERT**) tells *OpenQM* not to invert the case of entered text. This may seem an odd thing to do, but multi-value databases evolved from a time when everything was entered in upper case. Case inversion was often a default setting, thereby saving users from manually changing their CAPS LOCK key when coming to use the system.

The final line (**TERM 132,35**) tells *OpenQM* to set the terminal display size to 132 columns by 35 lines. This means that this terminal size will always be set whenever anyone starts an *OpenQM* session. This particular line may be better set in an account LOGIN item (where the setting will only apply to that particular account).

OpenQM has a number of OPTION settings that affect the behaviour of the database. These option settings should NOT be applied in the MASTER.LOGIN item despite the appeal of using this to enforce system-wide behaviour. This is because entries here affect the operation of the QMSYS account, and it is important that this remain in "*OpenQM*" mode.

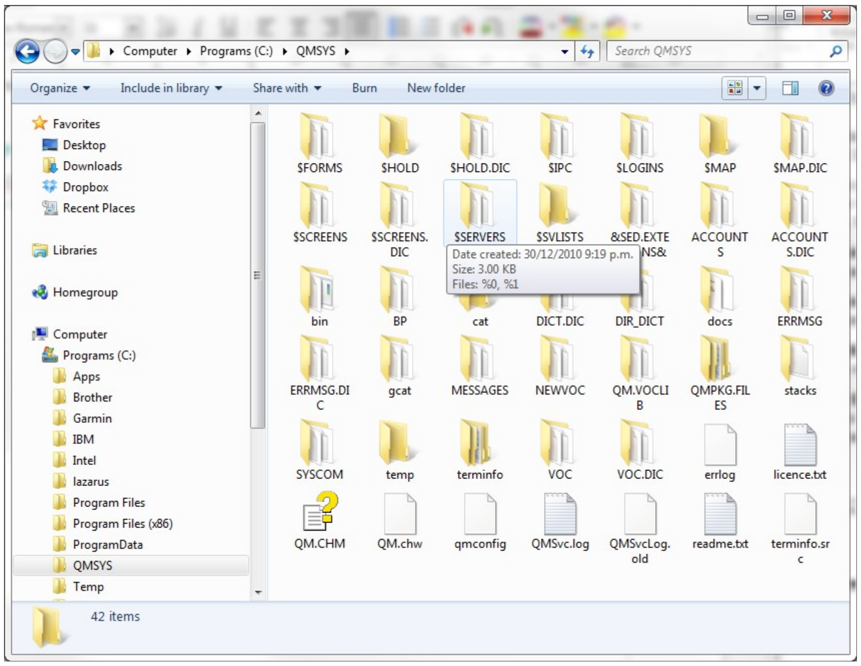
Therefore, keep entries in the MASTER.LOGIN item minimal.

3 Exploring the System

You now have a basic installation of *OpenQM* on your system. What does it look like?

3.1 The View from Windows

The *OpenQM* installation will have created a directory tree starting at `'C:\QMSYS'`. This looks like:



Each of the folders beneath QMSYS represents a file (or file dictionary) within *OpenQM*. The following file descriptions are (mostly) taken straight from the online help system:

Standard account files

File	*Description*
$ACC	This is the account directory viewed as an *OpenQM* directory file.
$COMO	*OpenQM* provides a facility to record output that is displayed at the user's screen in a file. This file is known as a como (command output) file for compatibility with other systems. The $COMO file is automatically created as a directory file when the COMO ON command is first used. The command also specifies the record name to be used to store the output. This file also contains the log files generated by background (phantom) processes.
$FORMS	This VOC entry points to a file in the QMSYS account that is shared by all accounts as a repository for PICK style form queue definitions created using the SET.QUEUE command and used by the SP.ASSIGN command.
$HOLD	This is a directory file used to receive output sent to a print unit by a program or standard command that has been set into mode 3 (output to hold file).
$SVLISTS	This is a directory file used to store saved select lists. See the SAVE.LIST and GET.LIST commands for more information.
$SCREENS	This is a dynamic file used to hold screen definitions that are to be shared between accounts. See the description of the SCRB screen builder for more information.
cat	A subdirectory under the account holding programs added to the private catalogue using the CATALOGUE verb. Users should not modify this file except by use of the associated *OpenQM* commands.
stacks	A subdirectory under the account used to store saved command stacks when a user exits from *OpenQM*.
VOC	The vocabulary, a file that controls all aspects of command processing within *OpenQM*.

The $ACC file needs a little explanation. This is a “virtual” file. When you look at the account through a file manager, the $ACC file does not exist. However, the VOC file in each account contains an entry for $ACC which is simply a file pointer to the account’s folder. Sorting the $ACC file shows the directory contents of the account folder other than O/S folders.

```
SORT $ACC
$ACC........
ERRLOG
LICENCE.TXT
QM.CHM
QM.CHW
QMCONFIG
QMSVC.LOG
QMSVCLOG.OLD
README.TXT
TERMINFO.SRC

9 record(s) listed
```

Files found only in the QMSYS account

File	*Description*
ACCOUNTS	The register of account names described above. This file is visible from all accounts as QM.ACCOUNTS. Field 1 contains the pathname of the account. Field 2 can be used to store a brief description of the account.
bin	A subdirectory, not visible from within *OpenQM*, containing all the operating system level executable programs that form part of *OpenQM*.
docs	A subdirectory containing the *OpenQM* documentation in PDF format. This is not visible from with *OpenQM*.
ERRMSG	A file of standard PICK style message texts provided for compatibility with other multivalue products and used by the *QMBasic* STOP, ABORT and ERRMSG statements for programs compiled with PICK style message processing.
gcat	Not directly visible from inside *OpenQM*, this is the global catalogue directory. This file should only be accessed using the standard catalogue processing commands.
MESSAGES	A file of error messages used internally by *OpenQM*.
NEWVOC	The template vocabulary file from which new accounts are created. This file should not be updated by users as it will be overwritten on upgrading to a new release.
$IPC	This file, not visible from inside *OpenQM*, is used to support inter-process communication and should not be touched by users. (Not present on a PDA).
$MAP	This file, visible from all accounts, is the default destination for a map of the system catalogue produced with the MAP command.
SYSCOM	The SYSCOM file holds standard definitions for use in *QMBasic* programs. It also contains *QMClient*.bas, a set of definitions for use in Visual Basic programs that use the *QMClient* API.
temp	Windows only. This subdirectory holds temporary files that are used to pass control information from the QMSvc service to the QM processes that it starts. All users must have full access to this directory.
terminfo	A subdirectory containing compiled definitions of control data for terminal devices. (Not present on a PDA). Source definitions for these devices are stored in a file named ‘terminfo.src’ in the root folder of the QMSYS account.

Other files

File	*Description*
BP	By convention, this is a Basic Program file. However, you can store basic programs in any file (but they are usually directory files).
DICT.DIC	The dictionary that *QMQuery* uses when listing the dictionary portion of QM files.
QMPKG.FILES	Part of the QM Package Manager.

Accounts will also contain all the other files created by the user(s) or by the installed applications.

3.2 The View from Within OpenQM

Listing the files

There are four commands available to list the files that are available within an *OpenQM* account:

LISTF	List all F-type records in the VOC
LISTFL	Show only local files (in the account)
LISTFR	Show only remote files (referenced in the VOC but not in the account)
LISTQ	List all Q-type records in the VOC

Files accessible by *OpenQM* generally need either an F-type or a Q-type record in the VOC. What does that mean?

When a file is created, the CREATE.FILE command will automatically create an F-type record in the VOC. This lets other *OpenQM* commands "find" the file when you reference it in *QMQuery* statements or *QMBasic* programs.

A Q-type record has three general purposes. The primary one is to reference a file in another account. By placing the Q-type record in the VOC of the current account, you can then reference the remote file as if it were a local file.

The second reason to use a Q-type is to create a synonym name or an alias for the target file. For example, if you have a main file INVOICES, which you archive by financial year to an annual data file associated with the INVOICES dictionary, then the full file name for the 2011-12 invoices could be: INVOICES,201112 This is inconvenient to type, so you may want to create an alias for this file – e.g. INV11.12 This is much easier to use in a *QMQuery* statement.

The third reason to use a Q-type is to reference a file on a remote *OpenQM* server. As dealing with multiple servers is an advanced topic, this will not be considered further in this book.

In essence, an F-type refers to a file within the account, while a Q-type is a pointer to a file which is (usually) in another account.

There is another category of files called 'remote' files. These files aren't local to this account, but may not be in any other account either. For example, we could set up an F-type item that points to a C:\Temp folder. Because the location C:\Temp is not within the local account, this is considered a remote file.

OpenQM comes with a few inbuilt remote files. You can list these with the LISTFR command. These are largely system files.

By default, remote files are not used very much. However, if you – as a developer, wish to separate the data components of an application or an account from the software components, you may find yourself using remote files extensively.

You will learn how to create and manage files in the next section. All we are trying to do at the moment is find out what files we have access to, and where they really exist.

Typing in any of the file listing commands will provide an output similar to those shown below:

```
LISTF
Files referenced by the VOC                                                 Page 1
File name...   FType...   Description.......   DATA Pathname.....   DICT Pathname.....
$ACC           Dir        File for account     .
                          directory
$FORMS         DH         File for form        @QMSYS\$FORMS
                          queue definitions
$HOLD          Dir        File for deferred    $HOLD                $HOLD.DIC
                          prints
$MAP           Dir        File for MAP         @QMSYS\$MAP          @QMSYS\$MAP.DIC
                          output
$SAVEDLISTS    Dir        File for saved       $SVLISTS
                          select lists
$SCREENS       DH         File - Shared        @QMSYS\$SCREENS      @QMSYS\$SCREENS.DI
                          screen definitions                        C
&SED.EXTENSI   DH         F                    &SED.EXTENSIONS&
ONS&
ACCOUNTS       DH         F                    ACCOUNTS             ACCOUNTS.DIC
BP             Dir        F                    BP
DICT.DICT      DH         File - Dictionary    @QMSYS\DICT.DIC      @QMSYS\DICT.DIC
                          for dictionaries
DIR_DICT       DH         F Directory file     @QMSYS\DIR_DICT      @QMSYS\DICT.DIC
                          dictionary
ERRMSG         DH         F Pick style error   @QMSYS\ERRMSG        @QMSYS\ERRMSG.DIC
                          message register
MESSAGES       DH         F                    @QMSYS\MESSAGES
NEWVOC         DH         File - Template      @QMSYS\NEWVOC        @QMSYS\VOC.DIC
                          VOC
QM.VOCLIB      DH         File - Shared VOC    @QMSYS\QM.VOCLIB     @QMSYS\VOC.DIC
                          extension library
Action (Abort/Quit/Next/Suppress pagination):
```

Note the final line of the listing. This line appears whenever a query has filled the screen with data and there is still more data to display. It gives you the opportunity to look at the data – otherwise, it would scroll past too fast to see.

You can answer A or Q to exit from the listing; N or enter to go to the next screen; or S to stop any pagination – this will simply let the query run to the end. Most of the time, you should simply press enter when you are ready to proceed further.

The LISTF commands show the filename, the type of file, a description of the file (if one is present, and the operating system locations (path) of the data and dictionary parts of the file. Note that some of these paths use a synonym of @QMSYS. This means the path of the QMSYS account.

There are two other path synonyms which can be used. These are:

@TMP	the path of the Temp directory defined in the *OpenQM* configuration parameters
@HOME	the path specified by the HOME environment variable on Linux or the HOMEPATH environment variable in Windows. Windows users may need to create this environment variable.

Looking at the listing generated by the LISTF command, several files are shown to have data portions but no dictionary. These include $ACC, $FORMS, $SAVEDLISTS, and BP. Most user files have dictionary portions.

Most files have both a dictionary and a data portion. For example, ACCOUNTS has a data portion named ACCOUNTS, and a dictionary portion named ACCOUNTS.DIC. If you look at the QMSYS account using a file manager, you will see both of these file portions implemented as folders beneath the main QMSYS folder.

Some files are termed 'multi-part'. Such files have one dictionary associated with multiple data portions. In the output from a LISTF command, these are shown like:

```
File name...    FType...    Description.......    DATA Pathname.....    DICT Pathname.....
SOURCE          Mult        F                                           SOURCE.DIC
  BP            Dir                               SOURCE\BP
  BP.SYSCTRL    Dir                               SOURCE\BP.SYSCTRL
  GUIBP.SYSC    Dir                               SOURCE\GUIBP.SYSCT
TRL                                               RL
  UVF.SYSCTR    Dir                               SOURCE\UVF.SYSCTRL
L
```

Note that some of the file names in this listing have wrapped making the structure a little less clear.

The base file name (SOURCE) is shown as 'Mult'. Multiple data portions are specified beneath this base file with their individual file types also shown (in this case they are all directory files). Viewing this using a file manager reveals that folders have been created named SOURCE, and SOURCE.DIC. Underneath the SOURCE folder, four more folders have been created named BP, BP.SYSCTRL, and GUIBP.SYSCTRL, and UVF.SYSCTRL.

```
LISTQ
Indirect file pointers in the VOC                                              Page 1
Name...........    Description.........    Account.....    File..........    Server......
ACCUTERMCTRL       Q                       ACCUTERM        ACCUTERMCTRL
FTBP               Q                       ACCUTERM        FTBP
FTBP.OUT           Q                       ACCUTERM        FTBP.OUT
GUIBP              Q                       ACCUTERM        GUIBP
GUIBP.OUT          Q                       ACCUTERM        GUIBP.OUT
MD                 Q - Vocabulary                          VOC
                   synonym
OBJBP              Q                       ACCUTERM        OBJBP
OBJBP.OUT          Q                       ACCUTERM        OBJBP.OUT
QM.ACCOUNTS        Q Register of QM        QMSYS           ACCOUNTS
                   accounts
SUIBP              Q                       ACCUTERM        SUIBP
SUIBP.OUT          Q                       ACCUTERM        SUIBP.OUT

11 record(s) listed
```

The LISTQ command shows four things about each file – a description (usually just 'Q'), the account the file really resides in, its real file name, and finally the server name – if it is on a remote server. Therefore, the FTBP file actually resides in the ACCUTERM account and has a name in that account of FTBP.

Compare the LISTQ output above with the LISTF output on the previous page. Note that many of the F-type items have descriptions of the file's purpose. You can add descriptions to any files you create (or reference with a Q-pointer) by editing the VOC item and adding the description after the initial descriptor element (F or Q). For example, for the interest rates file which we will create later, you might update the VOC entry to read:

```
File of interest rates
IRATES
IRATES.DIC
```

or:

```
F Interest rates
IRATES
IRATES.DIC
```

Other LISTx commands

OpenQM has a number of other LIST commands describing the system. These are:

LISTK	Lists Keywords in the VOC

LISTM	Lists Menus defined in the VOC
LISTPA	Lists PAragraphs defined in the VOC
LISTPH	Lists PHrases defined in the VOC
LISTPQ	Lists PROCs defined in the VOC
LISTR	Lists Remote items defined in the VOC
LISTS	Lists Sentences defined in the VOC
LISTU	Lists Users currently logged onto the system
LISTV	Lists Verbs defined in the VOC

Note that with the exception of LISTU, all the LISTx commands report on items defined in the VOC. This highlights the significance of the VOC as the central repository of things in *OpenQM*. But what are all these items?

All commands entered at the colon prompt must start with a verb. Some of these verbs are standalone (such as WHO), while others require additional parameters (such as LOGTO *accountname*). Many have optional parameters (such as TERM). Verbs are listed using the LISTV command.

Some commands, particularly those used in *QMQuery*, use keywords to identify parameters entered by the user. Note that keywords are only used within a command - they do not start a command. Keywords are listed using the LISTK command.

Some commands become quite complex. It is often useful to store some of these complex commands as Sentences (a single command) or PAragraphs (multiple commands) in the VOC. These entries are listed using the LISTS and LISTPA commands respectively.

Sentences are also used as partly completed commands. That is, you enter the sentence name and some additional text to complete the command. For example, see the EDIT.LIST command.

Sometimes you will find that you use a particular group of words quite frequently, particularly in *QMQuery*. You can place these words in a PHrase and then simply use the PHrase name in your commands. PHrases are listed using the LISTPH command.

PROCs are procedures often used as a form of job control. They are not a recommended structure within *OpenQM*, as other structures do the same job better. They are included in *OpenQM* for compatibility with other multi-value systems. PROCs are listed using the LISTPQ command.

Menus and users are self explanatory. These can be listed using the LISTM and LISTU commands respectively.

This leaves the LISTR command to list remote items. A remote item is any individual item that you wish to use in the current account which actually exists elsewhere on the system. This is useful when you want access to an item stored in a control file on a system-wide basis. Rather than create the control file and the control item in every account, you create a pointer to the control item using a remote item.

3.3 Getting Help

OpenQM

The Windows download of *OpenQM* comes with a complete set of documentation. This is installed in the `C:\QMSYS\docs` folder by default. This includes the following:

`Index.pdf`	A one page document that directs you to the main manuals
`QM.pdf`	The main *OpenQM* documentation
`Ref.pdf`	A quick reference guide
`Tutorial.pdf`	A tutorial for new users
`Conversion.pdf`	A guide for converting other multi-value systems to *OpenQM*.
`tyqm.pdf`	A "Teach yourself QM" guide.

The main documentation is also available as a Windows help file accessible from within the *OpenQM* environment. Simply type `HELP` or press `F1` from the command prompt. Note that `F1` only works if you are at the command prompt, and help is not context sensitive. Nevertheless, finding material in the help file is usually fairly easy.

If you have installed a Linux or FreeBSD version, you will need to download the manuals from the *OpenQM* website. A 6.8 MB download contains all the reference manuals in zipped PDF format.

Online help is available at the *OpenQM* Google Group, or at the *'Pick and Multivalue Databases'* Google Group. Support is also available directly from Ladybridge Systems for licensed users.

AccuTerm

AccuTerm comes with a standard Windows help file. This is accessed by choosing 'Help | View Help Contents' or 'Help | View Help Index' from the *AccuTerm* menu bar.

PDF manuals for *AccuTerm* are similarly available by choosing 'Help | Documentation | ...'.

Online help is available in the Support Forum at www.asent.com/forum/default.asp or in the 'Pick and Multivalue Databases' Google Group.

4 Accounts, Files and Editors

4.1 Creating and Deleting Accounts

Before we can do any real work in *OpenQM*, we need a work area. We could do everything in the QMSYS account, but that would be bad practice.

It is best to assume that the QMSYS account is the domain of Ladybridge Systems, and we should leave it alone as much as possible. Secondly, we may have different types of activities that we want to keep totally separate from anything else.

So we create accounts to keep sets of information logically grouped together, and separate from other sets of information which we hold in other accounts. It is important to note that each account can still reference information held in other accounts, so the separation of data does not mean that data is inaccessible.

The command to create an account is:

```
CREATE.ACCOUNT  accountname   accountpath
```

For example, the command to create the *AccuTerm* account in the Appendix was:

```
CREATE.ACCOUNT ACCUTERM E:\QM\ACCUTERM
```

This command does the following:

- creates a folder named ACCUTERM underneath the 'E:\QM' folder
- creates a number of standard files within the ACCUTERM folder
- copies the entries from the NEWVOC file in the QMSYS account into the VOC of the ACCUTERM account
- updates the ACCOUNTS file in the QMSYS account so that *OpenQM* knows where to find the ACCUTERM account.

Don't try and do this manually. While it is possible to do this, it is better to let *OpenQM* create (and delete) accounts for you.

The command to delete an account is:

```
    DELETE.ACCOUNT  accountname
```

This command will remove the account folder, all its subfolders, and the reference within the ACCOUNTS file.

Create an account

Let's create an account to test things in *OpenQM*. We'll call it QMINTRO and it will be located at E:\QM\QMINTRO. You should replace this path with a path that is appropriate on your system.

From the command prompt in the QMSYS account, enter in:

```
CREATE.ACCOUNT QMINTRO E:\QM\QMINTRO
Create new directory for account (Y/N)? Y
Creating VOC...
Creating $HOLD...
Creating $SAVEDLISTS...
Creating private catalogue directory...
Adding to register of accounts...
```

Let's check it really is in the ACCOUNTS file:

```
SORT ACCOUNTS
Account.........   Pathname.....................   Description..................
ACCUTERM           D:\QM\ACCUTERM
BRIAN              D:\QM\BRIAN
QMINTRO            D:\QM\QMINTRO
QMSYS              @QMSYS
SYSCTRL            D:\QM\SYSCTRL
```

Now, we'll log to the account:

```
LOGTO QMINTRO
```

Nothing happens when we get there – we are simply left at the colon prompt. Let's see if we really are there:

```
WHO
1 QMINTRO from QMSYS
```

So, this tells us we are logged in on line 1, we are in the QMINTRO account, and that we originally logged into the QMSYS account.

If we do a LISTF command to list the files in the account, we see only the standard files that *OpenQM* has created. We will create some new files shortly.

You might recall from the *AccuTerm* installation that we had to activate *AccuTerm* for the QMSYS account. We need to do that for our new account too. Log to the ACCUTERM account and run FTSETUP to activate this account.

```
LOGTO ACCUTERM
FTSETUP
              Activate the QMINTRO account.
LOGTO QMINTRO
```

How do we know that has worked? Type LISTQ to list the Q-type records in the VOC. You should see the ACCUTERM account referenced in the Account column of the listed Q-pointers.

We should also set up a LOGIN item for the account. This sets the options and other settings for the account. We'll use the *AccuTerm* editor (covered later in this chapter) to create this:

```
WED VOC LOGIN
```

Enter the following lines and the close the editor and save the changes.

```
PA
OPTION PICK
```

Why do we put these in the LOGIN item, and not in the MASTER.LOGIN item? Ladybridge Systems advise that the QMSYS account should always operate in "native" mode. Therefore, using one of the OPTION settings to change the way the account works is not recommended.

What does OPTION PICK do? It changes the way that *QMQuery* operates. This will be covered further in the chapters on *QMQuery*.

4.2 Multi-value File Concepts

It was noted earlier that multi-value files generally have two parts – a dictionary and a data portion. What is the significance of this structure?

The data portion

The data portion contains (surprisingly enough) – data.

Each record in the database is identified by a unique identifier. In multi-value terms, this is called the ID or item-id, and is equivalent to the primary key in relational databases. The real power of the ID comes from its use as a data locator within the database.

As long as the database is given the ID of a record, then the database can (usually) read the record in a single disk read – even when the database has not been indexed. This characteristic has long been used to provide high performance in multi-value systems – by making the ID of a file 'meaningful' (such as a customer number or part number), then the database can find the associated data very quickly.

An example of non-meaningful data as an ID is a sequential number (autoincrement field). This is simply an ID that is assigned to the record, and has no relationship to the data contained in the record.

The choice of the ID for each record thus becomes an important decision (and is discussed extensively in Section 7). There are arguments both for and against using meaningful data as the ID. In general, meaningful data should only be used if:

- it is NEVER going to change
- it will always be unique.

Therefore, a customer number is good, but a customer surname is not.

These rules may seem simple, but in real life you will often find data that does not conform to expectations. Duplicate numbers will exist in data where you expect the number to be unique, and "permanent" numbers will change over time. Part numbers are notorious for this type of duplication and change.

In a complex system, changing the primary key is a non-trivial exercise. Therefore, take care in the initial design stage of the database. If in doubt, use a sequential number and index the database on the key fields.

The dictionary portion

The dictionary portion contains data descriptions. *QMQuery* (covered later in this book) uses these descriptions to extract and display the data.

The following points may be of use in understanding dictionaries:

- Dictionaries are not a schema, although they should "describe" the data
- Dictionaries are not compulsory
- The descriptions they contain may not be accurate
- You can have multiple descriptions for each field
- Dictionary items can contain complex calculations as well as general formatting instructions
- Dictionary items can look up data in other files on the system
- It is up to you as the user/administrator to define what goes in the dictionary.

Notwithstanding the above, the dictionary SHOULD represent the data in the data file. Maintenance and understanding of the system is enhanced immeasurably if the dictionary is complete and up to date. Therefore, it is highly recommended that you use and maintain dictionaries to document the database.

Dictionaries may contain the following types of items:

D	Direct data items. These items describe the data in the file.
I	Indirect data items. These items calculate a new value from the data in the file.
L	Link items. These items join the current file to another file.
PH	Phrases.
C	Calculated data values. These items contain an embedded *QMBasic* program, and are used to generate calculated values.
A	A PICK style attribute defining item.
S	A PICK style synonym item.
X	Other miscellaneous data.

The dictionary type (one of the above values) is declared in the first field of the dictionary item. This book will cover the first four of these dictionary types.

In the next part of this section, we will create a few dictionary items. We will use those dictionary items to import and work with some data. But to fully understand dictionaries, you will need to read the sections on *QMQuery* and *OpenQM* Dictionaries – and then look the *OpenQM* manuals to provide greater depth than will be covered here.

4.3 Creating and Deleting Files

4.3.1 Standard files

Files store the data you want to work with. The command to create a file in *OpenQM* is:

```
CREATE.FILE  filename
```

The CREATE.FILE command as shown creates a two-part file – a dictionary part, and a data part. In some situations, you may only want to create a dictionary or a data portion. The commands to do this are covered in the section on multi-part files below.

The command to delete a file is:

```
DELETE.FILE  filename
```

Both of these commands have variants that allow individual parts of a file (a dictionary or data portion) to be created or deleted independently of the other parts. These variants are normally used to maintain multi-part files – a dictionary file that is associated with multiple data files. These are covered later in this section.

4.3.2 Directory files

Creating a directory file is accomplished using the following command:

```
CREATE.FILE  filename  DIRECTORY
```

As you can see, this is just a small variation on the command to create a standard (hashed) file.

Note that only the data part is created as a directory file – the dictionary is still created as a dynamic hashed (standard) file. In a directory file, all of the file items (records) are created as individual operating system level files rather than being contained within a single dynamic file. Directory files are usually used to store basic programs.

Directory files can be deleted using the standard DELETE.FILE command.

4.3.3 Multi-part files

Multi-part files need to be created when you want multiple data files to use a single dictionary. Typical uses for this type of structure are for archiving data, or keeping data from individual years separate.

When creating a multi-part file, each individual part of the file is created with a separate command:

```
CREATE.FILE  DICT  dictname
CREATE.FILE  DATA  dictname,dataname
```

The first command creates just a dictionary portion, and the command structure is quite obvious. However, the command to create a data portion needs a little more explanation.

In order to associate the data file with a specific dictionary, *OpenQM* must know which dictionary name to use. Therefore, the dictionary name is entered as part of the command. Note the comma that separates the dictionary name from the data file name.

OpenQM can change ordinary files to multi-part files if you add a second data part to it using the data command above.

Deleting parts from multi-part files uses a similar format command to that used for creating the parts:

```
DELETE.FILE  DICT  dictname
DELETE.FILE  DATA  dictname,dataname
```

Note that *OpenQM* allows you to delete the dictionary of a multi-part file without deleting the data portions. Under normal circumstances, this would be an unusual thing to do. But perhaps the dictionary didn't actually contain anything so served no purpose on the system.

Consider the following examples of creating and deleting files:

```
CREATE.FILE TEST
Created DICT part as TEST.DIC
Created DATA part as TEST
Added default '@ID' record to dictionary

CREATE.FILE DATA TEST,TEST2
TEST already exists but not as a multifile
Convert to multifile named "TEST,TEST" (Y/N)? Y
Created DATA part as TEST\TEST2

DELETE.FILE DICT TEST
DICT portion 'TEST.DIC' deleted

DELETE.FILE DATA TEST
Delete all data components of multifile? Y
OK to delete DATA portion 'TEST\TEST'? Y
DATA portion 'TEST\TEST' deleted
OK to delete DATA portion 'TEST\TEST2'? Y
DATA portion 'TEST\TEST2' deleted
Multifile directory 'TEST' deleted'
VOC entry 'TEST' deleted
```

In this sequence, an initial file (TEST) is created. This file is then converted to a multi-part file with the addition of a second data portion (TEST2). The dictionary is then deleted, followed by the two data portions.

4.3.4 Single level files

Sometimes, you may want only a data file without any accompanying dictionary. Such a file may be used for control purposes, and you do not want to do any reporting on the file using *QMQuery*.

The command format is similar to that used for multi-part files:

CREATE.FILE DATA *dataname* {DIRECTORY}

To delete such a file, you can use either the standard or the multi-part DELETE.FILE format. If you use the standard format, *OpenQM* reports that the dictionary part of the file does not exist:

```
CREATE.FILE DATA TEST
Created DATA part as TEST

DELETE.FILE TEST
DATA portion 'TEST' deleted
DICT part of file does not exist
VOC entry 'TEST' deleted
```

4.3.5 Distributed files

Distributed files are a means of treating data contained in multiple files as if they belong to a single file. Or viewed from the other end of the telescope, they are a means of splitting one dataset into a number of related individual files.

You may want to do these things to:

- spread a large file across multiple disks for load balancing
- overcome operating system file size limits

- store more data than can be fitted onto a single disk
- split the file across multiple servers.

A distributed file is not the same as a multi-part file – but you could make a multi-part file into a distributed file.

We won't be using distributed files in this book. Therefore, we won't go into any further details. If you want to know more, read the 'Distributed Files' section in the online help.

4.3.6 Q-Pointers

Q-pointers were briefly mentioned in Section 3.2 in the discussion on listing files. That section indicated that Q-pointers were used to:

- reference a file in another account
- reference a file on a remote server
- provide a friendly name for a file (in any account including the current account).

The best way to see what a Q-pointer does is to see one work.

One of the files in the QMSYS account is named BP (for Basic Programs). How do we access the contents of that file from our current account (QMINTRO)? We create a Q-pointer from our current account to that file. Let's call the Q-pointer BP.QMSYS.

We create the Q-pointer in the VOC of our current account, using one of the *OpenQM* editors:

```
ED VOC BP.QMSYS                 Invoke the editor
VOC BP.QMSYS                    OpenQM responds with this
New record                      and this
----: I                         Type I to go into insert mode
0001= Q                         Type Q to indicate a Q-pointer
0002= QMSYS                     This is the account name
0003= BP                        This is the file name
0004=                           Press enter to exit insert mode
Bottom at line 3                OpenQM responds with this
----: FI                        Type FI to file the item
'BP.QMSYS' filed in VOC         OpenQM responds with this
```

Now, test the Q-pointer:

```
SORT BP.QMSYS
HOLD.FILE.LO
GGER
INDEX.CLS
PCL
PCL.GRID
PREPROC
QMSAVE
SQL.CLS
U0032
U50BB
VFS.CLS

11 record(s) listed
```

We can see here that a Q-pointer can be used to access a remote file. The same concept can be used to access a local file using a different name.

While a Q-pointer is not a real file, it accesses a real file. You can create, modify, and delete items in the real file by using the appropriate commands on the Q-pointer.

Many people have got into problems because they deleted items in a Q-pointer file, thinking that they were only deleting copies of the items. In fact, they deleted the real items in the remote file – so be careful with Q-pointers.

To remove the Q-pointer, you simply delete the VOC entry:

```
DELETE VOC BP.QMSYS
```

It was noted above that Q-pointers could also reference a file on a remote server. As this is a fairly advanced topic, this will not be covered here. However, for a brief overview, you could look at the help topic for QMNet.

4.4 Creating an Example Database

4.4.1 Create a file

Before we create a file, we need to ask:

- what are we going to store in the file
- what are we going to call the file.

There are lots of things to consider about the structure too, but that can wait until after we've created the file.

Our first file will contain a series of exchange rates. We can download these in Excel spreadsheet form from: http://www.rbnz.govt.nz/statistics/tables/b1. Once you are on that page, download the file of daily exchange rates marked: 'B1 Daily (2010 to current)'. The file name is: 'hb1-daily.xls'.

As for a name, we could give the file a long name like EXCHANGERATES, or EXCHANGE.RATES, but for convenience, we'll use something a little shorter – XRATES. Therefore, our command to create the file is:

```
CREATE.FILE XRATES
Created DICT part as XRATES.DIC
Created DATA part as XRATES
Added default '@ID' record to dictionary
```

The computer responds by telling us that it has created the dictionary (DICT) and data (DATA) parts of the file, and has added a record named @ID to the dictionary.

Prepare the data

The data we downloaded isn't in the best layout for importing into *OpenQM*. So, we'll restructure that data so it is ready for importing. Open the file using any spreadsheet program that can read Excel files. The data we want starts in cell A6 and ends (currently) in cell AD878 (although the spreadsheet contains a few more columns).

We want to several things with this data:

- add a header row to identify the data
- strip out the unnecessary rows
- save the spreadsheet in csv format.

In row 5 (currently blank), add the following identifiers to identify each of the currencies. These are actually there official currency identifiers as specified by ISO 4217 (see: http://en.wikipedia.org/wiki/ISO_4217):

```
United States dollar      USD
UK Pound sterling         GBP
Australian dollar         AUD
Japanese yen              JPY
European Euro             EUR
Canadian dollar           CAD
South Korean Won          KRW
Chinese renminbi          CNY
Malaysian ringgit         MYR
Hong Kong dollar          HKD
Indonesian rupiah         IDR
Thai baht                 THB
Singapore dollar          SGD
New Taiwan dollar         TWD
```

Now, delete rows 1 to 4. This will leave the header row as the first row in the spreadsheet.

Cell A1 is now blank. Put a heading name of DATE in this cell.

Go right to the bottom of this spreadsheet and delete the comment rows there.

Now save the spreadsheet as a CSV file. Choose ‘File | Save as’ from the menu, select a file type of CSV[15], and save with the original base file name. This should save the file as ‘hb1-daily.csv’.

Create some dictionary items

As noted above, dictionary items describe the data contained in the data file. The point of creating dictionary items now is that the file import routines used by *AccuTerm* can use these dictionary items to correctly format the data as it is imported.

We want to create a dictionary item for each of the exchange rates in the file. To do this, we are going to use the MODIFY editor within *OpenQM*:

```
MODIFY DICT XRATES
```

The editor will respond: **Id (? for list):**

Type in: **DATE**

A list of fields will now be displayed:

```
Id (? for list): DATE
  1: TYPE/DESC=
  2: LOC      =
  3: CONV     =
  4: NAME     =
  5: FORMAT   =
  6: S/M      =
  7: ASSOC    =
```

And a prompt will appear at the bottom of the screen:

```
TYPE/DESC:
```

This is asking what type of dictionary item are we going to create. Answer: **D**

The prompt changes to LOC This is the field number. Answer: **0**[16]

15 Alternatively, you could save this as a TAB delimited file. Make sure you know which delimiter you are using as we will need to specify that at the file import stage.
16 A field number of zero indicates we are referring to the item-id.

The remaining prompts and answers are:

```
CONV                        D
NAME                        Date
FORMAT                      11R
S/M                         S
ASSOC
```

Just press enter at the `ASSOC` prompt to leave it blank. The following prompt will now appear:

```
Action(n/FI/Q/?):
```

If everything is correct, type **`FI`** to file the item. Otherwise, enter the line number of the entry you would like to correct. Once the item is correct, enter **`FI`** to file it.

Now create the following dictionary items:

ID	**Type**	**Loc**	**Conv**	**Name**	**Format**	**S/M**	**Assoc**
USD	D	1	MR44,	US Dollar	7R	S	
GBP	D	2	MR44,	UK Pound	7R	S	
AUD	D	3	MR44,	Aus Dollar	7R	S	
JPY	D	4	MR22,	Jap Yen	7R	S	
EUR	D	5	MR44,	Euro	7R	S	
CAD	D	6	MR44,	Canada Dollar	7R	S	
KRW	D	7	MR22,	SKorea Won	7R	S	
CNY	D	8	MR44,	Chinese Yuan	7R	S	
MYR	D	9	MR44,	Malay Ringgit	7R	S	
HKD	D	10	MR44,	HK Dollar	7R	S	
IDR	D	11	MR22,	Indonesia Rupiah	9R	S	
THB	D	12	MR44,	Thai Baht	8R	S	
SGD	D	13	MR44,	Singapore Dollar	7R	S	
TWD	D	14	MR44,	Taiwan Dollar	7R	S	

To exit the editor, simply press enter when it prompts you for a new `ID`. Now to see your completed dictionary items, type:

```
SORT DICT XRATES
@ID.........  TYPE  LOC..........  CONV..  NAME........  FORMAT  S/M  ASSOC...
DATE          D     0              D       Date          11R     S
@ID           D     0                      XRATES        10L     S
USD           D     1              MR44,   US Dollar     7R      S
GBP           D     2              MR44,   UK Pound      7R      S
AUD           D     3              MR44,   Aus Dollar    7R      S
JPY           D     4              MR22,   Jap Yen       7R      S
EUR           D     5              MR44,   Euro          7R      S
CAD           D     6              MR44,   Canada Dolla  7R      S
                                           r
KRW           D     7              MR22,   SKorea Won    7R      S
CNY           D     8              MR44,   Chinese Yuan  7R      S
MYR           D     9              MR44,   Malay Ringgi  7R      S
                                           t
HKD           D     10             MR44,   HK Dollar     7R      S
IDR           D     11             MR22,   Indonesia Ru  9R      S
                                           piah
THB           D     12             MR44,   Thai Baht     8R      S
SGD           D     13             MR44,   Singapore Do  7R      S
                                           llar
TWD           D     14             MR44,   Taiwan Dolla  7R      S
                                           r

16 record(s) listed
```

Note that some of the descriptions have wrapped within their display field.

You probably understand that we have just created a set of descriptions of the data we are going to store in the database – something like an SQL schema – but it won't be totally clear what the elements of the definitions mean.

The 'D' in the first field indicates that this is a 'D-type' (direct) dictionary item. A 'D-type' item describes the date in the file.

An ‘I’ in this field would indicate that this field is an ‘I-type’ (indirect) field. An indirect field may also go by the name of a virtual field, a lookup field, or a calculated field in other databases.

The second field contains the field number of the data. The listing shows that we are going to store the US Dollar data values in the first field of the database. An ‘I-type’ dictionary item would contain an expression in this field.

A field number of zero indicates that we are referring to the ID field.

The third field contains a conversion code. This code is used to convert the data between an internal (storage) and an external (display) format. The conversion code we entered for ‘IDR’ (the Indonesian Rupiah) was ‘MR22,’. The ‘MR’ means that we are using a masked decimal conversion where the output should be right-justified. The first ‘2’ means that we wish to display 2 decimal places, while the second ‘2’ indicates the position of the implied decimal point in the data. The comma indicates that we should insert thousands separators in the output format.

Let’s give an example. In January of 1999, there were 4,602.73 Indonesian Rupiah per NZ Dollar. If we apply an input conversion of ‘MR22,’ to this value, we get an internal storage value of 460273. If we apply an output conversion of ‘MR22,’ to the internal value of 460273 then we get a display value of 4,602.73.

We also used a conversion code of ‘D’ for the DATE dictionary item. ‘D’ is a generic date conversion. This can take on many variations as we will see later.

The fourth field is the display name for the field.

The fifth field gives the field width and justification.

The sixth field tells us whether the data is single or multi-valued. All this data is single valued.

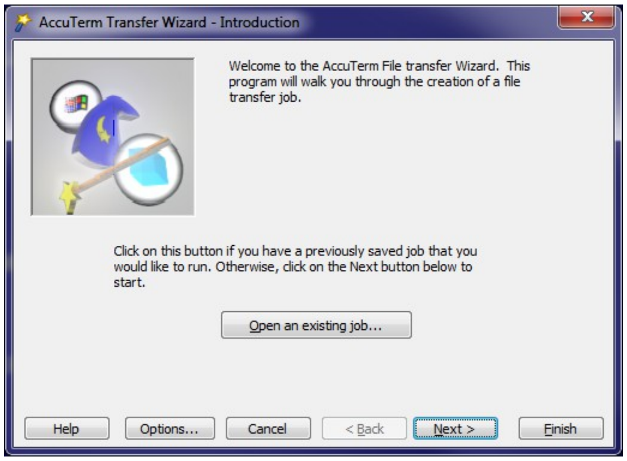
We haven’t used the seventh field. We can enter the name of an association in this field. An association links several fields together so the query processor knows to process them as linked fields.

Now we have a data set ready to be imported, and a basic set of dictionary items to describe the data. Let’s do the import.

Upload the data

We’ll use the File Transfer Wizard to import the data. We could also run the import directly from the command-line. The instructions for that process follow the instructions for using the wizard.

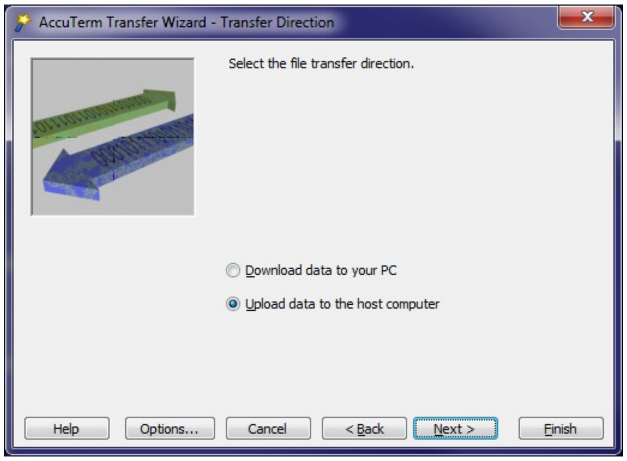
To start the upload, choose ‘MultiValue | Import Data’ from the *AccuTerm* menu bar. This brings up the following dialog box:



Welcome to the AccuTerm File transfer Wizard. This program will walk you through the creation of a file transfer job.

Click on this button if you have a previously saved job that you would like to run. Otherwise, click on the Next button below to start.

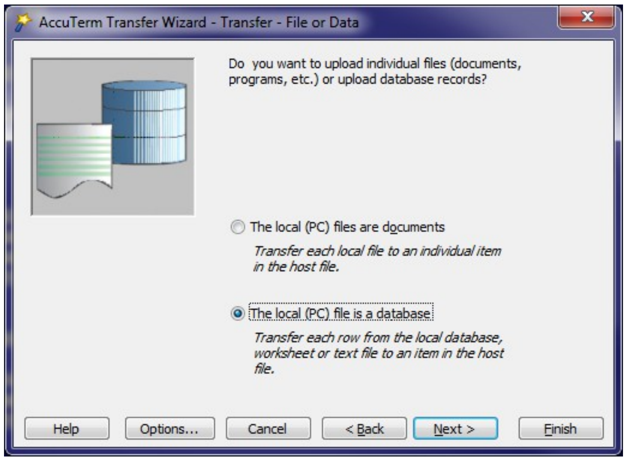
Click on 'Next'. This brings up the following screen:



Select the file transfer direction.

( ) Download data to your PC
(●) Upload data to the host computer

Select the upload option, and then click on 'Next'.

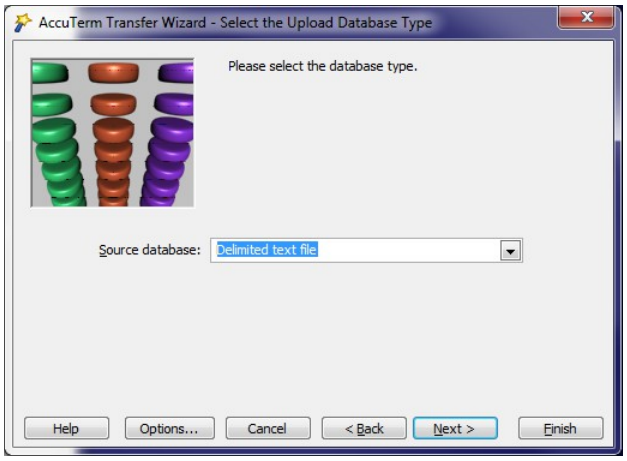


Do you want to upload individual files (documents, programs, etc.) or upload database records?

( ) The local (PC) files are documents
Transfer each local file to an individual item in the host file.

(●) The local (PC) file is a database
Transfer each row from the local database, worksheet or text file to an item in the host file.

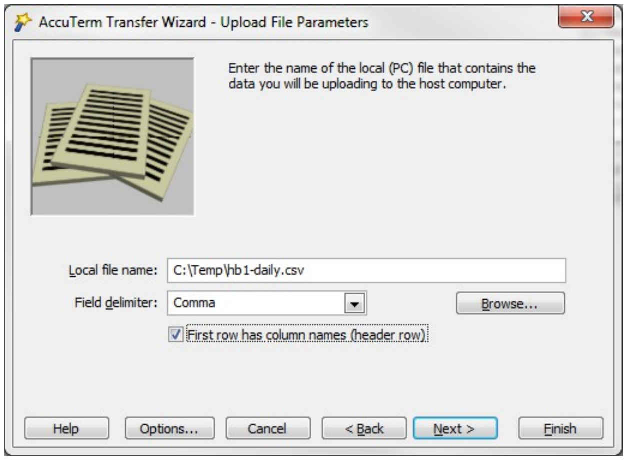
We now have to specify the type of data we are transferring. We would choose the 'documents' option if we were transferring program files. However, in this instance, we want the 'database' option. Select the 'database' option and click 'Next'.


AccuTerm Transfer Wizard - Select the Upload Database Type

Please select the database type.

Source database: Delimited text file

We now need to specify the source data type. Choose ‘Delimited text file’ and then click on ‘Next’.


AccuTerm Transfer Wizard - Upload File Parameters

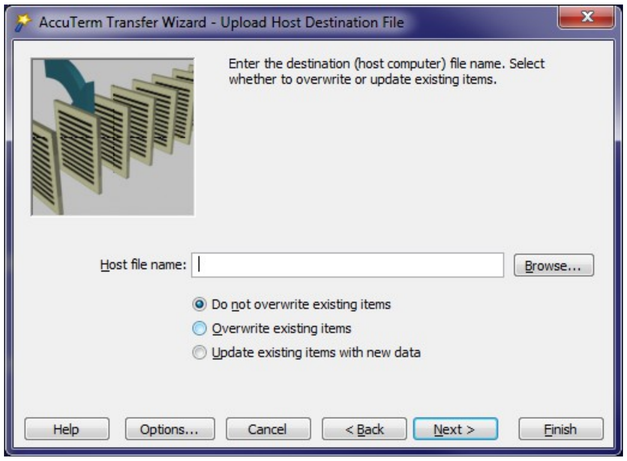
Enter the name of the local (PC) file that contains the data you will be uploading to the host computer.

Local file name: C:\Temp\hb1-daily.csv

Field delimiter: Comma

☑ First row has column names (header row)

Click on the ‘Browse’ button, and select the file containing the data. *AccuTerm* should detect the field delimiter character, but you should check that this is correct. Tick the box ‘First row has column names (header row)’. Click on ‘Next’.


AccuTerm Transfer Wizard - Upload Host Destination File

Enter the destination (host computer) file name. Select whether to overwrite or update existing items.

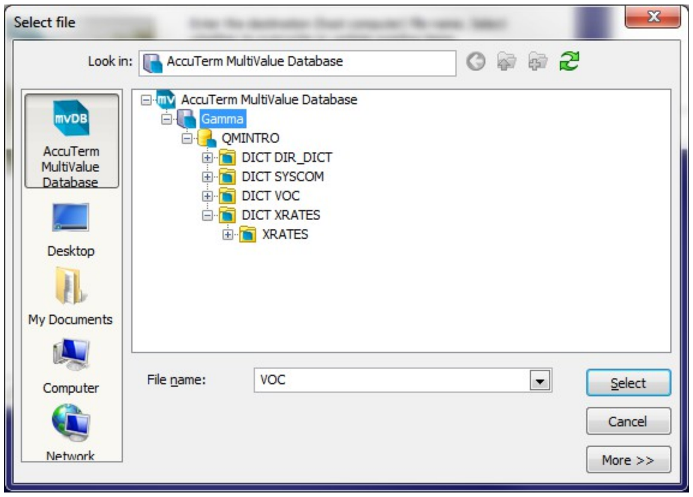
Host file name:

● Do not overwrite existing items

○ Overwrite existing items

○ Update existing items with new data

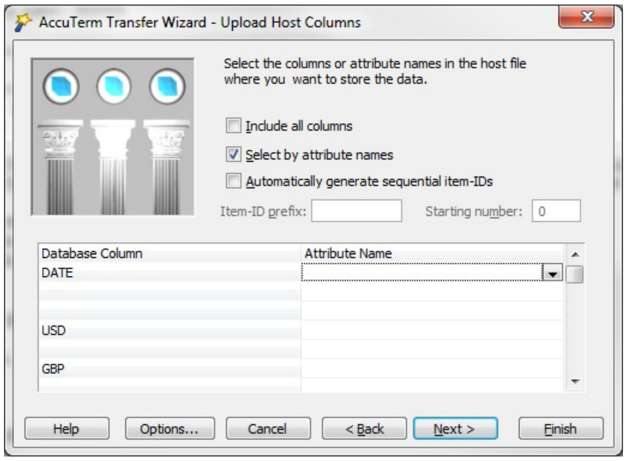
This brings us to an equivalent dialog box to specify the destination file in the *OpenQM* environment. We could manually complete this – which in this case is very simple, or we could click on the 'Browse' button to display a hierarchical tree structure of available files. Either type in 'XRATES' or click on the 'Browse' button to select the file with the mouse.


Select file
Look in: AccuTerm MultiValue Database
AccuTerm MultiValue Database
Desktop
My Documents
Computer
Network
Select
Cancel
More >>

AccuTerm MultiValue Database
Gamma
QMINTRO
DICT DIR_DICT
DICT SYSCOM
DICT VOC
DICT XRATES
XRATES

File name: VOC

Select one of the radio buttons. In this case, it doesn't matter which option is chosen because we are starting with an empty file. Click on 'Next'.



Select the columns or attribute names in the host file where you want to store the data.

- [ ] Include all columns
- [✓] Select by attribute names
- [ ] Automatically generate sequential item-IDs

Item-ID prefix: Starting number: 0

Database Column	Attribute Name
DATE	
USD	
GBP	

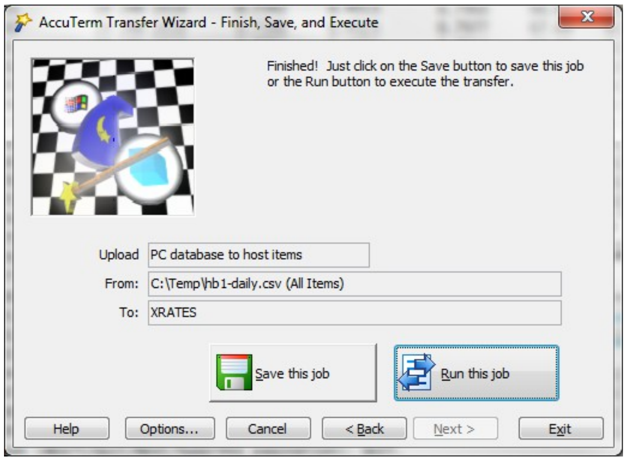
Now, we have to match the data fields in the source data file to their location in the destination file. The first thing to do here is click in the check box marked 'Select by attribute names'. This means that *AccuTerm* will use the dictionary items we have already set up to determine where to put each column of data, and how to format the data as it is imported.

Now, click in the grid area under 'Attribute name' in row 'Date'. Select 'DATE' from the drop-down list. This maps the data identified by the header element 'DATE' to the *OpenQM* database field with the dictionary item 'DATE'.

Now, continue down the grid, matching the source columns and dictionary items. As we have given the columns in the input file the same name as the database fields, this is quite simple:

```
Database Column     Attribute name
DATE                DATE
USD                 USD
GBP                 GBP
AUD                 AUD
JPY                 JPY
etc
```

Now click on 'Next'.



Finished! Just click on the Save button to save this job or the Run button to execute the transfer.

Upload: PC database to host items
From: C:\Temp\hb1-daily.csv (All Items)
To: XRATES

Now we can run the upload. Click on the 'Run this job' button and you will see an 'Executing this job' message appear momentarily in the dialog box, before it returns to the state shown above. You will also see a transfer message appear on the main *AccuTerm* screen:

```
Converting hb1-monthly.csv from CSV text format for upload... done.
```

Finally, click on the 'Exit' button to exit from the Wizard. You will be prompted to save the job. Click 'No', and you will be returned to the main *AccuTerm* screen.

Importing from the command line

Importing via the command line is actually simpler than using the wizard – as long as you follow a couple of simple rules:

- Make sure that the names you use in the header row match the dictionary names
- Make sure that the field delimiter character is set correctly.

At this point, we know that we have used the same identifiers in the header row as we used for the dictionary items, so we'll proceed and check the delimiter character as we go.

From the command-line, type in:

```
FTD
```

`FTD` is the import program for database items. If we wanted to import document type items (e.g. program files), we would use the `FT` import program. The program responds with:

```
AccuTerm Data Transfer Utility

(S)end, (R)eceive, (C)onfigure, (O)ptions, (H)elp or (E)xit ?
```

We need to set the delimiter character, so press 'O' for options. This displays the following menu:

```
Current parameter settings are:
    1. Preserve file extension....... Yes
    2. Overwrite existing item....... Yes
    3. Field delimiter............... Tab

Enter parameter number (1-3) to modify:
```

The field delimiter character is set to 'Tab' by default, but our import file is delimited by commas. Press '3' to change the delimiter character. Once the correct delimiter is displayed, press enter to return to the original menu.

Sending and receiving is relative to the *OpenQM* server. We want to import data into *OpenQM*, so we want to receive the data. Press 'R'. You are now given a choice of file transfer protocols:

```
File transfer protocol: (A)SCII or (K)ERMIT ?
```

Kermit is the recommended option. Press 'K'. You are now prompted for the location of your file to import:

```
Enter source (DOS) file name (d:\directory\file.ext):
```

Enter the filename including its full path (e.g. C:\Temp\hb1-daily.csv). You are now asked whether the file has a header record:

```
Use Header Record (<Y>/N):
```

Answer 'Y', or simply press enter. Finally, you are asked to enter the *OpenQM* file name:

```
Enter target (PICK) file name:
```

Enter the file name (XRATES) and press enter. A conversion message will appear on the screen, followed by transfer progress window. When the transfer is complete, the window will close and a status message will appear on the screen.

```
Converting hb1-daily.csv from CSV text format for upload... done.

Transfer status: Successful transfer.
Transferred 873 items, 132206 bytes.
```

If you spelt any of the names in the header row incorrectly, you will get a message like:

```
Transfer status: Attribute defining item 'JPB' not on file
```

Correct the spelling to make it match what you have used in the dictionary – or correct the dictionary item if you got that wrong. Then try the import again.

Did it work?

Theoretically, we now have all that exchange rate data in the *OpenQM* file named XRATES. Lets check that.

```
SORT XRATES
XRATES....
15346
15347
15348
15349
15352
15353
```

Well … something is in the file – but what is it? The ID that is being displayed is nothing like the date we had as the ID column in the spreadsheet. The answer is that the date has been converted to a serial number.

Let's try a little more:

```
SORT XRATES DATE USD GBP AUD JPY EUR
XRATES....   Date........   US Dollar   UK Pound   Aus Dollar   Jap Yen   Euro...
15346         05 JAN 2010      0.7344     0.4564       0.8046     67.98    0.5096
15347         06 JAN 2010      0.7343     0.4591       0.8056     67.39    0.5113
15348         07 JAN 2010      0.7378     0.4606       0.8020     68.11    0.5119
15349         08 JAN 2010      0.7325     0.4597       0.7980     68.50    0.5119
15352         11 JAN 2010      0.7395     0.4604       0.7965     68.45    0.5126
15353         12 JAN 2010      0.7422     0.4605       0.7980     68.33    0.5112
```

If we check back to our source spreadsheet, we find that the data being displayed matches with the values recorded there.

These SORT commands are actually basic *QMQuery* reports. We will develop more advanced reports in the later sections on *QMQuery*.

Now – check that the header row was not imported. This shouldn't happen – but sometimes it does. If it was imported, then it will have an ID of 'DATE' - this being the name we used in the ID field of the header row. Given the rest of the ID's are sequential numbers, 'DATE' will either sort at the very beginning of the data, or at the very end. We saw that it didn't appear in the ascending sort statement, so let's try a descending sort:

```
SORT XRATES BY.DSND @ID DATE USD
XRATES....   Date........   US Dollar
16609         21 JUN 2013      0.7770
16608         20 JUN 2013      0.7857
16607         19 JUN 2013      0.7991
16606         18 JUN 2013      0.7990
```

Everything is OK here, so we will move on to the next file. If an item named 'DATE' did appear, you should delete it:

```
DELETE XRATES DATE
1 record(s) deleted
```

4.4.2 Add another file to the database

The *QMQuery* section will cover (amongst other things) how to look up information in other files. To do this, we need another file.

Our previous file looked at exchange rates. Let's now create a file that contains interest rate data. We can get this data from the same site as the exchange rate data obtained earlier. See: http://www.rbnz.govt.nz/statistics/tables/b2. Download the spreadsheet marked 'B2 Daily (2010 to current)'.

This spreadsheet has some cells containing '-', which will be a nuisance for us. Select all the data in the spreadsheet, and use the 'Search and replace' function to replace all of these with nulls.

Now, create a header row of identifiers in row 5 as follows:

Date	DATE
Official Cash Rate	OCR
Overnight interbank cash rate	OVERNIGHT
30 days	DAYS30
60 days	DAYS60
90 days	DAYS90
1 year	YRS1
2 year	YRS2
5 year	YRS5
10 year	YRS10

Delete rows 1 to 4, then go to the bottom of the spreadsheet and delete the comment rows there. Save the spreadsheet as a CSV file.

Now let's create the *OpenQM* file, and enter some dictionary items to define the data and the location of each element for the data import.

```
CREATE.FILE IRATES

MODIFY DICT IRATES
```

Create the following dictionary items:

ID	Type	Loc	Conv	Name	Format	S/M	Assoc
DATE	D	0	D	Date	11R	S	
OCR	D	1	MR22,	Official Cash Rate	7R	S	
OVERNIGHT	D	2	MR22,	Overnight Cash Rate	7R	S	
DAYS30	D	3	MR22,	30 Day Bank Bill	7R	S	
DAYS60	D	4	MR22,	60 Day Bank Bill	7R	S	
DAYS90	D	5	MR22,	90 Day Bank Bill	7R	S	
YRS1	D	6	MR22,	1 Yr Govt Bonds	7R	S	
YRS2	D	7	MR22,	2 Yr Govt Bonds	7R	S	
YRS5	D	8	MR22,	5 Yr Govt Bonds	7R	S	
YRS10	D	9	MR22,	10 Yr Govt Bonds	7R	S	

While there are other columns in the spreadsheet, we won't import them.

Now start the file transfer from *AccuTerm*, and import the data into the IRATES file. Test the import by typing:

```
SORT IRATES DATE OVERNIGHT DAYS90 YRS10
```

The resulting list of interest rates should match the data in the source spreadsheet.

```
SORT IRATES DATE OVERNIGHT DAYS90 YRS10
IRATES....   Date.......   Overnight Cash Rate   90 Day Bank Bill   10 Yr Govt Bonds
15346        05 JAN 2010                  2.52               2.80               6.13
15347        06 JAN 2010                  2.37               2.79               6.07
15348        07 JAN 2010                  2.38               2.78               6.06
15349        08 JAN 2010                  2.25               2.78               6.06
15352        11 JAN 2010                                     2.76               6.05
15353        12 JAN 2010                  2.44               2.78               6.10
```

4.4.3 Add a third file to the database

The third file we are going to use is somewhat larger, and rather than stepping you through the creation of the file, you can just download it. There are a few supporting files too.

Download the files at: www.rushflat.co.nz/files/qmintro.zip

Unzip the files, and place them in your QMINTRO account. Now, we'll need to add entries to the VOC so that *OpenQM* can find them. Use one of the editors (see Section 4.6) to create the following VOC entries:

```
CT VOC NCY.C NCY.R TEX.H TEX.QCH
VOC NCY.C
1: F
2: NCY.C
3: NCY.C.DIC

VOC NCY.R
1: F
2: NCY.R
3: NCY.R.DIC

VOC TEX.H
1: F
2: TEX.H
3: TEX.H.DIC

VOC TEX.QCH
1: F
2: TEX.QCH
3: TEX.QCH.DIC
```

[An alternative approach is to create the files from the *OpenQM* command prompt, then delete the files using a file manager. Finally, drop the downloaded files into the places where you deleted the original files].

We now have a set of files, and a matching set of file pointers.

These files contain the following data:

NCY.C	Country names
NCY.R	Region names
TEX.H	Harmonised code descriptions
TEX.QCH	New Zealand export data by quarter, country, and HS chapter.

We'll investigate this data later.

4.4.4 Create a BASIC programs file

While this book does not really cover programming, we do create some programs that will be used by *QMQuery*. Programs are normally stored in directory files (although you can store them in hashed files if you wish). We'll create that file now:

```
CREATE.FILE DATA BP DIRECTORY
Created DATA part as XBP
```

Notice that we created this as a single level file. Normally, we don't need a dictionary for a programs folder.

We don't need to anything more at this stage.

4.5 General Comments About The File Import Process

As you loaded the data from the spreadsheets into *OpenQM*, you probably noticed a number of things about the *AccuTerm* file transfer facilities, and wondered why we've done things in that manner.

For example, you will have noted that *AccuTerm* gives a choice of data sources. It can import data direct from a number of common formats including Excel and Microsoft Access. Why then did we use a CSV file rather than using the Excel file directly?

The answer here is that using an Excel import requires Excel to be actually installed on your PC. If you don't have Excel, you can't use the Excel import. On the other hand, CSV files will work for everyone.

The identifiers we used in the heading row of the data matched the dictionary names. This makes it easy to map the import fields to the database fields.

Likewise, the dictionary names chosen for currencies in the XRATES file were the official ISO abbreviations for those currencies. Using standards like this wherever possible can save a lot of confusion at a later date.

In the mapping process, there were blank columns of data in the import file, and columns of data with a heading element, which we didn't import. This shows that the import process is flexible, and allows selective import.

The import process “converts” the data to internal format as it is imported. Looking at the `SORT` statement above, the obvious conversion is the date “05 JAN 2010” being converted to a date value of `15346`. This conversion was done by the ‘D’ value in the `CONV` field of the `DATE` dictionary item. And when we use the `SORT` statement to display the data, the ‘D’ conversion in the `DATE` field converts the value of `15346` back to “05 JAN 2010”.

All the other data was converted to internal format using the relevant conversion codes. To see the internal format of the data, use the `CT` command to display an item:

```
CT IRATES 15346
IRATES 15346
1: 250
2: 252
3: 273
4:
5: 280
6:
7: 432
8: 552
9: 613
```

The Official Cash Rate (OCR) was stored in attribute (field) 1 of the `IRATES` file, and was 2.50 per cent at the 5th of January 2010. But it is shown here as ‘250’. This is the internal format for 2.50 using a conversion code of ‘MR22,’. All the interest rates were stored using conversion codes of ‘MR22,’.

Most items in the `XRATES` file were stored using conversion codes of ‘MR44,’. So, lets look at one of those items:

```
CT XRATES 15346
XRATES 15346
01: 7344
02: 4564
03: 8046
04: 6798
05: 5096
06: 7645
07: 84529
08: 50145
09: 24952
10: 56954
11: 686035
12: 243857
13: 10269
14: 232915
```

The first attribute here is the United States dollar. This had a value of 0.7344 on the 5th of January 2010, and is shown here as 7344. Once again this is internal format for the value of 0.7344 using a conversion of ‘MR44,’.

In general, a conversion will remove formatting characters and store data to an implied decimal precision. Note that the data is NOT converted to a binary representation of the number as it is with many other databases – the data is stored as literal characters. This makes it easy to see exactly what is being stored by the database. (The implications of this on the space required for the database was commented on earlier in Section 1.1.1).

4.6 Editors

4.6.1 OpenQM editors

OpenQM has three editors for use in a terminal emulation environment. These are:

ED	A basic line editor
MODIFY	A specialised editor used for editing dictionary items and data files
SED	A full screen editor usually used for editing program source code

Although we have already used ED (when we created the MASTER.LOGIN item), this book will not use it again. Nor will it cover the use of SED. To fully understand these editors, see the on-line help file.

Of the *OpenQM* editors, it is MODIFY that you should particularly learn.

When used to edit dictionary items, MODIFY presents you with appropriate prompts for each line of the dictionary item. Less obviously, MODIFY compiles dictionary items as they are filed, thereby highlighting syntactical errors before the dictionary item is used.

When used to modify a data item, MODIFY uses the file dictionary to create the prompts displayed to the user. In contrast, the ED editor simply displays a line number which is unhelpful if you do not know the position of each data element in the item.

The basic syntax for using the MODIFY editor is:

```
MODIFY {DICT} filename {list of item-id’s}
```

If no item-id’s are specified, then MODIFY will prompt you for an item-id, or if a select-list is present, then MODIFY will take the item-id’s from the select-list (see Section 6.2 for information on select-lists). Example commands are:

```
MODIFY DICT XRATES
MODIFY DICT XRATES USD
MODIFY XRATES
MODIFY XRATES 15346
```

In the first and third examples, MODIFY will prompt for an item-id, while in the other two examples, the item-id has been supplied on the command-line.

4.6.2 WED – The AccuTerm editor

The principal reason that this book does not cover the use of ED or SED is that *AccuTerm* provides the WED editor. Most users will find WED to be much easier to use than ED or SED.

WED is a full-screen Windows editor.

Of course, if you aren’t using Windows, or you aren’t using *AccuTerm*, then you will need to learn one of the other editors, or some other editing tool (which may come with your terminal emulator).

Do NOT use WED to edit dictionary items! This is because some dictionary items contain compiled code. MODIFY hides this from you, but WED will not.

4.6.3 The command stack and command editing

OpenQM is a command driven environment – that is, you type commands from the keyboard to control what *OpenQM* does. *OpenQM* stores these commands as you work, and provides quick access to the last 99 commands issued (this number is configurable), so you can either re-run an earlier command or edit the command before running it.

These editing facilities are particularly important for use with *QMQuery*. *QMQuery* commands are often built up over a series of iterations. This may start with a basic statement that selects and sorts records. Break points and output data will then be added to this, followed by headings and footings. The final command may cover several lines on the screen.

There are two sets of command line editing facilities. The first are the ‘dot’ commands that all are present in all multi-value environments (with minor differences in each environment). The second allows direct editing of commands on the command stack using the arrow keys.

The dot commands

The dot commands are literally a dot (period) followed by a single character. Some of these commands are further followed by parameters that control *OpenQM*’s response to the command.

To see the list of dot commands, type `.?` (dot question-mark) at the command prompt:

```
.?
.An text     Append text to command stack entry n.
.Cn/s1/s2/G  Replace s1 by s2 in stack entry n. G = global replace.
.Dn          Delete stack entry n.
.D name      Delete named sentence or paragraph.
.In text     Insert text as stack entry n.
.Ln          Display last n lines from stack. Default is 20.
.L name      Display named sentence or paragraph.
.Rn          Recall stack entry n to top of stack.
.R name      Recall named sentence or paragraph to stack.
.S name s e  Save stack entries s to e as sentence or paragraph name.
.Un          Convert stack entry n to uppercase
.Xn          Execute command n. Default is 1.
.X file id   Execute command stored in named file and record
Spaces are required where shown.  n defaults to one in all cases if omitted.
Use .DP, .LP, .RP and .SP to reference private VOC file.
```

The most common dot command you will use is `.L` (dot L). This provides a listing of recent commands:

```
.L
20  MODIFY DICT XRATES
19  SORT XRATES
18  MODIFY DICT XRATES
17  SORT XRATES
16  SORT XRATES DATE USD GBP AUD JPY EUR
15  FTTCL
14  FTSERVER 1
13  CLEAR.FILE DATA IRATES
12  sort dict irates
11  MODIFY DICT IRATES
10  SORT IRATES
09  FTTCL
08  FTSERVER 1
07  SORT IRATES DATE OVERNIGHT DAYS90 YRS10
06  CT IRATES 15346
05  sort xrates
04  CT XRATES 15346
03  sort irates
02  SORT IRATES DATE OVERNIGHT DAYS90 YRS10
01  SORT XRATES DATE USD GBP AUD JPY EUR
```

The most recent commands are at the bottom of the list nearest the current cursor position. The default number of commands is 20, but this can be changed by specifying a number in the `.L` command:

```
.L5
05  sort xrates
04  CT XRATES 15346
03  sort irates
02  SORT IRATES DATE OVERNIGHT DAYS90 YRS10
01  SORT XRATES DATE USD GBP AUD JPY EUR
```

To execute a command that is already on the command stack, use `.Xn` where n is the number of the command. To repeat the sort on the dictionary of the `IRATES` file (command number 2), we would type `.X2`

```
.x2
SORT IRATES DATE OVERNIGHT DAYS90 YRS10
IRATES....   Date.......   Overnight Cash Rate   90 Day Bank Bill   10 Yr Govt Bonds
15346        05 JAN 2010                  2.52               2.80               6.13
15347        06 JAN 2010                  2.37               2.79               6.07
15348        07 JAN 2010                  2.38               2.78               6.06
15349        08 JAN 2010                  2.25               2.78               6.06
```

When we execute a command like this, the specified command is executed as if it were just entered at the keyboard. This duplicates the existing command, and pushes all previous commands down the stack. In this case, we now have the command:

```
SORT IRATES DATE OVERNIGHT DAYS90 YRS10
```

at both position 1 and position 3 on the stack.

The `.C` command changes the text in a command. While we could change the text in any command on the stack, it is usually most convenient to operate on the first command. In this case, we may need to retrieve (`.R`) an earlier command from the stack.

```
.R2
02  SORT XRATES DATE USD GBP AUD JPY EUR
.C/AUD/CNY
01  SORT XRATES DATE USD GBP CNY JPY EUR
```

This sequence has retrieved the 2nd command from the stack, and then changed '`AUD`' to '`CNY`' in the list of currencies to display. We could then execute this new command simply by typing `.X`

```
.X
SORT XRATES DATE USD GBP CNY JPY EUR
XRATES....   Date........   US Dollar   UK Pound   Chinese Yuan   Jap Yen   Euro...
15346         05 JAN 2010      0.7344     0.4564         5.0145     67.98    0.5096
15347         06 JAN 2010      0.7343     0.4591         5.0129     67.39    0.5113
15348         07 JAN 2010      0.7378     0.4606         5.0376     68.11    0.5119
15349         08 JAN 2010      0.7325     0.4597         5.0014     68.50    0.5119
```

In this case, because we did not specify a command number, it assumed the first command.

Use of the .S command and the alternate form of the .X command is covered later in this book (see Section 5.7.2). Otherwise, see the official *OpenQM* documentation for information on the rest of the dot commands.

Editing keys

OpenQM provides a much simpler and more intuitive way to edit the commands on the command stack than by using the dot commands. This is by using the editing keys (arrow keys, Home, End) on your keyboard[17].

Press the Up Arrow at the command prompt. The first command on the stack will be displayed at the prompt, with the cursor at the home position. Press the Up Arrow again, and

17 This works with *AccuTerm* using the special *AccuTerm* terminal definitions. You may need to configure your keyboard and/or terminal definition if you are using some other terminal emulator.

the next command will be displayed, and so on through the command stack. Pressing the down arrow will bring you back down the command stack.

When you have got to the command you want, you can use the Left and Right Arrow keys to move through the command, and the Home and End keys to move to each end of the command. Typing text in from the keyboard will insert the text into the command at the cursor position, pressing Delete will delete the text underneath the cursor, and using the Backspace key will delete the text before the cursor.

When you have finished editing the command, press Enter to execute it.

If you want to abandon the editing you have made, use the Up or Down Arrow keys to move off the command, or use Ctrl-G to return to the command prompt. Your changes won't be saved.

There are other keyboard editing commands and configuration options. Search the help file for "command editor" for more information.

Preserving the command stack between sessions

Command stacks are saved between sessions in the stacks folder of your entry account. The stacks folder isn't visible from within *OpenQM*, but you can see it using a file manager.

Now – what was that about the entry account? There is potentially a stacks folder in every account. This is created the first time that a command stack is saved in the account. So, if you can't see a stacks folder in the account, it is because no stacks have been saved there.

If you have multiple accounts and there is a stacks folder in each of them, then which stack is used when? Where is your current session stack actually saved?

You use the stack from the account by which you enter *OpenQM*, and the stack is saved there when you exit – even if you have logged to another account during the session.

For example, if you initially log into QMSYS, then you will use your command stack from the QMSYS account even when you log to the ACCUTERM account and the QMINTRO account. Likewise, your stack for your entire session will be saved into the stacks folder in the QMSYS account when you finally log off.

The WHO command will tell you which account your stack belongs to:

```
WHO
1 QMINTRO

WHO
2 QMINTRO from QMSYS
```

In the first example, you are in the QMINTRO account and will be using your stack from the QMINTRO account. In the second example, you are in the QMINTRO account, but will be using your stack from the QMSYS account.

Once you have *OpenQM* set up, you will usually log directly into your normal working account (QMINTRO), and you won't notice that there are different stacks in other accounts (because you'll always use the QMINTRO stack).

If your account is not saving the stack, create an item in the VOC named $COMMAND.STACK with an X in field 1. This is automatically added to new accounts when they are created, but may not be there for some old accounts (or it may have been deleted).

If you view the `stacks` folder from a file manager, you will find one stack in there for each user (that has entered that account). These are simply text lists which you can edit with a normal text editor. Note that you should be logged off from *OpenQM* before you edit these items in this manner.

5 Introduction to *QMQuery* and QM Dictionaries

5.1 What is *QMQuery*?

QMQuery is an ad-hoc reporting language that uses the definitions stored in the dictionary files to report on the data. The language contains elements for:

- data selection
- data sorting
- data grouping
- carrying out summary operations (totals, averages, percentages)
- formatting of data on output
- generating headers and footers on each page of output
- panning and scrolling of output when viewed on a monitor
- redirection of output to printers, text files, or to delimited files
- printer formatting (PCL printers only)

Most of these elements are optional. Therefore, you can start with a simple statement and then gradually add to it as you learn more of the language. This makes *QMQuery* relatively simple to learn and use.

QMQuery is intimately associated with QM dictionaries. Therefore, you can't really learn *QMQuery* without gaining a good understanding of what dictionary items do and how they are constructed.

This section aims to work through the key elements of *QMQuery* while introducing dictionaries to you. By the end of the section, you should be able to write comprehensive *QMQuery* statements and write dictionary items to use with those *QMQuery* statements.

5.2 Anatomy of a *QMQuery* Statement

General syntax

QMQuery statements always follow a general syntactical form. This is:

```
verb {DICT} filename  {USING {DICT} filename}  {selection.clause }  {sort.clause}
{display.clause}  {record.id...}  {FROM select.list.no}  {TO select.list.no}
```

Given that most of the elements are optional, the simplest form of a *QMQuery* statement is simply:

```
verb  filename
```

You have already used some of these commands, such as:

```
SORT VOC
SORT XRATES
SORT DICT XRATES
```

The most common verbs are LIST and SORT. Despite the difference in name, both verbs will sort the data. However, the LIST verb will only do this when a sort clause is included in the statement, while the SORT verb will always sort the data. The key difference is that the SORT verb appends a final sort by the item-id. Therefore, the command:

```
SORT INVOICES BY INV.DATE
```

is equivalent to:

```
LIST INVOICES BY INV.DATE BY @ID
```

In most of this section, we will only use the SORT verb. Some other verbs will be introduced later in the section (notably SELECT, SSELECT, and SEARCH), but for others you should refer to the *OpenQM* documentation. These verbs are also listed in the Quick Reference at the back of this book.

5.3 Selection clause

As its name implies, the selection clause restricts the set of items to be reported on to those matching one or more criteria.

The general format of the selection clause is:

```
WITH {EVERY} condition {AND | OR condition}
```

where:

condition is:	*field operator value*
or:	*field1 operator field2*

and operator is one of the terms in the following table:

Operator	Synonym	Synonym	Synonym	Synonym
EQ	=	EQUAL		
NE	#	NOT	<>	><
LT	<	LESS	BEFORE	
LE	<=	=<		
GT	>	GREATER	AFTER	
GE	>=	=>		
LIKE	MATCHES	MATCHING		
UNLIKE	NOT.MATCHING			
SAID	SPOKEN	~		
NO				
BETWEEN				

Note that some operators have multiple synonyms. *OpenQM* does not care which synonym you use – it offers you a choice for your convenience.

5.3.1 Creating a dictionary item for use in selecting data

Say we want to display the exchange rates for the year 2013. We could write a *QMQuery* statement like:

```
SORT XRATES WITH YEAR EQ 2013
```

Traditionally, multi-value databases have required that the comparison value in the expression be enclosed in quotes. So, in other databases, we would need to write:

```
SORT XRATES WITH YEAR EQ “2013”
```

OpenQM makes these quotes optional. You can choose to include them or omit as you please. But if you are going to use one of the other multi-value databases, it would be good practice to quote your comparison values.

Running this commad, *OpenQM* responds with:

```
YEAR is not a field name or expression
```

This means that *OpenQM* has not been able to find a definition of the word YEAR – it does not appear in either the file dictionary or the VOC of the account. Therefore, we need to define the word YEAR so that *QMQuery* understands what we mean.

To define YEAR, we use the MODIFY editor.

```
MODIFY DICT XRATES
```

Type in YEAR at the ID prompt, and define the item as follows:

ID	Type	Loc	Conv	Name	Format	S/M	Assoc
YEAR	I	OCONV(@ID, ‘DY’)		Year	4R	S	

Once you have filed the item, exit from the MODIFY editor and retry the *QMQuery* statement:

```
SORT XRATES WITH YEAR EQ "2013"
XRATES....
16440
16441
16444
16445
```

Let’s add the date so that we can see if we really have 2013 information:

```
SORT XRATES WITH YEAR EQ "2013" DATE
XRATES....   Date........
16440         03 JAN 2013
16441         04 JAN 2013
16444         07 JAN 2013
16445         08 JAN 2013
```

That looks good. Let’s see how we did that:

In the ‘Type’ field, we entered ‘I’. ‘I’ stands for indirect, and it means that this is a calculated field. In *OpenQM* jargon, this type of dictionary item is known as an I-type.

In the ‘Loc’ field, we entered the expression used to calculate the year. This expression was:

```
        OCONV(@ID, ‘DY’)
```

OCONV is a function from the *QMBasic* programming language. It is the output conversion function. We have already come across conversions – those are the expressions that go into the CONV field of dictionary items.

In this case, the expression says apply an output conversion of ‘DY’ to the @ID field. You will recall that a ‘D’ conversion is a generic data conversion. In fact, any conversion code starting with a ‘D’ is a date conversion, and ‘DY’ means return the year of the passed (internal) date. We can see from the dates displayed that all have a year of 2013.

If all we have done is apply an output conversion, why didn’t we do this by specifying a ‘DY’ conversion in the CONV field? Well, let’s try that and see what happens.

Create a dictionary item YEARX as follows:

```
ID      Type  Loc                 Conv      Name        Format    S/M    Assoc
YEARX   D     0                   DY        Year        4R        S
```

Let’s see what output we get from it:

```
SORT XRATES DATE YEAR YEARX
XRATES....    Date........   Year   Year
15346          05 JAN 2010   2010   2010
15347          06 JAN 2010   2010   2010
15348          07 JAN 2010   2010   2010
15349          08 JAN 2010   2010   2010
```

So, it displays the year correctly, just like our I-type item did. What about selecting data?

```
SORT XRATES WITH YEARX EQ “2013” DATE YEAR YEARX
0 record(s) listed
```

It didn’t select any data. Why not?

Let’s think about this. An output conversion in the CONV field is applied just before the data is displayed. However, when we are selecting and sorting the data, we are doing so in its internal data format – so it is still a date. Let’s check that:

```
SORT XRATES WITH YEARX GE "01 JAN 2013" DATE YEAR YEARX
XRATES....    Date........   Year   Year
16440          03 JAN 2013   2013   2013
16441          04 JAN 2013   2013   2013
16444          07 JAN 2013   2013   2013
16445          08 JAN 2013   2013   2013
```

So, even though it is displaying a year value, its internal value is still an entire date. So we can select on it as a date, but not as a year. Let’s delete that dictionary item:

```
DELETE DICT XRATES YEARX
1 record(s) deleted
```

Let’s add some other date type dictionary items:

```
MODIFY DICT XRATES

ID       Type   Loc                    Conv    Name      Format   S/M    Assoc
MTHNO    I      OCONV(@ID, ‘DM’)               Month     2R       S
MTH      I      OCONV(@ID, ‘DMA[3]’)   MCT     Mth       3L       S
MONTH    I      OCONV(@ID, ‘DMA’)      MCT     Month     10L      S
DOM      I      OCONV(@ID, ‘DD’)               Day       3R       S
DOW      I      OCONV(@ID, ‘DW’)               Day       3R       S
DAY      I      OCONV(@ID, ‘DWA[3]’)   MCT     Day       3L       S
```

What does all this mean? Well, lets start by seeing what output they generate:

```
SORT XRATES DATE MTHNO MTH MONTH DOM DOW DAY SAMPLE 5
XRATES....   Date........   Month   Mth   Month.....   Day   Day   Day
15487         26 MAY 2010      05   May   May           26     3   Wed
15654         09 NOV 2010      11   Nov   November      09     2   Tue
15670         25 NOV 2010      11   Nov   November      25     4   Thu
15759         22 FEB 2011      02   Feb   February      22     2   Tue
15801         05 APR 2011      04   Apr   April         05     2   Tue

Sample of 5 record(s) listed
```

MTHNO returns the month number of the year. It does this by applying a 'DM' conversion to the @ID (date) field.

MONTH returns the full name of the month, while MTH returns an abbreviated month name. In both cases, the central bit of the conversion is 'DMA' where the 'A' means return an alphabetic value. The MTH dictionary item then returns only 3 characters of this value.

DOM returns the day of month by applying a 'DD' conversion to the date, while DOW returns the day of week by applying a 'DW' conversion. The DAY dictionary item returns the day name by using a 'DWA' conversion.

Note that the dictionary items that return alphabetic values have a further conversion in the CONV field. This conversion is 'MCT' which capitalises the first letter of every word with the rest of each word in lower case. In recent versions of *OpenQM*, we could also have used 'MCS' which capitalises the first word of a sentence, with the rest of the sentence in lower case.

Note that while these secondary conversions make the output look better than the default all-capitals values, they make selection by these dictionary items more difficult.

Selection by month number:

```
SORT XRATES WITH MTHNO EQ "5" DATE MTHNO MTH MONTH DOM DOW DAY SAMPLE 5
XRATES....   Date........   Month   Mth   Month.....   Day   Day   Day
15487         26 MAY 2010      05   May   May           26     3   Wed
15849         23 MAY 2011      05   May   May           23     1   Mon
16195         03 MAY 2012      05   May   May           03     4   Thu
16564         07 MAY 2013      05   May   May           07     2   Tue
16580         23 MAY 2013      05   May   May           23     4   Thu

Sample of 5 record(s) listed
```

Note that we didn't specify the leading zero on the month number. This could fail in some other multi-value implementations as they could require that the leading zero be present in the comparison value.

Selection by abbreviated month:

```
SORT XRATES WITH MTH EQ "May" DATE MTHNO MTH MONTH DOM DOW DAY SAMPLE 5
Sample of 0 record(s) listed
```

This will not return anything regardless of the capitalisation of "May". If we take the 'MCT' conversion out of the dictionary item, then the following statement works:

```
SORT XRATES WITH MTH EQ "MAY" DATE MTHNO MTH MONTH DOM DOW DAY SAMPLE 5
XRATES....   Date........   Month   Mth   Month.....   Day   Day   Day
15487         26 MAY 2010      05   MAY   May           26     3   Wed
15849         23 MAY 2011      05   MAY   May           23     1   Mon
16195         03 MAY 2012      05   MAY   May           03     4   Thu
16564         07 MAY 2013      05   MAY   May           07     2   Tue
16580         23 MAY 2013      05   MAY   May           23     4   Thu

Sample of 5 record(s) listed
```

With the 'MCT' conversion in place, we need to do a case-insensitive comparison:

```
SORT XRATES WITH MTH EQ NO.CASE "MAY" DATE MTHNO MTH MONTH DOM DOW DAY SAMPLE 5
XRATES....    Date........   Month    Mth    Month.....    Day    Day    Day
15487          26 MAY 2010      05    May    May            26      3    Wed
15849          23 MAY 2011      05    May    May            23      1    Mon
16195          03 MAY 2012      05    May    May            03      4    Thu
16564          07 MAY 2013      05    May    May            07      2    Tue
16580          23 MAY 2013      05    May    May            23      4    Thu

Sample of 5 record(s) listed
```

In practice, we probably aren't going to make a selection on a literal month name – we would tend to use the month number for that. But we might want to select on someone's name, and we need to be aware of the impact of conversion codes on the selection process.

A full list of conversion codes can be found in the *OpenQM* documentation, but some alternatives and their output is shown below for date value 16607 (Wednesday, 19 June 2013).

Code	**Output**	
'D'	19 JUN 2013	
'D2'	19 JUN 13	
'D2/'	19/06/13	(or 06/19/13)
'D4/'	19/06/2013	(or 06/19/2013)
'D-YMD'	2013-06-19	
'DW'	3	
'DWA'	WEDNESDAY	
'DWAL'	Wednesday	
'DMA'	JUNE	
'DMAL'	June	
'DMAL[3]'	Jun	

You will realise by now that the text you enter in the 'Name' field of the dictionary is what appears in the column heading. Similarly, what you enter in the format field defines the width of the column – with some exceptions.

The MTHNO dictionary item specifies a format of 2R which means right-justify the field with a field width of 2 characters. However, the actual output displays the column heading fully, meaning that the actual field width is 5 characters wide.

Both the format codes and the conversion codes contain many options allowing powerful formatting of output from dictionaries. We'll cover some more of these later, but you need to read the manuals and help files to gain a full picture of their capabilities.

Many of the selection statements above have used the EQ operator (or = operator) in a comparison test. Many multi-value implementations assume an EQ operator if no operator is present. *OpenQM* can support this behaviour if the OPTION PICK.IMPLIED.EQ is set. Typically, this option will be set in the LOGIN item of the account. If this option is set, then the comparison value must be enclosed in double-quotes as shown above.

```
SORT XRATES WITH MTHNO "3" DATE MTHNO MTH MONTH DOM DOW DAY SAMPLE 5
Sample of 0 record(s) listed
'3' not found

OPTION PICK.IMPLIED.EQ

SORT XRATES WITH MTHNO "3" DATE MTHNO MTH MONTH DOM DOW DAY SAMPLE 5
XRATES....    Date........   Month    Mth    Month.....    Day    Day    Day
15408          08 MAR 2010      03    Mar    March          08      1    Mon
15412          12 MAR 2010      03    Mar    March          12      5    Fri
16143          12 MAR 2012      03    Mar    March          12      1    Mon
16147          16 MAR 2012      03    Mar    March          16      5    Fri
16517          21 MAR 2013      03    Mar    March          21      4    Thu

Sample of 5 record(s) listed
```

```
OPTION PICK.IMPLIED.EQ OFF
```

You decide for yourself whether you want this option on or off.

5.3.2 Multiple selection criteria

Often, we need to select on more than one criteria. For example, we want to display all dates in 2013 when the exchange rate has been below 80 US cents to the NZ dollar.

```
sort xrates with year eq "2013" and with usd lt "0.8000" date usd
XRATES....   Date........   US Dollar
16594          06 JUN 2013     0.7962
16595          07 JUN 2013     0.7994
16598          10 JUN 2013     0.7849
16599          11 JUN 2013     0.7890
16600          12 JUN 2013     0.7894
16601          13 JUN 2013     0.7951

6 record(s) listed
```

Note that *OpenQM* recognises the statement even though everything is in lower case.

In this statement, the WITH keyword is specified a second time after the AND. In *OpenQM*, this is not strictly necessary – but other multi-value environments require the second WITH to be included.

It is generally a good idea to make your statements as compatible with other multi-value environments as possible. This is because you might have to work on one of these other systems, and if you aren't familiar with their syntax, you will find their query language to be quite restrictive.

In practical usage, you can have as many selection criteria as you like. However, if you have a lot of selection criteria, you may find it better to use multiple SELECT statements (see Section 6.2) for selecting the data, followed by LIST or SORT statements to display the data.

There is another form of selection, where two criteria are applied to the same element. For example, display dates where the exchange rate is between 86 and 87 US cents:

```
SORT XRATES WITH USD GE "0.86" AND LE "0.87" DATE USD
XRATES....   Date........   US Dollar
15909          22 JUL 2011     0.8622
15912          25 JUL 2011     0.8646
15913          26 JUL 2011     0.8628
15921          03 AUG 2011     0.8623
15922          04 AUG 2011     0.8656
16539          12 APR 2013     0.8634

6 record(s) listed
```

We could also have used the BETWEEN operator to achieve the same selection:

```
SORT XRATES WITH USD BETWEEN "0.86" "0.87" DATE USD
```

The BETWEEN operator returns true if the field is greater than or equal to the first value specified, and less than or equal to the second value specified.

5.3.3 Comparison against a database value

The selection criteria used so far have compared a value in the database with a value we specify in the selection clause. There is another type of selection where we want to compare a database value against another database value.

To demonstrate this, we'll use the interest rates file that you imported in Section 4.4.2. Now, we normally expect longer term interest rates to be higher than short term interest rates, but this isn't always the case. We can use the query language and the selection criteria to find those dates where short term rates (30 days) are higher than longer term (90 days) rates.

```
SORT IRATES WITH DAYS30 GT DAYS90 DATE DAYS30 DAYS90
IRATES....   Date.......   30 Day Bank Bill   90 Day Bank Bill
15389        17 FEB 2010               2.72               2.70
15390        18 FEB 2010               2.72               2.68
15391        19 FEB 2010               2.72               2.69
15755        18 FEB 2011               3.18               3.17
15760        23 FEB 2011               3.04               3.01
```

5.3.4 Direct identification of items

There is another way to select items if you know their item ID. This corresponds to the {record.id ...} element shown in the general syntax of a *QMQuery* statement shown in section 5.2.

Using the first few item-ids from the above statement, we could write:

```
SORT IRATES '15389''15390''15391' DATE DAYS90
IRATES....   Date.......   90 Day Bank Bill
15389        17 FEB 2010               2.70
15390        18 FEB 2010               2.68
15391        19 FEB 2010               2.69

3 record(s) listed
```

While this seems fairly awkward, there are situations where you know the item-ids that you want, and this becomes an easy method to extract the desired records.

Note (once again) that while the item-ids above are shown to be single-quoted, *OpenQM* does not require this – but other multi-value databases do.

Note the following points about selection using a list of item-id's:

- long lists of item-id's will be cumbersome
- the item-id will need to comprise "meaningful" data if you are to have any chance of knowing it.

5.4 Sort clause

So far, we have been using the SORT verb, but only sorting the records into the default order. There are many cases where we need to sort the data into some other order. For example, we want to know when we had the lowest exchange rate against the US dollar:

```
SORT XRATES BY USD DATE USD
XRATES....   Date........   US Dollar
15500         08 JUN 2010      0.6584
15501         09 JUN 2010      0.6668
15488         27 MAY 2010      0.6675
15482         21 MAY 2010      0.6680
15487         26 MAY 2010      0.6689
15486         25 MAY 2010      0.6702
```

Of course, this dataset is limited to dates later than the start of 2010. If you load some of the historic data series available from the RBNZ website, you will find that the NZ Dollar reached a low of 0.3922 against the US Dollar in November 2000.

This listing shows the data sorted into ascending order. What about descending order? This uses a BY.DSND (note the period in the middle of the word) modifier instead of BY. *OpenQM* will also accept BY-DSND for compatibility with other multi-value environments.

```
SORT XRATES BY.DSND USD DATE USD
XRATES....   Date........   US Dollar
15919         01 AUG 2011      0.8822
15920         02 AUG 2011      0.8760
15914         27 JUL 2011      0.8722
15915         28 JUL 2011      0.8721
15916         29 JUL 2011      0.8702
```

Multiple sort values can be specified, and ascending and descending sorts freely mixed. Sorting will occur in the order specified within the statement. Consider:

```
SORT XRATES WITH USD LT "0.8450" BY.DSND USD BY DATE DATE USD
XRATES....   Date........   US Dollar
16545         18 APR 2013      0.8448
15906         19 JUL 2011      0.8447
16424         18 DEC 2012      0.8447
16448         11 JAN 2013      0.8447
16486         18 FEB 2013      0.8445
```

This sorts the selected data into descending exchange rate order and then applies a secondary ascending sort to the date. This means that where there a multiple entries for a given exchange rate, these will be sorted into ascending date order. So, the three entries for 0.8447 are shown with the 2011 entry first, followed by 2012, and then 2013.

Try changing the BY.DSND sort into an ascending sort to make sure that the data output is sorted correctly.

5.5 Display clause

The display clause tells *QMQuery* what to display and consists of a list of dictionary items along with optional processing and formatting codes. We have used simple display clauses throughout this section to see the results of the selection and sort clauses.

To see the dictionary items that are available for any given file, simply type:

```
SORT DICT filename
```

```
SORT DICT XRATES
@ID.........   TYPE   LOC...........   CONV..   NAME........   FORMAT   S/M   ASSOC...
DATE           D      0                D        Date           12R      S
@ID            D      0                         XRATES         10L      S
YEARX          D      0                DY       Year           4R       S
USD            D      1                MR44,    US Dollar      7R       S
GBP            D      2                MR44,    UK Pound       7R       S
AUD            D      3                MR44,    Aus Dollar     7R       S
JPY            D      4                MR22,    Jap Yen        7R       S
EUR            D      5                MR44,    Euro           7R       S
CAD            D      6                MR44,    Canada Dolla   7R       S
                                                r
KRW            D      7                MR22,    SKorea Won     7R       S
CNY            D      8                MR44,    Chinese Yuan   7R       S
MYR            D      9                MR44,    Malay Ringgi   7R       S
                                                t
HKD            D      10               MR44,    HK Dollar      7R       S
IDR            D      11               MR22,    Indonesia Ru   9R       S
                                                piah
THB            D      12               MR44,    Thai Baht      8R       S
SGD            D      13               MR44,    Singapore Do   7R       S
                                                llar
TWD            D      14               MR44,    Taiwan Dolla   7R       S
                                                r
```

```
DOM            I      OCONV(@ID,                Day            3R       S
                      'DD')
MTHNO          I      OCONV(@ID,                Month          2R       S
                      'DM')
MONTH          I      OCONV(@ID,      MCT       Month          10L      S
                      'DMA')
MTH            I      OCONV(@ID,      MCT       Mth            3L       S
                      'DMA[3]')
DOW            I      OCONV(@ID,                Day            3R       S
                      'DW')
DAY            I      OCONV(@ID,      MCT       Day            3L       S
                      'DWA[3]')
YEAR           I      OCONV(@ID,                Year           4R       S
                      'DY')

24 record(s) listed
```

At its simplest, a display clause is simply a list of dictionary items for the file:

```
SORT XRATES DATE USD GBP AUD
XRATES....   Date........   US Dollar   UK Pound   Aus Dollar
15346         05 JAN 2010      0.7344     0.4564       0.8046
15347         06 JAN 2010      0.7343     0.4591       0.8056
15348         07 JAN 2010      0.7378     0.4606       0.8020
15349         08 JAN 2010      0.7325     0.4597       0.7980
```

In this example, everything after the word XRATES makes up the display clause.

5.5.1 Modifiers

Does the output of the ID annoy you? We can suppress the display of the item ID by using the keyword: ID.SUP or ID-SUPP

You will note that *OpenQM* often has two very similar words which do the same thing. The native dialect used by *OpenQM* uses periods in these words, but words with hyphens are supported for compatibility with other multi-value environments.

```
SORT XRATES DATE USD GBP AUD ID.SUP
Date........   US Dollar   UK Pound   Aus Dollar
 05 JAN 2010      0.7344     0.4564       0.8046
 06 JAN 2010      0.7343     0.4591       0.8056
 07 JAN 2010      0.7378     0.4606       0.8020
 08 JAN 2010      0.7325     0.4597       0.7980
```

That looks a better presentation.

OpenQM has a range of keywords that modify the way the output is displayed. Some of these are listed below:

Keyword	Synonyms	Purpose
COL.HDR.SUPP	COL.HDR.SUP COL-HDR-SUPP COL-HDR-SUP	Suppresses page and column headings
COL.SUP		Suppresses column headings
COUNT.SUP		Suppresses the count of records selected
DBL.SPC	DBL-SPC	Double-spaces the output
DET.SUP	DET-SUPP	Suppresses detail lines in reports so that only totals are displayed
HDR.SUP	HDR-SUPP SUPP	Suppresses the default page heading
ID.ONLY	ONLY	Ignore the display clause and only list the ID
ID.SUP	ID-SUPP	Suppresses the display of the ID
LPTR		Send output to the printer
NEW.PAGE		Forces every record to display on a new page

You could try adding these to your queries to see what happens – but sometimes, they are best used in specific circumstances. On our reports so far, DET.SUP will suppress all the output lines, but will be useful with more advanced reports. Using LPTR may not work well until we define a printer.

5.5.2 Grouping records

Often, we want to display information over a period of time, but be able to break that report into blocks of information. Therefore, each year, product, or customer will be separated from other years, products, or customers.

This grouping is often (but not always) accompanied by a summary of information relating to that group – such as the total of sales for a given year or customer. This section will cover how to group information, while the next section will cover how to summarise the information.

Grouping data involves two steps – sorting the data so that related items appear together, and then placing a break in the output to visually group the items. Sorting has already been covered, so we the new concept is breaking the data into groups.

The keyword that *OpenQM* uses to break the data into groups is BREAK.ON (or BREAK-ON), and has the following general format:

```
BREAK.ON  { "options" }  dict-item
```

or:

```
BREAK.ON  dict-item  { "options" }
```

While two ways of entering this expression are shown, you can actually only use one of them. Which one you can use is determined by the OPTION settings in place. When we created the QMINTRO account (see Section 4.1), it was suggested that the line:

```
OPTION PICK
```

be inserted in the LOGIN entry. If that option is in effect (or more specifically, the PICK.BREAKPOINT option – which is activated by OPTION PICK), then you must use the second format of the BREAK.ON keyword. To see which options you have in effect, type:

```
OPTION
```

Given that OPTION PICK was specified earlier, instructions here will use the PICK format. However, be aware that your installation may be using the Information format. In that case you would need to modify these instructions.

A simple statement breaking the exchange rate data into groups would be:

```
SORT XRATES BREAK.ON YEAR BREAK.ON MONTH DATE USD ID.SUP
Year  Month.....  Date........  US Dollar
2010  January      05 JAN 2010     0.7344
2010  January      06 JAN 2010     0.7343
etc
2010  January      28 JAN 2010     0.7062
2010  January      29 JAN 2010     0.7048
      **

2010  February     01 FEB 2010     0.7006
2010  February     02 FEB 2010     0.7085
2010  February     03 FEB 2010     0.7131
2010  February     04 FEB 2010     0.7021
```

Most of the January entries have been removed to reduce the size of the listing.

At the end of each group, two asterisks are placed in the column of the field causing the break. The command said to break on year and month. We can see the asterisks in the month column, and if we followed the listing to the point where the year changes, we would see the asterisks appear there too.

```
SORT XRATES BREAK.ON YEAR BREAK.ON MONTH DATE USD ID.SUP
Year   Month.....   Date........   US Dollar
etc
2010   December      30 DEC 2010      0.7684
2010   December      31 DEC 2010      0.7706
       **
  **

2011   January       05 JAN 2011      0.7665
2011   January       06 JAN 2011      0.7580
```

What say we don't want the asterisks to appear. There are a couple of ways we can get rid of them. Consider:

```
BREAK.ON MONTH " "
```

```
BREAK.ON MONTH "'L'"
```

The first method shown above simply replaces the asterisks with the text between the quotation marks. In this case, that text is a single space, so nothing is displayed.

The second method suppresses the break line. This has the effect of moving the groups of data closer together:

```
SORT XRATES BREAK.ON YEAR "'L'" BREAK.ON MONTH "'L'" DATE USD ID.SUP
Year   Month.....   Date........   US Dollar
2010   January       05 JAN 2010      0.7344
2010   January       06 JAN 2010      0.7343
etc
2010   January       28 JAN 2010      0.7062
2010   January       29 JAN 2010      0.7048

2010   February      01 FEB 2010      0.7006
2010   February      02 FEB 2010      0.7085
```

You can also use the 'O' option to only show the break value when it first occurs. This is another way of visually separating the groups:

```
SORT XRATES BREAK.ON YEAR "'L'" BREAK.ON MONTH "'LO'" DOM USD ID.SUP
Year   Month.....   Day   US Dollar
2010   January       05      0.7344
2010                 06      0.7343
etc
2010                 28      0.7062
2010                 29      0.7048

2010   February      01      0.7006
2010                 02      0.7085
```

We didn't specify an 'O' option for the year break because too much data would pass between the each break, and we would be left wondering which year we were looking at.

We could further add a 'P' option to the break. This would cause a page break to occur whenever the break-point was reached. In other words, once all the January data was displayed, it would pause until you pressed enter before displaying February on a fresh screen.

The introduction to this section noted that the first thing we had to was sort the data. But none of these commands actually specifies a sort criteria – although they use the SORT verb. The reason that no sort criteria is specified is that sorting by the ID (which is what the SORT verb does after all other sort criteria have been implemented) puts the records in ascending date order – which is what we want. Therefore, no explicit sort order needs to be specified.

The BREAK.ON clause contains a number of other useful options, but we need to explore these in combination with the creation of summary data. So, we'll look at that now.

5.5.3 Generating summary information

Typical summary information that is generated from grouped data includes totals, averages, and percentages. *OpenQM* provides keywords to calculate these summaries as well as a few others. The table below shows the available summary keywords, and their synonyms:

Keyword	**Synonyms**	**Purpose**
AVERAGE	AVG	Averages the specified field
CUMULATIVE		Reports the cumulative value of the field
ENUMERATE	ENUM	Counts the values in the field
MAX		Reports the maximum value of the field in the group
MEDIAN		Reports the median value of the field in the group
MIN		Reports the minimum value of the field in the group
PERCENTAGE	PERCENT PCT %	Reports the field value as a percentage of the field total
TOTAL		Reports the total of the field

The general format used by these keywords is as follows:

```
keyword dict-item { field qualifiers }
```

The field qualifiers have a range of functions. Some of these modify the way the keywords operate (e.g. NO.NULLS tells the AVERAGE keyword to ignore null items), while others specify how to display the results (e.g. CONV, FMT, COL.HDG).

MIN, MAX, MEDIAN, AVG

A simple set of summary data from the exchange rate file would be:

```
SORT XRATES BREAK.ON YEAR "'UV'" BREAK.ON MONTH "'UV'" DOM MIN USD MEDIAN USD AVG USD MAX USD
ID.SUP
Year   Month.....   Day   US Dollar   US Dollar   US Dollar   US Dollar
2010   January      05       0.7344      0.7344      0.7344      0.7344
2010   January      06       0.7343      0.7343      0.7343      0.7343
etc
2010   January      28       0.7062      0.7062      0.7062      0.7062
2010   January      29       0.7048      0.7048      0.7048      0.7048
                          ---------   ---------   ---------   ---------
       January               0.7048      0.7344      0.7277      0.7427

2010   February     01       0.7006      0.7006      0.7006      0.7006
2010   February     02       0.7085      0.7085      0.7085      0.7085
```

This listing tells us that the average exchange rate against the US dollar in January of 2010 was approximately 72.8 US cents to the NZ dollar, with a minimum value during the month of about 70.5 US cents and a maximum of about 74.3 US cents. The median value was higher than the average at 73.4 US cents.

Note that the BREAK-ON clause has options of ‘UV’. The ‘V’ tells *OpenQM* to display the value of the break field in the break line, while the ‘U’ means display a row of underline characters between the group and the break line.

Note also that these option codes are contained within single quote marks inside a pair of double quote marks. This is explained by the general format of these BREAK.ON options:

```
"text 'codes'"
```

The text is used to replace the asterisks on the break line (covered in the previous section), while the codes provide other instructions to *OpenQM*. The single quote marks are necessary to distinguish the codes from the text. If the text itself contains a single quote mark, then this should be entered as two single quote marks.

Suppressing detail lines

So far, all queries have used displayed all of the selected records. However, we frequently want to display only summary information without showing the individual record detail. Once again, *OpenQM* has another keyword to enable this:

```
      DET.SUP      or          DET-SUPP
```

For example, to simply show the monthly summary data for the same query as shown above, we would use:

```
SORT XRATES WITH YEAR EQ "2010" BREAK.ON MONTH DOM MIN USD MEDIAN USD AVG USD MAX USD DET.SUP
Month.....    Day   US Dollar   US Dollar   US Dollar   US Dollar
January       29       0.7048      0.7344      0.7277      0.7427
February      26       0.6823      0.6973      0.6974      0.7131
March         31       0.6862      0.7030      0.7033      0.7145
April         30       0.7040      0.7110      0.7124      0.7245
May           31       0.6675      0.7085      0.6992      0.7307
June          30       0.6584      0.6964      0.6928      0.7133
July          30       0.6846      0.7115      0.7111      0.7331
August        31       0.7015      0.7124      0.7154      0.7335
September     30       0.6998      0.7282      0.7259      0.7393
October       29       0.7337      0.7513      0.7501      0.7584
November      30       0.7458      0.7726      0.7727      0.7952
December      31       0.7360      0.7490      0.7504      0.7706

                       0.6584      0.7152      0.7215      0.7952

253 record(s) listed
```

Note that we've dropped the ID.SUP from the above statement, because it is implied by the use of DET.SUP. You can leave it there if you want – it makes no difference.

There is another curiosity with the above listing. The DOM field has generated an entry on the break line even though it does not do this in the detail listing. This is actually the last 'day of month' value before the break.

Formatting column headings

Now, while this shows the summarised data as we want, it is pretty confusing to have each column headed "US Dollar". We really need to change the headings of the columns to make it clear which column represents what data. This is where we use some of the field qualifiers referred to above:

```
SORT XRATES WITH YEAR EQ "2010" BREAK.ON MONTH DOM MIN USD COL.HDG "Min USD" MEDIAN USD COL.HDG
"Med USD" AVG USD COL.HDG "Avg USD" MAX USD COL.HDG "Max USD" DET.SUP
Month.....    Day   Min USD    Med USD    Avg USD    Max USD
January       29     0.7048     0.7344     0.7277     0.7427
February      26     0.6823     0.6973     0.6974     0.7131
March         31     0.6862     0.7030     0.7033     0.7145
April         30     0.7040     0.7110     0.7124     0.7245
May           31     0.6675     0.7085     0.6992     0.7307
June          30     0.6584     0.6964     0.6928     0.7133
July          30     0.6846     0.7115     0.7111     0.7331
August        31     0.7015     0.7124     0.7154     0.7335
September     30     0.6998     0.7282     0.7259     0.7393
October       29     0.7337     0.7513     0.7501     0.7584
November      30     0.7458     0.7726     0.7727     0.7952
December      31     0.7360     0.7490     0.7504     0.7706

                     0.6584     0.7152     0.7215     0.7952

253 record(s) listed
```

This uses the COL.HDG keyword to supply a more appropriate column heading for the maximum, minimum and average exchange rates. The format of the COL.HDG keyword is as follows:

```
field COL.HDG "text"
```

where field is the dictionary item[18] for which we wish to apply the new column heading, and text is the new column heading. The text must be enclosed in either single or double quotes. The text may also contain some formatting options.

While *OpenQM* allows a choice of either single or double quotes, a standard convention would be to use double quotes (as shown above), and to use single quotes to delimit options within the text.

Three formatting codes are allowed for COL.HDG. These are:

Code	**Purpose**
'L'	Breaks the text into multiple lines
'R'	Right-aligns the column heading
'X'	Suppresses the dot fillers normally inserted into column headings

To illustrate the use of the these options, consider:

```
SORT XRATES WITH YEAR EQ "2010" BREAK.ON MONTH MIN USD COL.HDG "'RX'Min'L'USD" AVG USD COL.HDG
"'R'Avg'L'USD" MAX USD COL.HDG "Max'L'USD" ID.SUP DET.SUP
Month.....        Min   ....Avg   Max....
                  USD   ....USD   USD....
January        0.7048    0.7277    0.7427
February       0.6823    0.6974    0.7131
March          0.6862    0.7033    0.7145
April          0.7040    0.7124    0.7245
May            0.6675    0.6992    0.7307
June           0.6584    0.6928    0.7133
July           0.6846    0.7111    0.7331
August         0.7015    0.7154    0.7335
September      0.6998    0.7259    0.7393
October        0.7337    0.7501    0.7584
November       0.7458    0.7727    0.7952
December       0.7360    0.7504    0.7706

               0.6584    0.7215    0.7952

253 record(s) listed
```

This uses the 'L' option to split the column headings into two lines. It then uses the 'R' and 'X' options on the first column to right-justify the heading and suppress the dot fillers. The second column only uses 'R' to right-justify, while the final column doesn't use either of these options.

This demonstrates that column headings can be broken into multiple lines. Of course, it would be inconvenient if we always had to use a COL.HDG keyword to do this – there must be a way to do this directly in the dictionary. And of course, there is:

Consider:

```
SORT IRATES WITH YEAR EQ "2010" BREAK.ON MONTH MIN OVERNIGHT AVG OVERNIGHT MAX OVERNIGHT ID.SUP
DET.SUP
YEAR is not a field name or expression
```

It seems that we haven't defined YEAR in the IRATES dictionary yet. And of course, there were a number of other dictionary items we defined in XRATES that are not in IRATES. Now, because these dictionaries work off the ID of each item, and because the ID's used in the IRATES file are identical to those used in the XRATES file, we can just copy these items from the XRATES dictionary to the IRATES dictionary.

18 COL.HDG can also apply to a calculated field that is specified entirely in the *QMQuery* statement rather than in a dictionary item. Creation of this type of field is covered later.

```
COPY FROM DICT XRATES TO DICT IRATES YEAR MTHNO MONTH MTH DOM DOW DAY
7 record(s) copied.
```

The COPY command has a number of formats and options. See the documentation or online help for details.

Now, let's try again:

```
SORT IRATES WITH YEAR EQ "2010" BREAK.ON MONTH MIN OVERNIGHT AVG OVERNIGHT MAX OVERNIGHT ID.SUP
DET.SUP
Month.....   Overnight Cash Rate   Overnight Cash Rate   Overnight Cash Rate
January                                           1.78                  2.71
February                                          2.20                  2.48
March                                             2.04                  2.60
April                       2.25                  2.28                  2.37
May                         2.25                  2.32                  2.44
June                                              2.37                  2.67
July                                              2.56                  3.05
August                                            2.75                  3.05
September                                         2.52                  3.21
October                                           2.79                  3.20
November                                          2.11                  3.20
December                    2.75                  2.88                  3.03

                                                  2.39                  3.21

253 record(s) listed
```

Clearly, the column heading is much wider than the actual data. It would be much better if the heading were split between the words "Overnight" and "Cash".

Go back to the MODIFY editor, and display the OVERNIGHT dictionary item. We want to modify line 4 (the NAME), so type in 4, and the heading will be displayed at the bottom of the screen. Use the arrow keys to move the cursor to the space character, and delete it using the Delete key. Now, hold down the Ctrl key and press Q. Release the Ctrl key. You will note the *OpenQM* is prompting you with "Quote char" at the bottom of the screen. Press V. You will now see another character appear where the space character used to be. The actual character you see will depend on the character set in use by your computer, but a y with two dots over the top is common (ÿ). Press enter, and your modified column heading will appear in the main dictionary display. Type in FI to file the item, and enter to return to the colon prompt.

What have we done? We have inserted a Value Mark into the heading. This should act to split the column heading into two lines.

```
SORT IRATES WITH YEAR EQ "2010" BREAK.ON MONTH MIN OVERNIGHT AVG OVERNIGHT MAX OVERNIGHT ID.SUP
DET.SUP
Month.....   Overnight  Overnight  Overnight
             Cash Rate  Cash Rate  Cash Rate
January                      1.78       2.71
February                     2.20       2.48
March                        2.04       2.60
April             2.25       2.28       2.37
May               2.25       2.32       2.44
June                         2.37       2.67
July                         2.56       3.05
August                       2.75       3.05
September                    2.52       3.21
October                      2.79       3.20
November                     2.11       3.20
December          2.75       2.88       3.03

                             2.39       3.21

253 record(s) listed
```

Value marks are one of a number of system delimiters used by *OpenQM* and multi-value databases in general. Indeed the "value" in "Value Mark" is the origin of the term "multi-value". Value marks delimit the boundaries of each value of information in items (fields) so that we can store, retrieve, and manipulate multi-valued items.

Clearly, most of the other items in the IRATES dictionary can have their column headings split into two lines. Note that you can put more than one value mark in the heading to make 3 or 4 line headings, or even more if you so wish.

Now – there is something wrong with the above listing. The minimum column is showing a lot of null values. If we list the data for the overnight cash rate, we find that quite a number of days have no quoted value. This is a situation where we should use the NO.NULLS modifier:

```
SORT IRATES WITH YEAR EQ "2010" BREAK.ON MONTH MIN OVERNIGHT NO.NULLS AVG OVERNIGHT NO.NULLS MAX
OVERNIGHT ID.SUP DET.SUP
Month.....   Overnight   Overnight   Overnight
             Cash Rate   Cash Rate   Cash Rate
January           2.25        2.42        2.71
February          2.12        2.32        2.48
March             2.25        2.35        2.60
April             2.25        2.28        2.37
May               2.25        2.32        2.44
June              2.26        2.49        2.67
July              2.50        2.68        3.05
August            2.75        2.88        3.05
September         2.75        2.91        3.21
October           2.75        2.94        3.20
November          2.75        2.91        3.20
December          2.75        2.88        3.03

                  2.12        2.61        3.21

253 record(s) listed
```

We have now applied the NO.NULLS keyword to both the MIN and the AVG columns, but haven't bothered with the MAX column. This is because the presence of a null has no effect on what value is returned for a maximum. However, you can compare the two listings to see the other two columns clearly have been affected.

You may think that perhaps we should always use NO.NULLS – but it really depends on the data that you have, and what you are trying to measure. If you are trying to get the average income of a group, and some of that group have no income, then their zero or null income should be included as part of the average (or minimum) value.

Let's test the way that NO.NULLS works. If we sort the IRATES file and display the OVERNIGHT rate, we find that item 15361 (20 January 2010) does not display a value. If we now edit that item and store a zero in attribute 2, and then re-run the above query, we get:

```
SORT IRATES WITH YEAR EQ "2010" BREAK.ON MONTH MIN OVERNIGHT NO.NULLS AVG OVERNIGHT NO.NULLS MAX
OVERNIGHT ID.SUP DET.SUP
Month.....   Overnight   Overnight   Overnight
             Cash Rate   Cash Rate   Cash Rate
January           0.00        2.26        2.71
February          2.12        2.32        2.48
March             2.25        2.35        2.60
April             2.25        2.28        2.37
May               2.25        2.32        2.44
June              2.26        2.49        2.67
July              2.50        2.68        3.05
August            2.75        2.88        3.05
September         2.75        2.91        3.21
October           2.75        2.94        3.20
November          2.75        2.91        3.20
December          2.75        2.88        3.03

                  0.00        2.60        3.21

253 record(s) listed
```

Now, the query shows the minimum overnight interest rate in January was 0.00 per cent, and the average rate for the month is reduced to 2.26 per cent (down from 2.42). Clearly, a distinction is being made here between a zero value and an empty (null) value.

Going back to the example of average income above, this allows us to cope with the situation where some people have zero income, while others in the group decline to give us their income. Those with zero income would have a zero stored in their income field, while those who didn't tell us, have a null value. Using the AVG keyword with the NO.NULLS modifier will now give us the correct average *of those values we were given*.

Totalling data

Another keyword that is important in reporting summary information is TOTAL. We haven't used that yet because interest and exchange rate data isn't good for totalling. However, we also loaded a file named TEX.QCH that contains better data for totalling. Most of the dictionary items to do this have already been created. You can view these by typing:

```
SORT DICT TEX.QCH
```

Let's look at some of those dictionary items. The CTRY dictionary item looks like:

```
I
FIELD(@ID, '*', 2)

Ctry
2L
S
```

Before we proceed further, it is useful to know that the @ID field of TEX.QCH has a structure of:

```
YYYYQ * CC * HH
```

where YYYYQ is the year and quarter; CC is a country identifier; and HH is an HS code. There are no spaces between the elements. This structure will be further explained in Section 6.3.2.

The FIELD function extracts a field from a delimited string. In this case, we have specified that the source string is the @ID field, the delimiter is an asterisk ('*'), and we want the second field.

This expression could also have been written as:

```
OCONV(@ID, 'G1*1')

@ID[7,2]

@ID['*', 2, 1]
```

The first of these is termed a "Group conversion" which is analogous to the FIELD function, but uses a different syntax. The 'G' indicates a group conversion; the first '1' means skip one field; the asterisk is the delimiter character; and the final '1' means return one field.

The second alternative is simply a string extraction. It says return two characters from the @ID, starting at position 7. This relies on the components of the @ID being fixed in length, whereas the other versions extract the second field delimited by asterisk characters.

The third expression[19] looks like a string extraction because it uses the square brackets, but it is actually a group extraction. This statement says extract 1 group from @ID, delimited by the asterisk character, starting at group 2.

19 This syntax is not supported in the GPL version of OpenQM.

Which expression you use is up to you. But you need to be aware that there is more than one way to achieve a given end, and that other people may use a different expression than you would.

The YYYYQ and HS dictionary items are very similar to CTRY – except that they extract a different part of the @ID. The YEAR and QTR dictionaries take the field returned by YYYYQ and carry out substring extractions to get the year and quarter numbers.

We'll deal with the other dictionary items later.

Now, let's look at total exports for a year, broken into country destinations:

```
SORT TEX.QCH WITH YEAR EQ "2012" BY CTRY BREAK.ON CTRY TOTAL FOB DET.SUP
Ctry   ......FOB Value
AD                   0
AE         579,039,997
AF              78,642
etc
ZA         244,036,303
ZM              88,493
ZR                   0
ZW           1,763,241

        44,356,210,157

96824 record(s) listed
```

This query statement used the DET.SUP keyword to suppress the individual record detail, so that only the country totals are displayed. If you look at the count of records in the report, you will realise that a lot of records have been summarised in this report.

If we take the DET.SUP keyword out of the statement, then we find that many records actually contain no data:

```
SORT TEX.QCH WITH YEAR EQ "2012" BY CTRY BREAK.ON CTRY TOTAL FOB
TEX.QCH.....   Ctry   ......FOB Value
20121*AD*01    AD
20121*AD*02    AD
20121*AD*03    AD
```

We could exclude these records with:

```
SORT TEX.QCH WITH YEAR EQ "2012" AND WITH FOB GT "0" BY CTRY BREAK.ON CTRY TOTAL FOB DET.SUP
Ctry   ......FOB Value
AE         579,039,997
AF              78,642
AG             726,777
etc
ZA         244,036,303
ZM              88,493
ZW           1,763,241

        44,356,210,157

17023 record(s) listed
```

There are two things to note here. Firstly, the record count has shrunk from over 968,000 down to just over 17,000 – so we've excluded a lot of empty records. The second thing to note is that because we've excluded those records, not all countries appear in the listing. For example, country 'AD' (Andorra) is shown in the first listing but not the second. This means that New Zealand didn't export anything to Andorra during 2012.

This raises an important issue regarding database design. If we want to display information about an entity (country), then we need to record that data *even when that data is empty*. Or *the absence of data can be just as important as its presence*.

If we only recorded actual exports in this database, we would not be able to produce a listing of exports that included ALL countries. Someone who didn't have knowledge of our database wouldn't be sure whether we didn't export to that country in the period, or whether we simply hadn't recorded the data. So, there is value to data, even when the data is null.

In this database design, we store a lot of empty records. On the other hand, the user now has the choice of displaying all countries in a report, or only those countries for which there is genuine data. We can even answer the question "Which countries did New Zealand NOT export to in 2012?"

Change the entity to make the example more relevant to you. For example, in a hotel booking system, we want to record the status of all rooms in the hotel – not just those rooms that have a booking.

Let's show a bit more detail in this report:

```
SORT TEX.QCH WITH CTRY EQ "AU""GB""JP""DE""US" AND WITH YEAR EQ "2012" BY CTRY BY HS BREAK.ON CTRY
BREAK.ON HS TOTAL FOB DET.SUP
Ctry   HS Code   ......FOB Value
            01       78,985,296
            02       65,252,009
            03      230,202,508
etc
            98      104,033,980
            99                0
AU               ---------------
                  9,159,794,657
            01                0
            02      343,692,327
etc
```

The report now shows exports for each country broken into major categories according to the Harmonised System (HS) for export classification. The above listing shows that country 'AU' (Australia) took $9,159m of exports during 2012, of which $230m was fish (HS code 03).

Page breaks

In the above listing, you may want greater separation for each country. We can do this by inserting a page break at each country break by using the `'P'` option in the break command:

```
SORT TEX.QCH WITH CTRY EQ "AU""GB""JP""DE""US" AND WITH YEAR EQ "2012" BY CTRY BY HS BREAK.ON CTRY
"'P'" BREAK.ON HS TOTAL FOB ID.SUP DET.SUP
Ctry   HS Code   ......FOB Value
            01       78,985,296
            02       65,252,009
            03      230,202,508
etc
            98      104,033,980
            99                0
AU               ---------------
                  9,159,794,657
```

The next page would show the data for Germany (country 'DE'), while the one after that will show the UK data (country 'GB'), and so on.

One problem with these listings is that the country information is not being displayed until the break line. That is several screens of information before you find which country you are displaying.

One solution to this is to include the country identifier with the HS chapter identifier to create a compound attribute. We'll use an `I-type` dictionary named `CTRY.HS` to do this with an expression of:

```
      CTRY:’ ‘:HS
```

This concatenates the CTRY and HS dictionaries, with a space character between them. The format code becomes ‘5L’ to accommodate the width of the combined item. Now:

```
SORT TEX.QCH WITH CTRY EQ "AU""GB""JP""DE""US" AND WITH YEAR EQ "2012" BY CTRY BY HS BREAK.ON CTRY
"'P'" BREAK.ON CTRY.HS TOTAL FOB DET.SUP
Ctry   Ctry HS   ......FOB Value
       AU 01          78,985,296
       AU 02          65,252,009
       AU 03         230,202,508
etc
       AU 98         104,033,980
       AU 99                   0
AU               ---------------
                   9,159,794,657
```

Now, we may decide that we don’t want the main country column displaying, but we still want to break on that value. We use the BREAK.SUP keyword for this:

```
SORT TEX.QCH WITH CTRY EQ "AU""GB""JP""DE""US" AND WITH YEAR EQ "2012" BY CTRY BY HS BREAK.SUP
CTRY "'P'" BREAK.ON CTRY.HS TOTAL FOB DET.SUP
Ctry HS   ......FOB Value
AU 01          78,985,296
AU 02          65,252,009
AU 03         230,202,508
etc
AU 98         104,033,980
AU 99                   0
          ---------------
            9,159,794,657
```

Grand totals

OpenQM automatically creates a grand total line for your query when you use the TOTAL, AVG, ENUM, MAX, MIN, or PCT keywords. This consists of a row of double underlines followed by the summary value.

```
SORT TEX.QCH WITH YEAR EQ "2009""2010""2011""2012" BREAK.ON YEAR "'V'" TOTAL FOB ID.SUP DET.SUP
Year   ......FOB Value
2009    37,776,649,244
2010    41,769,862,081
2011    45,905,449,806
2012    44,356,210,157

       169,808,171,288

387296 record(s) listed
```

You can change the format of this grand total line by using the GRAND.TOTAL keyword. Options include providing specific text for the grand total line, suppressing the grand total line completely, or printing it on a separate page.

```
SORT TEX.QCH WITH YEAR EQ "2012" AND WITH HS EQ "02" AND WITH CTRY EQ "AU""CN""DE""GB""JP""US" BY
CTRY BREAK.ON CTRY TOTAL FOB GRAND.TOTAL "Meat" DET.SUP
Ctry   ......FOB Value
AU          65,252,009
CN         411,719,261
DE         343,692,327
GB         589,510,001
JP         280,448,620
US       1,180,039,662

Meat     2,870,661,880

24 record(s) listed
```

In the above command, the text “Meat” is inserted into the grand total line by the GRAND.TOTAL keyword. To suppress the grand total, use the ‘L’ option:

```
SORT TEX.QCH WITH YEAR EQ "2012" AND WITH HS EQ "02" AND WITH CTRY EQ "AU""CN""DE""GB""JP""US" BY
CTRY BREAK.ON CTRY TOTAL FOB GRAND.TOTAL "'L'" DET.SUP
Ctry   ......FOB Value
AU          65,252,009
CN         411,719,261
DE         343,692,327
GB         589,510,001
JP         280,448,620
US       1,180,039,662

24 record(s) listed
```

Alternatively, you can use the NO.GRAND.TOTAL modifier:

```
SORT TEX.QCH WITH YEAR EQ "2012" AND WITH HS EQ "02" AND WITH CTRY EQ "AU""CN""DE""GB""JP""US" BY
CTRY BREAK.ON CTRY TOTAL FOB NO.GRAND.TOTAL DET.SUP
```

5.5.4 Scaling data

All our queries have so far reported FOB exports down to the last dollar. Usually, we don’t want or need that level of detail. Typically, exports will be expressed in millions of dollars (and in larger countries, perhaps billions of dollars). So, how do we do that?

We’ve already seen that conversions can be used to change the way that data is displayed. And that is exactly what we want to do here. We want to change the way that the data is displayed without actually changing it internally.

Our FOB dictionary item currently uses a conversion of MR,. This could also be written as MR0, or MR00,. Essentially, this means to display the data with zero decimal places, and no scaling. What we want is to display the data in millions of dollars, and to display this to one decimal place.

We can do this with a conversion code of MR16,. This means that we want to descale[20] the value by six decimal places, and display to one decimal place. We can put this in a CONV modifier to test it:

```
SORT TEX.QCH WITH HS EQ "02" AND WITH CTRY EQ "AU""CN""DE""GB""JP""US" AND WITH YEAR EQ "2012" BY
CTRY BREAK.ON CTRY TOTAL FOB CONV "MR16Z," COL.HDG "'RX'FOB $m" DET.SUP
Ctry            FOB $m
AU                65.3
CN               411.7
DE               343.7
GB               589.5
JP               280.4
US             1,180.0

               2,870.7

24 record(s) listed
```

This has also used a COL.HDG modifier to show that the values are now being expressed in millions of dollars.

If you compare the above listing with those found on previous pages, you will find that both the individual values and the grand total have been correctly expressed in millions rounded to one decimal place.

As this is a common output format, we should put this into a dictionary item. Let’s call it FOB.M:

20 See Section 6.4.2 for more on descaling in conversion codes.

```
       I
       FOB
       MR16,
       'R'FOB $m
       9R
       S
```

```
SORT TEX.QCH WITH HS EQ "02" AND WITH CTRY EQ "AU""CN""DE""GB""JP""US" AND WITH YEAR EQ "2012" BY
CTRY BREAK.ON CTRY TOTAL FOB.M DET.SUP
Ctry   ...FOB $m
AU          65.3
CN         411.7
DE         343.7
GB         589.5
JP         280.4
US       1,180.0

         2,870.7

24 record(s) listed
```

Note that this dictionary item has also been given an output width that is more appropriate to the size of data that is appearing.

5.5.5 Page headings and footings

From the *QMQuery* elements introduced so far, you should be able to construct a basic report that selects, sorts, groups, and summarises data. However, to make this look like a proper report, we need to add page headings and footings. The keywords to do this in *OpenQM* are HEADING and FOOTING, and have the following format:

```
HEADING “text”

FOOTING “text”
```

where “text” contains the text to be displayed in the heading or footing, along with a number of optional formatting codes. These formatting codes are enclosed in single quotes within the text string.

Some of the formatting codes are listed below:

Code	**Purpose**
B	Inserts the value of the data from the corresponding B code in a BREAK.ON option string
C	Centres the heading text
D	Inserts the date
F	Inserts the file name
G	Inserts a gap to utilise the full width of the output device
L	Inserts a new line
P	Inserts the page number

A full list of formatting codes can be found in the *OpenQM* documentation and on-line help.

For example:

```
TERM 65

SORT XRATES WITH YEAR EQ "2012" BREAK.ON MONTH AVG USD NO.NULLS AVG AUD NO.NULLS AVG GBP NO.NULLS
DET.SUP HEADING "'DGC'Average Monthly Exchange Rates'G'Page 'PLC'Year: 2012'L'"

TERM 132
```

produces:

```
24 JUN 2013      Average Monthly Exchange Rates        Page    1
                           Year: 2012

Month.....   US Dollar   Aus Dollar   UK Pound
January         0.8007       0.7691     0.5162
February        0.8343       0.7780     0.5281
March           0.8208       0.7779     0.5186
April           0.8190       0.7908     0.5117
May             0.7762       0.7766     0.4869
June            0.7801       0.7813     0.5014
July            0.7982       0.7757     0.5118
August          0.8098       0.7733     0.5156
September       0.8174       0.7867     0.5077
October         0.8198       0.7967     0.5099
November        0.8192       0.7875     0.5130
December        0.8318       0.7943     0.5155

                0.8102       0.7821     0.5112

251 record(s) listed
```

The first TERM statement adjusts the width of the output device (screen) to something appropriate for the centring of the text. The second TERM statement returns the screen width to its previous value.

The HEADING statement specifies a 3 line heading. However, there is nothing in the third line, so this simply spaces the heading from the body of the report.

The first line has 3 components – the date (specified by the 'D' option), a main heading, and a page number (specified by the 'P' option). These are separated by 'G' options which force the heading to occupy the total width of the output device (the screen), while the second element is centred by the 'C' option.

The second line only has one element which is centred by the 'C' option.

Let's extend the above statement to place the year in the heading automatically from a BREAK.ON clause:

```
TERM 65

SORT XRATES WITH YEAR GE "2010" BREAK.ON YEAR "'B'" BREAK.ON MONTH AVG USD NO.NULLS AVG AUD
NO.NULLS AVG GBP NO.NULLS DET.SUP HEADING "'DGC'Average Monthly Exchange Rates'G'Page 'PLC'Year:
'BL'"

TERM 132
```

This produces:

```
24 JUN 2013      Average Monthly Exchange Rates        Page    1
                           Year: 2010

Month.....   US Dollar   Aus Dollar   UK Pound
January         0.7277       0.7959     0.4501
February        0.6974       0.7869     0.4455
March           0.7033       0.7713     0.4670
April           0.7124       0.7685     0.4644
May             0.6992       0.8019     0.4761
June            0.6928       0.8105     0.4696
July            0.7111       0.8134     0.4658
August          0.7154       0.7945     0.4566
September       0.7259       0.7766     0.4667
October         0.7501       0.7647     0.4734
November        0.7727       0.7805     0.4837
December        0.7504       0.7573     0.4808
             ---------   ----------   --------
                0.7215       0.7852     0.4669
```

The key change in this statement is the inclusion of a 'B' option in the BREAK.SUP clause. This is matched by another 'B' option in the HEADING clause. This puts the break value into the heading. Therefore, the heading correctly identifies the year whose months are displayed in the report.

There is another case of an implied option being used here. We didn't actually specify that there should be a page break on a year change, but this option is implied by putting the break value into the heading. Once again, other multi-value databases will probably require you to specify the `'P'` option in the `BREAK.ON` clause.

Footings are included in a similar fashion. For example the following footing could be added to the above statement:

```
FOOTING "Source: Rush Flat Consulting'L'       Based on RBNZ data"
```

This would be output as:

```
Source: Rush Flat Consulting
        Based on RBNZ data
```

Note that the footing appears at the bottom of the output device (screen or printed page) rather than immediately following the body of the report.

5.6 Printing and Report Styles

5.6.1 Printing

So far, we have found how to create a report and display it on the screen. That is good, but we often want a printed copy of the report too. So, how do we send the report to the printer?

Well, the answer is simple – there is another keyword to send the report to the printer – but there are a number of complexities associated with printing.

The LPTR keyword

To send a report to the printer, all we need to do is add the `LPTR` keyword to the *QMQuery* statement. The full format of the keyword is:

```
      LPTR {unit}
```

where unit is the print unit number. If unit is omitted, then print unit 0 is used.

The exact meaning of print units will be covered shortly. But first, let's look at a *QMQuery* statement that sends output to the printer:

```
SORT XRATES WITH YEAR GE "2010" BREAK.ON YEAR "'B'" BREAK.ON MONTH AVG USD NO.NULLS AVG AUD
NO.NULLS AVG GBP NO.NULLS DET.SUP HEADING "'DGC'Average Monthly Exchange Rates'G'Page 'PLC'Year:
'BL'" LPTR
```

If you execute this statement, perhaps something will come out of the printer, or perhaps not. It depends on how your print units are set, and what sort of printer you have.

Print units

OpenQM uses logical print units to provide flexibility in report output. A logical print unit may direct output to a printer, to a file, or both. Individual print units may be defined to allow printing to different printers, and in different formats (portrait, landscape, different fonts, different page sizes).

Up to 256 print units – numbered 0 to 255 – can be defined for any session. It is unlikely that you will need to define anywhere near this number of print units.

To take advantage of all of the printing features of *OpenQM*, a PCL printer is required. If you have a GDI printer, features such as control of page orientation and print pitch are not available – but are still available to you if you define a print unit as PCL and print through a PCL to Windows print driver (such as Anzio Print Wizard, or PageTech PCLReader). Given the availability of this software (with PCLReader being free), further discussion here will assume use of the PCL settings.

To see the print units currently defined, type:

```
SETPTR DISPLAY
```

If you haven't defined any print units, you will see the following:

```
SETPTR DISPLAY
Unit Width Depth Tmgn Bmgn Mode Options
   0    80    66    0    0    1
```

Print unit 0 is the default print unit. Failing any other definition, it is defined as a page 80 characters wide by 66 lines deep, with top and bottom margins of 0 lines. Mode 1 means that output will be sent to a printer, and with no printer being specifically defined here, it will go to the default printer. No options are specified.

Let's change this definition to something a bit more useful. Type in:

```
SETPTR 0,90,66,0,2,3, AS NEXT QMPRINT, PCL, CPI 12, LEFT.MARGIN 2
```

OpenQM will respond:

```
SETPTR 0,90,65,0,2,3, AS NEXT QMPRINT, PCL, CPI 12, LEFT.MARGIN 2
PRINT UNIT 0
   Page width     : 90
   Lines per page : 65
   Top margin     : 0
   Bottom margin  : 2
   Mode           : 3 (Hold file: $HOLD QMPRINT)
                      Using next suffix number
   LEFT.MARGIN 2
   PCL: CPI = 12, LPI = 6, Weight = MEDIUM, Symbol set = ROMAN-8, Paper size = A4

OK to set new parameters (Y/N)?
```

Before you answer 'Y', consider what paper size you use. The A4 papersize being set here is fine for European countries, but North America typically uses Letter sized paper. In that case, the command should be:

```
SETPTR 0,90,61,0,2,3, AS NEXT QMPRINT, PCL, CPI 12, LEFT.MARGIN 2, PAPER.SIZE LETTER
```

This defines the output page as 90 characters wide by 61 lines deep (letter size paper is shorter than A4 paper), with a top margin of 0, and a bottom margin of 2. The output mode is 3 – meaning that the report will not be printed, but will be written to the `$HOLD` file in the account that you are working in. It will be saved there with a name of `QMPRINT`_nnnn where nnnn is a number that is incremented with each print job. We have said that the printer is PCL capable, and that we want to print in a 12 pitch font, and have a left margin on the page of 2 characters. The paper size is A4 unless we specify a different paper size in the `SETPTR` command.

Now, if we execute our *QMQuery* statement:

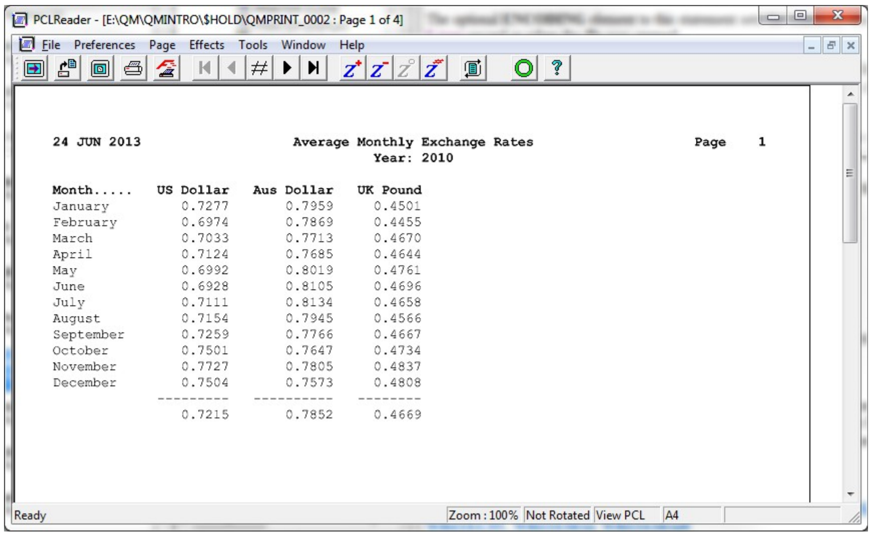
```
SORT XRATES WITH YEAR GE "2010" BREAK.SUP YEAR "'B'" BREAK.ON MONTH AVG USD NO.NULLS AVG AUD
NO.NULLS AVG GBP NO.NULLS DET.SUP HEADING "'DGC'Average Monthly Exchange Rates'G'Page 'PLC'Year:
'BL'" FOOTING "Source: Rush Flat Consulting'L'        Based on RBNZ data" LPTR
```

… nothing appears to happen. The cursor simply returns to the command prompt. However, if you look in the $HOLD file of the QMINTRO account, then you will see a print file in there named QMPRINT_0001 (or some other number).

```
SORT $HOLD
$HOLD.....
QMPRINT_00
01

1 record(s) listed
```

If you open this file with a text editor[21], you will find it contains the report output, plus some HP PCL formatting codes.

If you have a PCL viewer, then you can view the contents of the report by opening this file with the viewer, and then print the report to any Windows printer. This is shown in the screenshot below.


PCLReader - [E:\QM\QMINTRO\$HOLD\QMPRINT_0002 : Page 1 of 4]
File Preferences Page Effects Tools Window Help
Ready
Zoom : 100%
Not Rotated
View PCL
A4

```
24 JUN 2013                  Average Monthly Exchange Rates                 Page    1
                                       Year: 2010

Month.....   US Dollar   Aus Dollar   UK Pound
January         0.7277       0.7959     0.4501
February        0.6974       0.7869     0.4455
March           0.7033       0.7713     0.4670
April           0.7124       0.7685     0.4644
May             0.6992       0.8019     0.4761
June            0.6928       0.8105     0.4696
July            0.7111       0.8134     0.4658
August          0.7154       0.7945     0.4566
September       0.7259       0.7766     0.4667
October         0.7501       0.7647     0.4734
November        0.7727       0.7805     0.4837
December        0.7504       0.7573     0.4808
             ---------   ----------   --------
                0.7215       0.7852     0.4669
```

Now, what happens if we want to print a report in landscape mode? We create another print unit for landscape printing:

```
SETPTR 1,132,41,0,2,3, AS NEXT QMPRINT, PCL, CPI 12, LANDSCAPE, LEFT.MARGIN 2
```

Make sure you add a PAPER.SIZE setting and adjust the line count if you are not using A4.

The differences between this and the previous definition are:

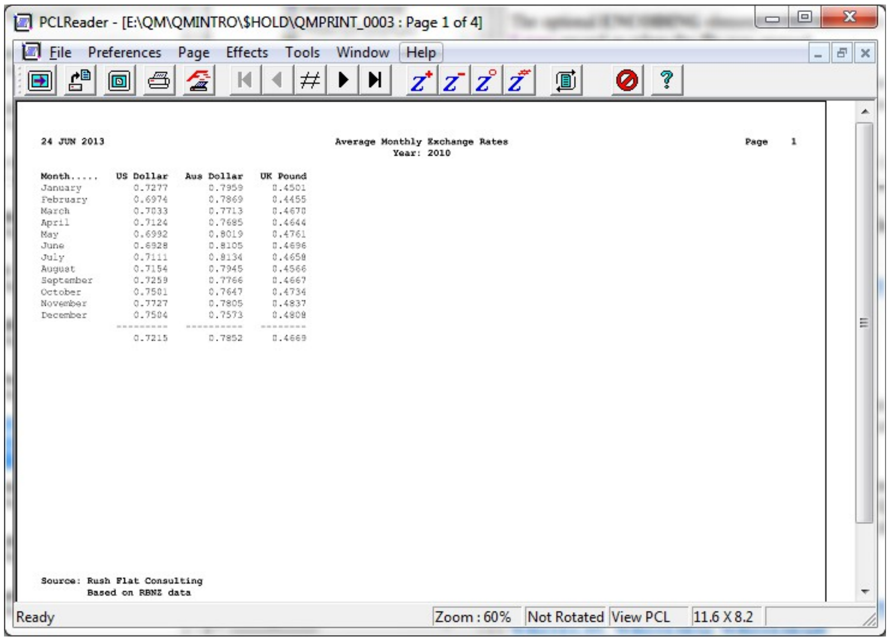
- this defines print unit 1 (rather than 0)
- the page size is 132 characters wide by 41 lines deep
- the LANDSCAPE option is specified

To use this print unit, we need to specify the print unit number in the *QMQuery* command:

```
SORT XRATES WITH YEAR GE "2010" BREAK.SUP YEAR "'B'" BREAK.ON MONTH AVG USD NO.NULLS AVG AUD
NO.NULLS AVG GBP NO.NULLS DET.SUP HEADING "'DGC'Average Monthly Exchange Rates'G'Page 'PLC'Year:
'BL'" FOOTING "Source: Rush Flat Consulting'L'        Based on RBNZ data" LPTR 1
```

and the output looks like:

21 You could view the contents using WED by typing: WED $HOLD QMPRINT_0001


PCLReader - [E:\QM\QMINTRO\$HOLD\QMPRINT_0003 : Page 1 of 4]
File Preferences Page Effects Tools Window Help
Ready
Zoom : 60%
Not Rotated
View PCL
11.6 X 8.2

```
24 JUN 2013                        Average Monthly Exchange Rates              Page    1
                                             Year: 2010

Month.....   US Dollar   Aus Dollar   UK Pound
January         0.7277       0.7959     0.4501
February        0.6974       0.7869     0.4455
March           0.7033       0.7713     0.4670
April           0.7124       0.7685     0.4644
May             0.6992       0.8019     0.4761
June            0.6928       0.8105     0.4696
July            0.7111       0.8134     0.4658
August          0.7154       0.7945     0.4566
September       0.7259       0.7766     0.4667
October         0.7501       0.7647     0.4734
November        0.7727       0.7805     0.4837
December        0.7504       0.7573     0.4808
             ---------   ----------   --------
                0.7215       0.7852     0.4669

Source: Rush Flat Consulting
        Based on RBNZ data
```

Note that we haven't fully utilised the available page areas. The portrait mode report has plenty of vertical space available, while the landscape mode report has plenty of horizontal space available.

Initialising print units

Of course, you don't want to have to define a print unit every time you want to print a report. Therefore, we need a way to automatically define the print units so they are ready for your use (print units only exist for your current session – if you close your session, and then start a new session, the print units you defined above will be gone).

The best way to do this is to place your print units in the LOGIN item of the account:

```
WED VOC LOGIN
```

Add the following two lines to the LOGIN entry:

```
SETPTR 0,90,65,0,2,3, AS NEXT QMPRINT, PCL, CPI 12, LEFT.MARGIN 2, BRIEF
SETPTR 1,132,41,0,2,3, AS NEXT QMPRINT, PCL, CPI 12, LANDSCAPE, LEFT.MARGIN 2, BRIEF
```

These are the same SETPTR commands we used above, with the addition of the BRIEF option. This suppresses the 'OK to set new parameters' prompt when defining a print unit.

Next time you log into this account, these two print units should be automatically defined for you. To check this, log out, then log back in and type:

```
SETPTR DISPLAY
```

This should report the two print units and their settings.

Defining your own print units

The above two print units will get you started in defining your own print units to suit your own purposes. Start with what printer you have attached to your system, and what paper size it uses. If it is a PCL printer, or you have PCL printing software, then define the print unit as PCL as this gives you more options.

Now, you'll need to start playing with settings to see what suits you. How may characters wide do you need your printouts? This will define whether you use portrait or landscape mode, and what printing pitch is necessary to achieve this. Note: You need PCL printing to do this.

Start with your paper size. Can your printer print in landscape mode? What print pitches can it print at? Basic HP print pitches are 10, 12, and 16.67 characters per inch, while standard lines per inch settings are 6 and 8. Even these limited settings can give you a reasonable range of print units.

Now put your fully defined print unit into the account LOGIN item so that it is defined every time you log into the account.

Spooling print files

If your print units send their output to a file (mode 3), then you need a way to print these files. Using a PCL viewer to do this has already been outlined above, but you can send them direct to the printer without using a PCL viewer. However, to do this, you need a physical print unit defined – i.e. one that points to your printer.

You could duplicate the print units you already have:

```
SETPTR 10,90,65,0,2,1, PCL, CPI 12, LEFT.MARGIN 2, BRIEF
SETPTR 11,132,41,0,2,1, PCL, CPI 12, LANDSCAPE, LEFT.MARGIN 2, BRIEF
```

This defines print units 10 and 11 (10 more than the equivalent units that go to the $HOLD file). These don't have the 'AS NEXT ...' option as this isn't relevant to a physical printer.

To print reports from the $HOLD file to the print unit defining the physical printer, we use the SPOOL command:

```
        SPOOL file record(s) LPTR n
```

So, to print our landscape report onto the landscape physical printer:

```
SPOOL $HOLD QMPRINT_0003 LPTR 11
```

Of course, this assumes that your printer is capable of receiving the data in the file. So if you are creating PCL files but you don't have a PCL printer, then you will need to use a PCL viewer.

This command allows you to specify the file name. While the default output file is $HOLD, you can specify an alternate file in the SETPTR command (using the AS PATHNAME option) or create print files in an alternate file direct from *QMBasic*.

Deleting print files

If your print units are set to send their output to a file – then over time you will build up a number of items within that file. The $HOLD file is a directory file, so you can simply delete them using a file manager. Alternatively, you could delete them using the DELETE or CLEAN.ACCOUNT verbs:

```
        DELETE $HOLD item-id

        DELETE $HOLD ALL

        DELETE filename item-id

        DELETE filename ALL

        CLEAN.ACCOUNT
```

CLEAN.ACCOUNT clears not only the $HOLD file, but the $SAVEDLISTS and $COMO (if it exists) files as well. The purpose of the $SAVEDLISTS file is covered in Section 6.2. See the documentation for information on the $COMO file.

5.6.2 Report styles

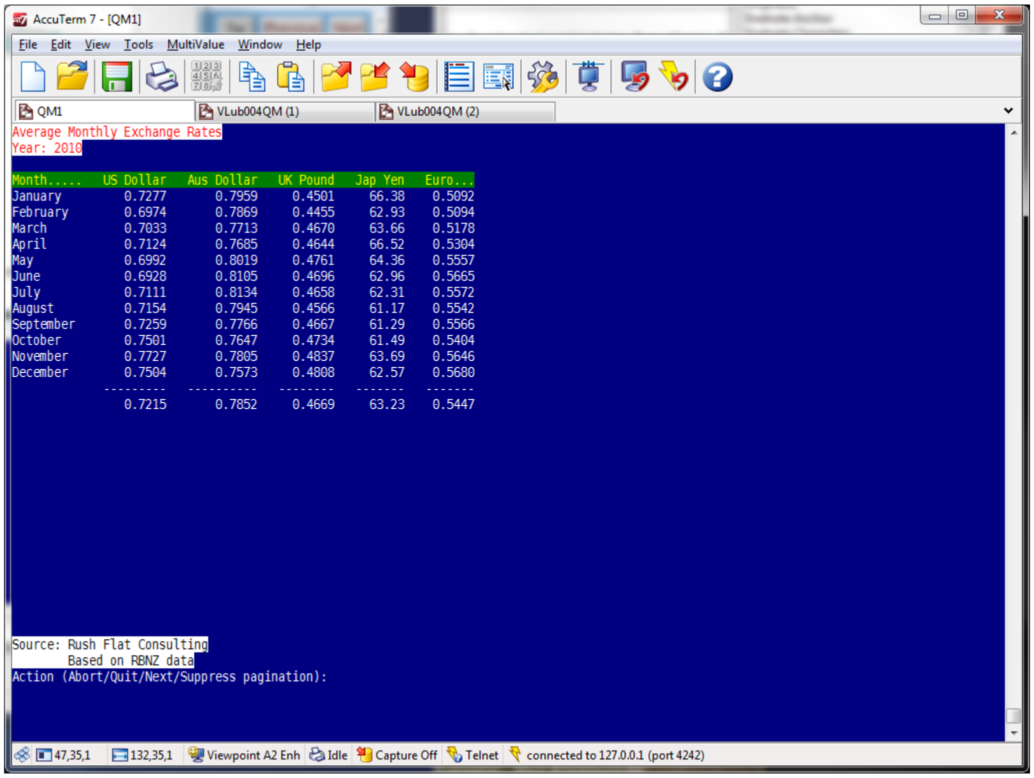
Report styles set the font colour for on-screen reports, and the font-weight for printed reports. Enter in the following report style definition:

```
WED VOC DEFAULT.STYLE
X
Heading=BRIGHT RED,BRIGHT WHITE,BOLD
ColumnHeading=YELLOW,GREEN,BOLD
Detail=BRIGHT WHITE,BLUE
SubTotal=BRIGHT WHITE,RED,BOLD
Total=BRIGHT WHITE,RED,BOLD
Footing=BLACK,BRIGHT WHITE
Other=BRIGHT WHITE,BLUE
Exit=YELLOW,BLUE
```

Save the item, then type:

```
REPORT.STYLE DEFAULT.STYLE
```

Nothing appears to happen when you do this, but now try displaying a *QMQuery* report to the screen:



```
Average Monthly Exchange Rates
Year: 2010

Month.....  US Dollar  Aus Dollar  UK Pound  Jap Yen  Euro...
January        0.7277      0.7959    0.4501    66.38   0.5092
February       0.6974      0.7869    0.4455    62.93   0.5094
March          0.7033      0.7713    0.4670    63.66   0.5178
April          0.7124      0.7685    0.4644    66.52   0.5304
May            0.6992      0.8019    0.4761    64.36   0.5557
June           0.6928      0.8105    0.4696    62.96   0.5665
July           0.7111      0.8134    0.4658    62.31   0.5572
August         0.7154      0.7945    0.4566    61.17   0.5542
September      0.7259      0.7766    0.4667    61.29   0.5566
October        0.7501      0.7647    0.4734    61.49   0.5404
November       0.7727      0.7805    0.4837    63.69   0.5646
December       0.7504      0.7573    0.4808    62.57   0.5680
            ---------  ----------  --------  -------  -------
               0.7215      0.7852    0.4669    63.23   0.5447

Source: Rush Flat Consulting
        Based on RBNZ data
Action (Abort/Quit/Next/Suppress pagination):
```

Your report is now in colour. Now try sending the report to the printer. It should now have bolded headings, column headings, and total lines.

Of course, you can define multiple style records to provide a different look to reports. And *OpenQM* provides multiple ways to switch between styles:

- include the style name in the SETPTR command
- include the STYLE keyword and style name in the *QMQuery* command
- issue the REPORT.STYLE command before running the *QMQuery* command

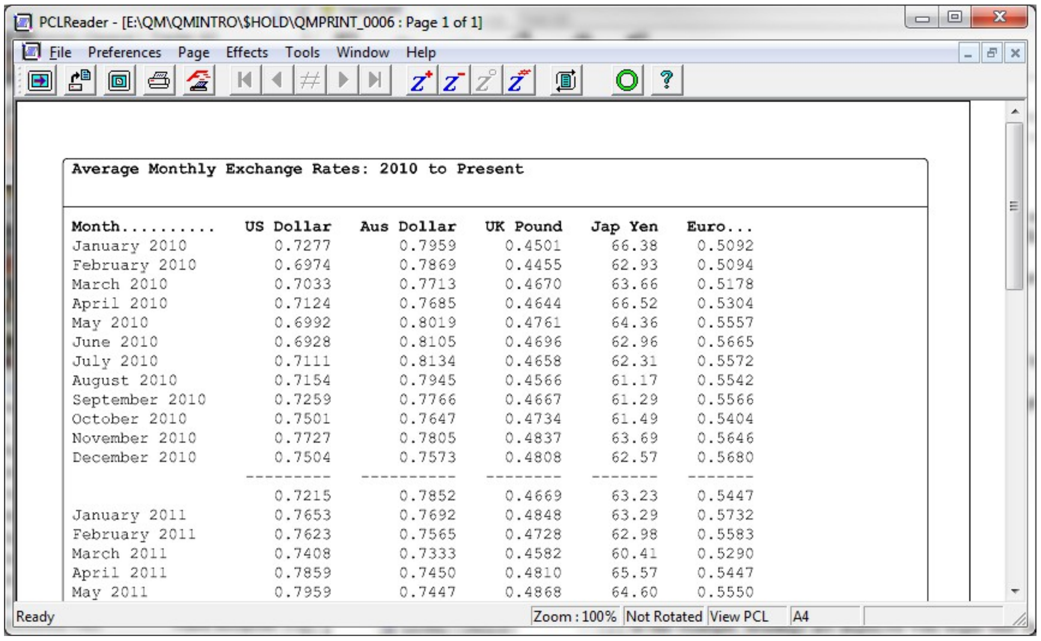
Once you have found a set of colours and font weights that you like, and you want to make this the default setting for *OpenQM*, place the REPORT.STYLE command in the LOGIN item for the account.

If you have multiple accounts for which you wish to apply a single standard style, you could create the 'real' style record in one master account, and then use a remote item in each of the other accounts to point to the real style item. This means that when you change the style item in the master account, you will change the style for all accounts that reference that style.

5.6.3 Other printing and display options

Boxed reports

Boxed reports are another feature of the PCL printing mode in *OpenQM*. Essentially, use of the BOXED keyword in the *QMQuery* statement will cause *OpenQM* to draw a box around the page borders, with the *QMQuery* report inside the box. Headers and footers also occupy their own ruled off areas in the boxed report. This is shown below:


PCLReader - [E:\QM\QMINTRO\$HOLD\QMPRINT_0006 : Page 1 of 1]
File Preferences Page Effects Tools Window Help
Ready
Zoom : 100%
Not Rotated
View PCL
A4

```
Average Monthly Exchange Rates: 2010 to Present

Month.........  US Dollar  Aus Dollar  UK Pound  Jap Yen  Euro...
January 2010       0.7277      0.7959    0.4501    66.38   0.5092
February 2010      0.6974      0.7869    0.4455    62.93   0.5094
March 2010         0.7033      0.7713    0.4670    63.66   0.5178
April 2010         0.7124      0.7685    0.4644    66.52   0.5304
May 2010           0.6992      0.8019    0.4761    64.36   0.5557
June 2010          0.6928      0.8105    0.4696    62.96   0.5665
July 2010          0.7111      0.8134    0.4658    62.31   0.5572
August 2010        0.7154      0.7945    0.4566    61.17   0.5542
September 2010     0.7259      0.7766    0.4667    61.29   0.5566
October 2010       0.7501      0.7647    0.4734    61.49   0.5404
November 2010      0.7727      0.7805    0.4837    63.69   0.5646
December 2010      0.7504      0.7573    0.4808    62.57   0.5680
                ---------  ----------  --------  -------  -------
                   0.7215      0.7852    0.4669    63.23   0.5447
January 2011       0.7653      0.7692    0.4848    63.29   0.5732
February 2011      0.7623      0.7565    0.4728    62.98   0.5583
March 2011         0.7408      0.7333    0.4582    60.41   0.5290
April 2011         0.7859      0.7450    0.4810    65.57   0.5447
May 2011           0.7959      0.7447    0.4868    64.60   0.5550
```

The presence of the box improves the appearance of the report. Note that the lines do take up some printing space, so the effective page width is a little less when then the BOXED keyword is used.

Panning and scrolling

The PAN and SCROLL keywords are available for use with on-screen reports.

- PAN lets you create a report that is wider than the screen width, and then "move" the screen over the report so that you can see all columns.

- SCROLL lets you move back over pages that have already been displayed on the screen.

PAN operates differently depending on where the keyword appears in the statement. If it appears before or after all of the output fields, then the whole screen display is panned. However, if it appears between the output fields, only those columns appearing after the PAN keyword are scrolled, while those columns that appear before the PAN keyword remain fixed on the screen.

5.7 Miscellaneous Aspects of QMQuery

5.7.1 Default display and phrases

Type in: **SORT VOC**

Similarly: **SORT DICT XRATES**

Notice that both of these commands produce a report with one or more output fields in the display – when we didn't specify a display clause in the statement.

OpenQM can use a default display clause for those occasions when the user does not explicitly specify a display clause. This default display clause is contained in the dictionary of the file with an item-name of @.

So, the default item for VOC looks like:

```
CT DICT VOC @
DICT VOC @
1: PH
2: DESC FMT '60T'
```

And for the dictionary listing:

```
CT DICT.DICT @
DICT.DICT @
1: PH
2: TYPE FLD CONVERSION NAME FORMAT.CODE SMV ASSOCIATION_
3: BY TYPE.CODE BY LOC.R BY NAME
```

Before we look at the contents of these items, consider the files where these items were found.

For the VOC, the @ item was found in DICT VOC. That isn't too surprising, but if you look at the QMINTRO account through a file manager, you won't find a file named VOC.DIC. However, if you issue a LISTF command, you will see that the path to the VOC dictionary is @QMSYS\VOC.DIC (where @QMSYS is a token referring to the O/S level path to the QMSYS account). This means that all accounts on your *OpenQM* system will share the same VOC dictionary (but they each have their own individual VOC data file).

Now, what about the default dictionary listing. Do dictionaries have a dictionary to define words and phrases? The answer is 'Yes', and it is named DICT.DICT. Once again, this is a shared dictionary for all accounts and has a real pathname of @QMSYS\DICT.DIC. Note the two different spellings – the real filename is DICT.DIC but is referenced in all accounts as DICT.DICT. Why the difference? Internally, all dictionaries are referred to as DICT but have a real filename of *filename*.DIC, so it is simply consistent with the rest of the file naming conventions.

Let's look at what was contained in those '@' items.

Both items had 'PH' in the first line. This identifies the item to the *QMQuery* processor as a PHRASE. Put simply, a phrase is a shortcut expression for a larger set of words. When the *QMQuery* processor encounters a phrase, it substitutes the words contained on line 2 of the phrase definition for the phrase name in the *QMQuery* statement.

Therefore, line 2 contains the default display clause for those files (the VOC and any dictionary).

The default display clause for the VOC contains only a single output field (DESC) and a display format expression (FMT '60T').

In contrast, the default display clause for dictionaries contains seven output items and a three-level sort clause. This indicates that phrases can contain more than just the display clause of a *QMQuery* statement.

Note also that those elements are spread across lines two and three in the phrase, when the phrase is supposed to only exist on line two. However, you can spread phrases across multiple lines, as long you terminate any lines prior to the last line with a continuation character (an underscore).

Let's try creating a default display item in the XRATES dictionary:

```
PH
WITH YEAR GE "2010"_
BREAK.ON YEAR BREAK.ON MONTH_
AVG USD NO.NULLS AVG AUD NO.NULLS AVG GBP NO.NULLS AVG EUR NO.NULLS AVG JPY NO.NULLS_
HEADING "New Zealand Exchange Rates Since 2010'L'" DET.SUP
```

Note that we have used extension characters to spread the phrase across multiple lines.

Now try a *QMQuery* statement without no display clause:

```
SORT XRATES
```

This now generates a basic report, complete with a heading and with the ID suppressed. This makes simple reporting somewhat easier.

Now try these statements:

```
SORT XRATES HEADING “Example heading”

SORT XRATES BY.DSND @ID

SORT XRATES WITH YEAR GE “2012”
```

The first two of these work OK, but the third doesn't.

In the first case, the HEADING specified in the second statement was used in preference to the default heading, while the second example changed the sort criteria. However, the selection criteria in the third example did not override that in the default display phrase.

Therefore, when you set up phrases, make sure that they work properly with all the statements you are likely to use.

We can send these default reports to the print units simply by adding LPTR to the statement. However, *OpenQM* has another twist for default printing.

The default display clause for printing is actually contained in the item '@LPTR'. This is also contained in the dictionary of the file being used in the *QMQuery* statement. However, if '@LPTR' does not exist, then the default display specified in '@' will be used.

This lets you define one default display for on-screen reporting, and a different (perhaps wider) default display for printing.

5.7.2 Saving QMQuery statements for later use

You now have your completed *QMQuery* statement. But it is several lines long, and you don't want to type it in every time that you want to run that particular report. What we need now is some way to save the statement, so that you can run it again later.

OpenQM provides several ways to do this.

The traditional method amongst Information style databases is to create a 'sentence' or a 'paragraph' in the VOC, and then run the sentence or paragraph by typing in its name.

PICK style databases let you store the query statements as items in a file. You can then run the commands contained in the item by typing: *command filename itemname* where the command varies with the particular database flavour.

OpenQM supports both methods of saving and running statements, and adds another variant with its Private VOC[22].

A sentence contains an 'S' in the first line, and the *QMQuery* command in the second line. Therefore, a sentence named 'MY.QUERY' might look like:

```
CT VOC MY.QUERY
VOC MY.QUERY
1: S
2: SORT XRATES WITH YEAR GE "2010" BREAK.SUP YEAR "'B'" BREAK.ON MONTH AVG USD NO.NULLS AVG AUD
NO.NULLS AVG GBP NO.NULLS DET.SUP HEADING "'DGC'Average Monthly Exchange Rates'G'Page 'PLC'Year:
'BL'" FOOTING "Source: Rush Flat Consulting'L'        Based on RBNZ data" LPTR
```

If you already have this command sitting on your command stack, you can get *OpenQM* to create the sentence for you. Let's assume that the command you want to load into the sentence is in stack position 5. Type in:

```
.S MY.QUERY 5 5
```

Make sure you include the dot before the 'S'. This says save lines 5 to 5 as an item in the VOC named 'MY.QUERY'. Because there is only one line to save here, the second 5 could have been omitted.

If you specify more than one line to be saved, the *OpenQM* will save the commands as a paragraph (rather than a sentence).

Now, typing 'MY.QUERY' from the command prompt will display the report.

To see the paragraphs and sentences stored in the VOC, type:

LISTS	Lists sentences
LISTPA	Lists paragraphs

What about running the command if it is stored in a separate file? Well, let's create a file to hold the queries in first:

```
CREATE.FILE QUERIES DIRECTORY
```

Note that this statement defines the file as a directory file. This means that you can use any text editor to edit the items directly from the operating system – although we'll still access them like any other item from inside *OpenQM*.

We'll use *AccuTerm*'s cut and paste features to create the command as an item in the file. List the command stack by typing:

22 From version 2.6-9 onwards. i.e. not in the GPL version.

```
.L
```

Then click at the start of the section you want to copy and drag a box around the commands. Choose 'Edit | Copy' from the menu, or use the shortcut keys defined in the terminal settings (usually Ctrl-C or Ctrl-Insert). Now start the *AccuTerm* editor:

```
WED QUERIES MY.QUERY
```

… and paste the commands into the item. Tidy them up using the editor, make the first line of the item 'S' or 'PA' as appropriate, then file the item.

Now you can run the item by typing either of:

```
RUN QUERIES MY.QUERY
```

```
.X QUERIES MY.QUERY
```

Which storage method should you use for your queries? That is up to you.

If you only have a small number of saved queries, then it won't matter which method you use. However, if you have a large number of saved queries, then it is probably better to use a distinct file for this purpose. This will help keep the VOC reserved for its correct purpose – that of being a repository of keywords used by *OpenQM*.

From version 2.6-9 onwards, *OpenQM* has added a Private VOC. This is a place where users can store paragraphs and sentences that are private to themselves.

The reason for the implementation of a private VOC is similar to that suggested above for using a separate file for storing your queries. In a multi-user environment with many people using the same account, the main VOC became crowded with stored sentences and paragraphs. Therefore, users had difficulty in finding a name for a new paragraph, and had difficulty finding the paragraph they wanted to run at a specific time. Giving each user a private VOC relieves some of this pressure, and also means that other users can't look at your private queries.

The private VOC uses variants on the .D (delete), .L (list), .R (read), and .S (save) commands. These variants are .DP, .LP, .RP, and .SP.

The private VOC is created automatically (if it does not already exist) when you save a query to it using .SP:

```
.SP MY.QUERY 14
Created DATA part as PVOC\BRIAN
```

```
CT PVOC,BRIAN MY.QUERY
PVOC,BRIAN MY.QUERY
1: S
2: SORT FX.DAILY BREAK.ON YEAR TOTAL USD NO.GRAND.TOTAL DET.SUP
```

```
.RP MY.QUERY
1 lines loaded from VOC record 'MY.QUERY'
```

```
.DP MY.QUERY
Delete private VOC record 'MY.QUERY'? Y
```

Note the second command listed above references the private VOC as PVOC,BRIAN. This is the way to reference a multi-file:

```
dict-name,data-name
```

Note the comma separating the dictionary name from the name of the data file. This method of referencing multi-files is also used to query multi-files using *QMQuery*. For example:

```
SORT CUSTOMERS,ARCHIVE BY POSTCODE BY SURNAME FULLNAME ADDRESS
```

5.8 Summary

This chapter has provided an introduction to *QMQuery* and QM dictionaries. By now, you should know how to:

- use the MODIFY editor to create simple dictionary items
- select records
- sort records
- specify which fields to display
- format column headings
- break the display into logical groups
- generate summary information (totals, averages etc.)
- scale output values to sensible magnitudes
- create headings and footings on the reports
- insert page breaks and break information into the headings and footings
- send reports to the printer
- define and select from multiple print units
- define and use report styles
- put a box around a printed report
- pan and scroll reports sent to the screen
- define default display clauses
- save queries for later use and run your saved queries

That may seem a lot, but that is only the introduction to *QMQuery*. The next chapter looks at some of the more advanced features available in *QMQuery*.

6 Advanced QMQuery

6.1 Advanced Elements of QMQuery

The previous chapter gave a broad overview of what *QMQuery* does, and how to construct a *QMQuery* statement. However, there are a number of areas where more coverage is required. These include:

- advanced record selection techniques
- use of inline prompts
- working with multiple data files
- performing calculations with I-types
- using alternate key indices
- using *QMBasic* in I-types
- output to O/S level files
- reformatting data files
- working with multi-values

Once we have covered that, you will just need lots of practice to make sure you understand everything!

6.2 Advanced Record Selection

The previous chapter introduced two standard verbs for use in *QMQuery* – LIST and SORT. These verbs allowed records to be selected by using a WITH or WHEN clause.

This overall structure of using a single statement to both select and display records generally works well. However, if the selection criteria are complex, then this structure can be limiting.

6.2.1 Parentheses in selection expressions

Traditionally, PICK style multi-value databases did not support parentheses in selection expressions, and therefore had difficulties when selection criteria were ambiguous without parentheses . For example:

```
SORT XRATES WITH YEAR GE "2011" AND WITH USD LT "0.75" OR GT "0.84" DATE USD
```

This statement is ambiguous because it is not clear how the AND and the OR parts of the expression relate to each other. It would be much clearer if written:

```
SORT XRATES WITH YEAR GE "2011" AND WITH (USD LT "0.75" OR GT "0.84") DATE USD
```

This makes it clear that the YEAR criteria always needs to be met, and is not overridden by the OR clause on the value of the US dollar.

However, Information style multi-value databases – such as *OpenQM* – DO support parentheses, and the statement with the parentheses shown above will execute correctly.

6.2.2 The SELECT and SSELECT verbs

Prioritisation of multiple selection criteria in PICK databases is achieved by allowing the selection to take place over a series of statements. The initial statements simply carry out record selection (and optionally sorting the records), with only the final statement providing the display clause. *OpenQM* allows this approach too.

The key verbs to use in this type of selection are SELECT and SSELECT. These are the selection equivalents of LIST and SORT. This means that both statements can utilise a sort clause, but the SSELECT verb will always have a final sort by item-id.

These verbs are used as precursors to one of the other *QMQuery* verbs (not necessarily just LIST and SORT). For example:

```
      SELECT XRATES WITH YEAR GE "2012"
      SELECT XRATES WITH USD LT "0.75" OR GT "0.84"
      SORT XRATES DATE USD ID.SUP
```

As these selection commands are executed, *OpenQM* will respond as follows:

```
:SELECT XRATES WITH YEAR GE "2012"
368 record(s) selected to list 0
::SELECT XRATES WITH USD LT "0.75" OR GT "0.84"
45 record(s) selected to list 0
::
```

The first thing to note is that *OpenQM* tells you how many records have been selected, and that they have been selected to list 0.

The second thing to note is that the prompt has changed from a single colon (:) to a double colon (::). This is your visual indicator that a select-list is active.

We will return to the concept of lists in a moment. First, we will continue to execute the series of statements outlined above:

```
SORT XRATES DATE USD ID.SUP
Date........   US Dollar
 13 DEC 2012      0.8430
 14 DEC 2012      0.8431
 17 DEC 2012      0.8463
 18 DEC 2012      0.8447
 19 DEC 2012      0.8414
 11 JAN 2013      0.8447
```

Now, this statement does not contain a selection clause itself, but has used the selection criteria contained in the previous two statements. We can see that not all records have been selected (the first records displayed are late in 2012 when we know the data starts in 2010), and those displayed meet the selection criteria for the value of the US Dollar.

6.2.3 Select-lists

The above selection is an example of using a select-list. A select-list is simply an internal list of item-ids created by one or more *QMQuery* statements for use with subsequent *QMQuery* statements.

The concept of using select-lists goes right to the heart of multi-value environments. Given an item-id, an MV database can read the matching item with a single disk read. This makes a list of item-ids an extremely efficient way of identifying a record-set for processing.

OpenQM can maintain up to 11 select-lists at once. These are numbered from 0 (the default list) to 10.

If select-list 0 is active (i.e. if it is present), then each subsequent *QMQuery* statement will use this select-list. Each *QMQuery* statement will either use and maintain the list, or use and extinguish the list. For example, a `SELECT` command will use and maintain the list (the list will still exist after the `SELECT` command has executed), while a `SORT` command will use and extinguish the list (the list will no longer exist after the `SORT` command has executed).

QMQuery commands can use lists other than the default list by specifying a `FROM` clause. For example:

```
SORT XRATES DATE USD ID.SUP FROM 4
```

If select-list 4 does not exist when this statement is executed, then *OpenQM* will simply report:

```
0 record(s) listed
```

Select-lists are usually created using the `SELECT` or `SSELECT` verbs. By default, these will use the default select-list (list 0), but can use a different list by specifying a `FROM` and/or a `TO` clause:

```
SELECT XRATES WITH YEAR GE "2012" TO 4
368 record(s) selected to list 4
SELECT XRATES WITH USD LT "0.76" OR GT "0.85" FROM 4 TO 4
22 record(s) selected to list 4
```

The first statement above creates the select-list and assigns it to list 4. The second statement uses list 4, carries out another selection, and re-assigns the output to list 4.

If the second statement did not have the `FROM` clause, then it would select from the entire `XRATES` file.

If the second statement did not have the `TO` clause, then it would assign its output to list 0.

In most cases, you can simply work with the default list, and not worry about the FROM and TO clauses.

Note that the prompt remains at a single colon (:) when you assign the selection to a list other than 0. This means there is no active list, and that if you now execute a *QMQuery* statement, it will not use the selections you have just made (unless you explicitly specify the list number to use).

Clearing a select-list

When working with select-lists – particularly when testing selections, you may find that you have created a select-list, but you no longer want it. You can clear this list by using the CLEARSELECT statement:

```
CLEARSELECT

CLEARSELECT 4
```

The first form clears the default select-list. The second form clears the nominated select-list – list 4 in this case.

Saving, restoring, and managing select-lists

As you work with select-lists, you will find there are times when you wish to re-use the list. For example, you need to run a number of reports off the same records in the same file. The file is large, which means the selection and sorting processes take some time. In this case, it would be more convenient to save the list, and then call it back every time you want to run a report.

OpenQM provides a number of commands for managing select-lists in this manner. These are:

```
SAVE.LIST listname

GET.LIST listname

EDIT.LIST listname

MERGE.LIST list1 op list2

COPY.LIST list1,list2

DELETE.LIST listname
```

SAVE.LIST saves a select-list in the $SAVEDLISTS file (this is actually named $SVLISTS on disk) for later recall. It is saved with an item-id of the name that you give it. By default, the SAVE.LIST command saves list 0, but can save a different list if a FROM clause is specified:

```
SAVE.LIST JUNK FROM 4
```

A list is recalled for use using the GET.LIST command. This list is then available for processing by other *QMQuery* commands, just as if you had just created it using the SELECT or SSELECT commands. By default, the list becomes list 0, but you can specify a different list using a TO clause:

```
GET.LIST JUNK TO 3
```

Returning to our sequence of commands used earlier, we could save the list as:

```
SELECT XRATES WITH YEAR GE "2012" TO 4
368 record(s) selected to list 4
SELECT XRATES WITH USD LT "0.76" OR GT "0.85" FROM 4 TO 4
22 record(s) selected to list 4
SAVE.LIST JUNK FROM 4
22 records saved to select list 'JUNK'
```

and then use the list as:

```
GET.LIST JUNK
22 record(s) selected to select list 0
::SORT XRATES DATE USD ID.SUP
```

This reported could be repeated multiple times simply by re-executing the above two statements.

Saved select-lists are just like any other item in *OpenQM* in that they can be viewed, edited, and deleted. The EDIT.LIST command is simply a shortcut way of invoking the ED editor to edit an item in the $SAVEDLISTS file. So, rather than using the ED (line) editor, you could use the *AccuTerm* WED editor:

```
WED $SAVEDLISTS JUNK
```

[You could even edit the EDIT.LIST command that appears in the VOC to invoke the WED editor rather than the ED editor. However, this would probably be overwritten with the normal item the next time that you upgraded *OpenQM*].

Deleting lists can be achieved using the DELETE.LIST verb or by using the DELETE command. The following two commands are identical in effect:

```
DELETE.LIST JUNK

DELETE $SAVEDLISTS JUNK
```

Likewise, copying a list can use the COPY.LIST command, or the normal *OpenQM* COPY (or COPYP) command.

```
COPY.LIST JUNK,JUNK2

COPY FROM $SAVEDLISTS JUNK,JUNK2
```

Of course, the above two statements won't work because you have already deleted the JUNK list from the $SAVEDLISTS file!

If you need to know the names of the saved select-lists, you can query the file in the usual manner:

```
SORT $SAVEDLISTS
```

Finally, *OpenQM* allows you to manipulate two lists jointly to create a new list that is:

- the intersection of the two lists (i.e. those list items that are in both lists)
- the union of the two lists (i.e. list1 plus list2)
- the difference between the two lists (i.e. list1 members that are not in list2)

These three variations can all be achieved using the MERGE.LIST command:

```
MERGE.LIST list1 op list2 {TO newlist}
```

where: op = INTERSECTION, UNION, or DIFFERENCE

Alternatively, each of these operations can be carried out with their own unique command:

```
LIST.INTER list1 {list2 {newlist}}

LIST.UNION list1 {list2 {newlist}}

LIST.DIFF list1 {list2 {newlist}}
```

Newlist is the name to be applied to the list that emerges from the manipulation of the two source lists.

There is an important difference in the usage of these two styles of list processing (MERGE.LIST cf LIST.xxx) that is not immediately apparent. MERGE.LIST uses list numbers, while LIST.xxx uses list names. The examples shown in the help system illustrate this:

```
GET.LIST FRANCE.CUSTOMERS TO 1
27 records selected.
GET.LIST GERMANY.CUSTOMERS TO 2
31 records selected.
MERGE.LIST 1 UNION 2 TO 3
58 records selected

LIST.UNION FRANCE.CUSTOMERS GERMANY.CUSTOMERS MERGED.CUSTOMERS
58 records selected
```

Further details on these commands can be found in the help system.

6.2.4 NSELECT

There are times that we want to know what items are in one file, but not in another. We could do this by creating a select-list for each file, and then using the LIST.DIFF command outlined above. Or we could use NSELECT:

Our XRATES and IRATES files both have similar item-ids. Let's see if the XRATES file has any items that the IRATES file does not:

```
SELECT XRATES
873 record(s) selected to list 0
::NSELECT IRATES
3 record(s) selected to select list 0
::LIST XRATES DATE USD
XRATES....   Date........   US Dollar
16607         19 JUN 2013      0.7991
16609         21 JUN 2013      0.7770
16608         20 JUN 2013      0.7857

3 record(s) listed
```

So – we've SELECT'ed one file, and NSELECT'ed the other. The remaining list tells us the items that are in the first file but not in the second. We've then listed the first file to show that those items do indeed exist there.

Let's list the second file to make sure that those items don't exist there:

```
SELECT XRATES
873 record(s) selected to list 0
::NSELECT IRATES
3 record(s) selected to select list 0
::LIST IRATES DATE DAYS90
0 record(s) listed
'16607' not found
'16608' not found
'16609' not found
```

Well, that confirms that those items don't exist in the IRATES file. What about the other way around? Are there any items in the IRATES file that don't exist in the XRATES file?

```
SELECT IRATES
870 record(s) selected to list 0
::NSELECT XRATES
0 record(s) selected to select list 0
```

No – every item that is in the IRATES file is also in the XRATES file.

6.2.5 QSELECT

The QSELECT verb originated in the days before alternate key indexing was present in multi-value databases, and was used as a form of indexing. Today, QSELECT is not used much, and is mainly included in *OpenQM* for compatibility with other multi-value environments.

At one level, QSELECT is similar to the GET.LIST command – they both convert item-ids stored in an item into a select-list. However, QSELECT is not constrained to lists stored in the $SAVEDLISTS file, and its method of handling data within the item is also different.

This similarity continues to the point where QSELECT can directly substitute for GET.LIST:

```
SELECT XRATES WITH YEAR EQ "2012"
251 record(s) selected to list 0
::SAVE.LIST JUNK
251 records saved to select list 'JUNK'
QSELECT $SAVEDLISTS JUNK
251 record(s) selected to select list 0
::
```

Here we have saved a select-list to the $SAVEDLISTS file, and then used QSELECT to transform the saved item back to an active select-list.

The traditional usage of QSELECT has been as a form of alternate key indexing. Say we have a CUSTOMERS file containing several million records, and we wanted to find all the customers with a SURNAME of Jones. If we used a SORT or a SSELECT verb, then (if there was no alternate key indexing) finding those customers would require reading every record in the whole file. The QSELECT verb offers a way around that problem.

The first step is to create an index file. This might be called CUSTOMERS.NDX. Then the index file will be populated with items corresponding to each of the surnames in the CUSTOMERS file. The contents of each item will be the list of item-ids of those customers with that particular surname. Typically, these item-ids were stored as values within a single field in the item.

It is important to recognise here that the CUSTOMERS.NDX file is maintained programmatically – user intervention in maintenance of index records is undesirable. Whenever a new record is added to the CUSTOMERS file, the program would update the CUSTOMERS.NDX file. If the new customer had a surname that was not already present in the CUSTOMERS file, then it would create a new item in the CUSTOMERS.NDX file containing the item-id of the new customer. Otherwise, it would update the existing item for that surname and add the item-id of the new customer to that item. Likewise, deleting a customer would remove the matching item-id from any surname items in the CUSTOMERS.NDX file.

A normal selection of Jones from the CUSTOMERS file would look like:

```
SORT CUSTOMERS WITH SURNAME EQ NO.CASE "JONES" SURNAME
CUSTOMERS.   Surname.....
         1   Jones
         5   jones
         6   JoneS
         7   jOnes

4 record(s) listed
```

These items would be indexed into the CUSTOMERS.NDX file by creating an item with an ID of 'JONES'. The internal structure of that item is shown below:

```
CT CUSTOMERS.NDX JONES
CUSTOMERS.NDX JONES
1: 1ý5ý6ý7
```

This is a multi-valued item with all values in a single field in the item. This is important because the syntax of the QSELECT verb depends on the structure of this item.

The QSELECT verb can now transform this index item into a select-list for processing:

```
QSELECT CUSTOMERS.NDX JONES
4 record(s) selected to select list
::SORT CUSTOMERS SURNAME
CUSTOMERS.   Surname.....
         1   Jones
         5   jones
         6   JoneS
         7   jOnes

4 record(s) listed
```

Now, let's say the item contains two multi-valued fields, each with the same list of item-id's but sorted into a different order. The dictionary has an item for each field – ORDER1 for field 1, and ORDER2 for field 2. The index item may look like this:

```
CT CUSTOMERS.NDX JONES
CUSTOMERS.NDX JONES
1: 1ý5ý6ý7
2: 7ý6ý5ý1
```

Now, if we use QSELECT as above, it creates a select-list of 8 item-ids. Effectively, this is each individual select-list concatenated together. Really, we only want to return one of those select-lists, so we need to specify which field to return:

```
QSELECT CUSTOMERS.NDX JONES SAVING ORDER2
4 record(s) selected to select list
LIST CUSTOMERS SURNAME
CUSTOMERS.   Surname.....
         7   jOnes
         6   JoneS
         5   jones
         1   Jones

4 record(s) listed
```

Note the order of customer ID's matches the order shown in the second field of the JONES item in CUSTOMERS.NDX.

Other ways that QSELECT can be used include:

- multiple items can be processed to provide a select list resulting from all specified items (e.g. we could return a list of item-ids from items SMITH and JONES)
- all fields with the current item (or items) can be processed to generate a select list. This is useful where the item-ids are stored as fields in the item rather than as values. This is effectively the situation for normal select-lists stored in the $SAVEDLISTS file.

As noted at the start of this section, QSELECT has largely been superseded by other forms of indexing. It has only been covered here for completeness. You should not need to use QSELECT if you are building a new application.

6.2.6 Inline prompts

Inline prompts are not part of record selection as such, but are part of the *OpenQM* command language. However, they may be used to get user responses as part of the selection process.

An inline prompt queries the user for the value of a variable. This variable is then used in the *QMQuery* statement – usually in the selection clause, but can also be used in the heading or footing.

Inline prompts are surrounded by double angle-brackets, and at a minimum contains the prompt text. Consider the following example:

```
SELECT TEX.QCH WITH YEAR EQ <<Year>> AND WITH CTRY EQ <<R,Countries>> AND WITH HS EQ <<HS
Chapter>>
Year=2012
Countries=US
Countries=GB
Countries=JP
Countries=KR
Countries=
HS Chapter=02
16 record(s) selected to list 0
::
```

The first inline prompt simply prompts for the year to select. The second inline prompt has an initial instruction 'R' to repeat the prompt until the user presses Enter without entering a value. The final inline prompt gets the HS chapter code. The *QMQuery* statement that is executed after these prompts and responses is:

```
SELECT TEX.QCH WITH YEAR EQ "2012" AND WITH CTRY EQ "US""GB""JP""KR" AND WITH HS EQ "02"
16 record(s) selected to list 0
```

While this statement is shown as if typed in at the command prompt, inline prompts are most useful if the commands are stored for periodic use. Such stored commands can be generic in nature, but once you have answered the inline prompts, they become specific to the occasion. In a way, this is a simple form of programming.

Answer QUIT if you wish to abort a query when you are at an inline prompt.

Inline prompts have a number of options allowing screen control, use of user defined variables and system environment settings, and data validation after entry. See the documentation for more details on these options.

6.2.7 Selection of string data

Selections so far have largely tested against numeric values. Of course, *OpenQM* can also test against string data – using the same relational operators of EQ, GT, LT, GE, LE, and their synonyms. For example:

```
SORT CUSTOMERS WITH SURNAME EQ "JONES"
```

Note that for this query to select any data, the data referenced by the SURNAME dictionary item must be entered exactly as 'JONES' because the selection is case-sensitive. Entries of 'jones' or 'Jones' will not be matched.

Clearly, we need to make the comparison case insensitive. We can do this in two quite different ways:

- globally, affecting all string comparisons (including sorting)
- just for this comparison

As there are situations where you genuinely want to make selections that ARE case sensitive, it is recommended that you apply case-insensitive comparisons only where you need them.

However, if you really want to make all string comparisons case-insensitive, then it can be done with the following OPTION command:

```
OPTION QUERY.NO.CASE
```

You could execute this from the command line so that it is present just for this session, or you could include it in the account LOGIN item so that selection and sorting will always be case insensitive.

A more flexible approach to selection that allows both case sensitive and case-insensitive selections is shown below:

```
SORT CUSTOMERS WITH SURNAME EQ NO.CASE “JONES”
```

NO.CASE must appear after the relational operator in the selection clause. This will allow selection of any entered casing of 'Jones'.

But what about sorting? Consider the following *QMQuery* statement:

```
SORT CUSTOMERS BY SURNAME SURNAME
CUSTOMERS.   Surname.....
4            Beeblebrox
3            Brown
6            JoneS
1            Jones
2            Smith
7            jOnes
5            jones

7 record(s) listed
```

Clearly the sort is case-sensitive here because ‘jones’ sorts after ‘Smith’. We need to make the Surname case-insensitive for sorting. We can do this by sorting by an expression rather than by the actual data itself:

```
SORT CUSTOMERS BY EVAL "UPCASE(SURNAME)" SURNAME
CUSTOMERS.   Surname.....
4            Beeblebrox
3            Brown
1            Jones
5            jones
6            JoneS
7            jOnes
2            Smith

7 record(s) listed
```

Here, we have told *OpenQM* to evaluate an expression, and then use the result of that evaluation to sort the data. Note that the 'jones' entries have sorted themselves into ID order rather than the order indicated by their specific casing – check back to the previous listing to see the difference. This is because each of these entries has evaluated to the same value for sorting purposes (JONES), but the SORT verb always applies a final BY @ID to the expression.

The syntax of the expression is:

```
EVAL “expression”
```

The expression must be enclosed in single or double quotes, and the expression itself must be a valid *QMBasic* expression. In this case, the expression used was:

```
UPCASE(SURNAME)
```

which fairly logically says convert the surname to upper case. We could also have used:

```
OCONV(SURNAME, ‘MCU’)
```

which says convert surname for output using a conversion code of 'MCU' - convert characters to upper case.

We could also have used the NO.CASE modifier in the BY clause:

```
SORT CUSTOMERS BY NO.CASE SURNAME SURNAME
```

It will take some time to become familiar with the functions available. Start by looking through the online help – particularly the '*QMBasic* Statements and Functions by Type' section. Functions can be identified as the keywords that are followed by a set of parentheses e.g `ABS()`. Clearly, not all of these are relevant for inclusion in *QMQuery* expressions, but it still helps for you to look through what is there.

6.2.8 Selection using wildcard criteria

There are many times when we may not know the exact spelling of a name, or we are looking for people with similar names. In this case, we could supply a variety of different spellings and ask *OpenQM* to return those names that match any of the spellings that we supply, or we could use a pattern match for comparison purposes, and ask *OpenQM* to return any names that match the pattern provided. This pattern can use wildcard characters, fixed or variable length strings, literal characters, or the sound of a word, so the same template can match a number of different spellings.

Wildcard characters

Wildcard characters are only allowed if `OPTION PICK` or `OPTION PICK.WILDCARD` is set. These wildcard characters are `^`, `[`, and `]`, and are used as follows:

`^`	a single character
`[text`	any string ending in "text"
`text]`	any string starting with "text"

```
SORT CUSTOMERS WITH SURNAME EQ NO.CASE "SM]" BY EVAL "UPCASE(SURNAME)" SURNAME
CUSTOMERS.   Surname.....
        10   Small
         2   Smith
         8   Smyth
         9   Smythe

4 record(s) listed
```

The template in this query specified that the Surname had to start with the letters "sm". Therefore, "small", and three different spellings of "smith" were selected.

```
SORT CUSTOMERS WITH SURNAME EQ NO.CASE "SM^TH]" SURNAME
CUSTOMERS.   Surname.....
         2   Smith
         8   Smyth
         9   Smythe

3 record(s) listed
```

This is a more restrictive template that requires the matched surnames to have "sm" as the first two characters, with any character in the next position, followed by "th", and then any characters are allowable thereafter. Amongst others, this would select "smithers", "smithies", and "smithson" as well as the variants of "smith" shown above.

These templates are shown using an `EQ` relational operator, but any of the usual relational operators can be used – you could use a wildcard to find surnames that are not equal to the template. Likewise, wildcard templates can be used in conjunction with other normal selection criteria to produce an overall selection criteria that is quite specific.

Pattern matching

Unlike the wildcard characters described above, *OpenQM* will always recognise the use of pattern matching with the `LIKE` relational operator. This type of selection criteria looks like:

```
      WITH field LIKE “pattern”
```

Patterns consist of the following elements:

```
      A            alphabetic
      N            numeric
      X            any character
```

The number of each of these characters can be entered as:

```
      3            exactly 3 characters
      1-3          between 1 and 3 characters
      0            any number of characters (including none)
```

There is a special pattern of ‘...’ (3 dots) which essentially means any set of characters, that work like the wildcard characters discussed on the previous page.

Any other characters are considered to be a literal string. For clarity, it is recommended that literal strings be enclosed in single quotes, and the entire template be enclosed in double quotes – although these are not always necessary.

Using this style of selection, our two previous examples of selection would become:

```
SORT CUSTOMERS WITH SURNAME LIKE NO.CASE "'SM'0A" SURNAME
CUSTOMERS.   Surname.....
         2   Smith
         8   Smyth
         9   Smythe
        10   Small

4 record(s) listed
```

or:

```
SORT CUSTOMERS WITH SURNAME LIKE NO.CASE "SM..." SURNAME
```

and:

```
SORT CUSTOMERS WITH SURNAME LIKE NO.CASE "'SM'1A'TH'0A" SURNAME
CUSTOMERS.   Surname.....
         2   Smith
         8   Smyth
         9   Smythe

3 record(s) listed
```

Word sounds

OpenQM can use a phonetic search. This means that it will search based on the way that the word sounds. The format for this type of section is:

```
      WITH field SAID “word”
```

For example:

```
SORT CUSTOMERS WITH SURNAME SAID "SMITH" SURNAME
CUSTOMERS.   Surname.....
         2   Smith
         8   Smyth
         9   Smythe

3 record(s) listed
```

This search is not case sensitive – presumably because the case of the letter does not change the sound of the word:

```
SORT CUSTOMERS WITH SURNAME SAID "JONES" SURNAME
CUSTOMERS.   Surname.....
         1   Jones
         5   jones
         6   JoneS
         7   jOnes

4 record(s) listed
```

The synonyms SPOKEN and ~ can be used instead of the keyword SAID.

6.2.9 Searching files

Sometimes you may want to search a file for a particular string which may occur in more than one field. Or, you may not know which field the string occurs in. On these occasions, use of the SEARCH verb lets you find the records containing the string.

The format of the SEARCH verb is:

```
SEARCH {DICT} filename {ALL.MATCH | NO.MATCH} {NO.CASE}
```

OpenQM prompts you repeatedly for the string to search for, and allows up to 20 strings to be searched for at once. To terminate entry of the search strings, press enter at the STRING prompt.

```
SEARCH CUSTOMERS NO.CASE
String: JONES
String:
4 record(s) selected to list 0
::
```

Note that the SEARCH verb has returned a select-list, which you can now use in conjunction with one of the other *QMQuery* verbs (such as LIST or SORT).

Normal use of SEARCH is to select the record if any of the specified strings is in the record. This behaviour can be modified using the ALL.MATCH or NO.MATCH keywords. ALL.MATCH requires that all of the specified strings should be present in the record, while NO.MATCH selects those records that do not match any of the specified strings.

One of the common uses of the SEARCH verb is to search the file containing *QMBasic* programs for certain variables or file names. Because such references may appear anywhere in this type of item (program), it is not practical to use a SELECT statement to find specific strings, as this would require a field name or number to be specified.

6.3 Working With Multiple Data Files

A key feature of most databases is their ability to link related data from different data files, and to display that data in a single view. *OpenQM* is no different. However, the method of linking related data is different to that of an SQL database.

In the examples of *QMQuery* that you have worked with so far, you have seen how the dictionary items determine what data is displayed, and how to display it. Looking up data in other files is no different.

6.3.1 Using a dictionary to look up data in another file

Thinking back on what we know about dictionary items so far, we can deduce that we will need to create an I-type – an indirect dictionary item. Further, the real work that is done by an

I-type is the expression that is contained in line 2 of the dictionary item. So, what might this formula look like?

Once again, we can deduce a number of things that we must pass to the formula. These include:

- the name of the data file
- the ID of the item to look up
- something to identify the data element to return (data position or name)

The formula to look up data in another file is:

```
TRANS(filename, id, loc, code)
```

XLATE is a synonym for TRANS, so this lookup could be written:

```
XLATE(filename, id, loc, code)
```

The code tells *OpenQM* what to do if the data item is not found. This code must be one of:

```
C            return the item-id
V            return a null and print a warning message
X            return a null
```

So, let's try it. Create the following dictionary item in the XRATES dictionary:

```
ID              Type      Loc       Conv     Name      Format   S/M      Assoc
IRATES.DAYS90   I         see below MR22,    90 Day    7R       S

Loc for IRATES.DAYS90:     TRANS('IRATES', @ID, DAYS90, 'X')
```

If we look at this dictionary item, we can see that the instruction we are providing is:

- open the IRATES file
- find the item with the ID that is the same as current ID in the XRATES file
- return the DAYS90 field from the item
- if the item isn't found, then return a null value

And if we put that into a query statement:

```
SORT XRATES DATE USD IRATES.DAYS90
XRATES....    Date........   US Dollar   90 Day.
15346          05 JAN 2010      0.7344      2.80
15347          06 JAN 2010      0.7343      2.79
15348          07 JAN 2010      0.7378      2.78
15349          08 JAN 2010      0.7325      2.78
15352          11 JAN 2010      0.7395      2.76
15353          12 JAN 2010      0.7422      2.78
```

We can check that the 90 day interest rates that have been displayed are the same as those on the IRATES file:

```
SORT IRATES DATE DAYS90
IRATES....    Date.......   90 Day Bank Bill
15346         05 JAN 2010               2.80
15347         06 JAN 2010               2.79
15348         07 JAN 2010               2.78
15349         08 JAN 2010               2.78
15352         11 JAN 2010               2.76
15353         12 JAN 2010               2.78
```

Note that in this example, we specified that *OpenQM* should look up the DAYS90 field in the item. We could also have specified the position of the data within the record. When we created the IRATES file, we put the 90 day interest rates into position 5. Therefore, try replacing line 2 of the I-type with:

```
      TRANS('IRATES', @ID, 5, 'X')
```

This should work identically to the original version of the dictionary item. However, it is preferable to use the dictionary name rather than the data position in translates because of the risk that you will get the number wrong. You might accidentally use field 6 in the translate, and return the 1-year rates. In general, we can spot an incorrect dictionary name much more quickly than an incorrect data position.

The issue of incorrect data positions is much more serious if you restructure your data files (move data to different positions in the file). Presumably, you would update your dictionary to reflect the new positions. In that situation, if you recompile all dictionaries in the account, any translates that use dictionary names will be updated to reflect the new data positions. However, if you've used the data position in the translate, you will need to hunt these down manually to change them.

Use the CD command (or COMPILE.DICT) to recompile a dictionary:

```
      COMPILE.DICT XRATES

      CD ALL
```

The first version of these two commands recompiles only the dictionary items in the XRATES file, while the second version recompiles the dictionary items for all dictionaries in the account.

Beware that some dictionary names do not work (easily) within I-types. Copy the DAYS90 dictionary item in the IRATES file to a new item 90DAY:

```
COPY FROM DICT IRATES DAYS90,90DAY
```

Make sure this dictionary item works:

```
SORT IRATES DATE DAYS90 90DAY
IRATES....   Date.......   90 Day Bank Bill   90 Day Bank Bill
15346        05 JAN 2010               2.80               2.80
15347        06 JAN 2010               2.79               2.79
15348        07 JAN 2010               2.78               2.78
15349        08 JAN 2010               2.78               2.78
```

Now, lets create a new I-type dictionary item in XRATES called IRATES.90DAY. The expression should be:

```
      TRANS('IRATES', @ID, 90DAY, 'X')
```

When you try to file this change (using the MODIFY editor), you get an error message:

```
Compiling IRATES.90DAY...
*** Comma not found where expected
Compilation failed. Record not written to dictionary.
Press return to continue
```

The problem here is that although 90DAY is a valid name for a dictionary item, its use in a *QMBasic* function requires it to be a valid *QMBasic* variable name – which it is not because it starts with a numeric character. To get around this problem, you can quote the dictionary name to get the lookup dictionary item to compile[23]:

```
      TRANS('IRATES', @ID, '90DAY', 'X')
```

If you don't want the annoyance of having some dictionaries fail to compile when you first create them, make sure that you start all you dictionary names with an alphabetic character,

23 This requires version 2.6-8 or later.

and not a numeric character. Otherwise, it isn't difficult to quote the dictionary name when you come to use it in an I-type.

Using translations to look up data in another file requires a dictionary in your primary file for every field that you wish to look up in the external file. Perhaps the external file has links back the other way too. Extensive use of translations can mean that your dictionaries will end up containing many items.

OpenQM provides another means to look up data in external files. This is by linking the two files.

6.3.2 Linking data files

One of the problems with "looking up" data as described above is that you effectively end up with duplicate definitions of the same data field – you need a definition in the dictionary of whatever file you are working in. This duplication has some obvious problems:

- you have spent extra time creating them
- there is scope to get some wrong (look up the wrong data)
- when you change the data structure of a file, you need to find all the affected lookup items.

By using file links, most of these problems disappear. A file link lets you use the dictionary item in the remote file, therefore you don't have duplicate definitions. It will look up the same data as it does when used from its primary file, and if you change the data structure, you only need to change the one dictionary item.

So, what are file links? Essentially, a file link is a dictionary item that defines the ID of the file that you wish to reference. This link can then be referenced in a *QMQuery* statement in the following format:

```
    link%dictname
```

where *dictname* is the name of a dictionary in the remote file.

Create a link item in the XRATES dictionary for the IRATES file:

```
MODIFY DICT XRATES IRATES
F1   L
F2   @ID
F3   IRATES
```

The F1, F2, and F3 are the MODIFY prompts, and should not be entered. Press enter until you get to the file prompt, and file the item.

The first line tells *QMQuery* that this is a link item. The second line contains the expression for the ID of the remote file. In this case, the ID of IRATES is the same as the ID of XRATES, so we simply enter @ID as the expression. The third line tells *QMQuery* the name of the file to look up.

Now, let's try it:

```
SORT XRATES DATE USD IRATES%DAYS30 IRATES%YRS1 ID.SUP
Date........   US Dollar   30 Day Bank Bill   1 Yr Govt Bonds
 05 JAN 2010      0.7344              2.73
 06 JAN 2010      0.7343              2.73
 07 JAN 2010      0.7378              2.73
 08 JAN 2010      0.7325              2.73
 11 JAN 2010      0.7395              2.73
 12 JAN 2010      0.7422              2.73
```

As you can see, link items let you access all your related files with a minimum of effort in terms of adding new dictionary items to your files. This is one of the really useful innovations in *OpenQM* that is not available in any other multi-value database.

Note that if your dictionary items start with a numeric character (such as '90DAY'), this statement will fail on versions of *OpenQM* prior to 2.6-8. However, if the dictionary item started with an alphabetic character, everything works as expected.

Now, can these links be chained? Well, let's find out.

We used file TEX.QCH when we dealing with totals and breaks. Let's look a bit closer at its structure:

It's ID is of the form:

```
YYYYQ * CC * HH
```

where:

YYYYQ is a year and quarter number (1 to 4)

CC is a 2 character country identifier

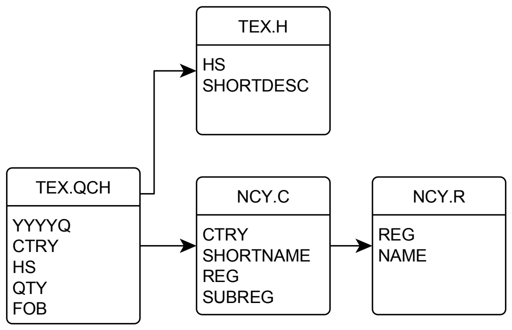
HH is a 2 digit HS Chapter identifier

There are only two attributes in TEX.QCH – these being QTY and FOB (in NZ dollars). At the Chapter level of HS Codes, the quantity is always blank (because you can't add quantities with differing units e.g. kilograms and square metres). So effectively, there is only one data field at this level of aggregation

Country names are found in file NCY.C, and the HS Chapter descriptions are found in file TEX.H. The ID's for these two files are CC and HH respectively, so we can perform a lookup from the ID of TEX.QCH to those files.

Further, country regions are found in file NCY.R with the links to NCY.R being found in the REG and SUBREG fields of NCY.C.

The relationships can be visualised as follows:



Looking at this structure, we can see that we can get a region (or sub-region) for our country in TEX.QCH by linking to the NCY.R file via the NCY.C file. Let's do that in an I-type first.

Create an I-type item in the TEX.QCH dictionary named REG.NAME. It will have an expression of:

```
    TRANS('NCY.R', TRANS('NCY.C', CTRY, REG, 'X'), NAME, 'X')
```

and a format of 15L.

That looks an awkward expression. It says: Use the CTRY field in TEX.QCH to look up the corresponding REG field in NCY.C. Then use this REG field to return the corresponding NAME field from NCY.R.

Now, check the results:

```
LIST TEX.QCH CTRY C%SHORTNAME REG.NAME
TEX.QCH.....   Ctry   Country...........................   Region name....
20131*RS*85    RS     Serbia                               Europe
20124*SB*10    SB     Solomon Islands                      Oceania
20124*LC*74    LC     Saint Lucia                          Americas
20111*RS*05    RS     Serbia                               Europe
20104*SB*90    SB     Solomon Islands                      Oceania
```

In this statement, the 'C%SHORTNAME' in the display clause uses a link 'C' that has already been set up in the dictionary to link to the 'NCY.C' file.

Everything looks OK, but lets check properly. If we look up country 'RS' in the NCY.C file, we find that it is 'Serbia', and that it has region identifier of '150'. Looking up '150' in NCY.R, we find that is 'Europe'. Similarly, 'SB' is the 'Solomon Islands' and has a region code of '009', which in turn is 'Oceania'.

Now, we know there is a link in place from TEX.QCH to NCY.C named 'C'. There are also links from NCY.C to NCY.R called 'R' (region) and 'SR' (sub-region). We can show that these work as follows:

```
SORT NCY.C SHORTNAME R%NAME SR%NAME
NCY.C.....   Country............   Name.........................
Name..........................
AD           Andorra               Europe                         Southern Europe
AE           UAE                   Asia                           Western Asia
AF           Afghanistan           Asia                           Southern Asia
AG           Antigua               Americas                       Caribbean
AI           Anguilla              Americas                       Caribbean
```

Now, can we link these together?

```
LIST TEX.QCH CTRY C%SHORTNAME C%R%NAME
Linked item R%NAME in link C not found
```

Apparently not[24].

However, all is not lost. In the NCY.C dictionary, there are dictionary items to look up the region and sub-region names, named REG.NAME and SUBREG.NAME respectively. Let's link to those dictionary names using the 'C' link.

```
LIST TEX.QCH CTRY C%SHORTNAME C%REG.NAME C%SUBREG.NAME
TEX.QCH.....   Ctry   Country............   Region name.........   Subregion Name......
20131*RS*85    RS     Serbia                Europe                 Southern Europe
20124*SB*10    SB     Solomon Islands       Oceania                Melanesia
20124*LC*74    LC     St Lucia              Americas               Caribbean
20111*RS*05    RS     Serbia                Europe                 Southern Europe
20104*SB*90    SB     Solomon Islands       Oceania                Melanesia
```

24 In versions prior to 3.1-0, this error reported: Link item C not found.

So far, we have only used link expressions in *QMQuery* statements. However, we can use them in dictionary items too – effectively replacing the TRANS expressions. Let's modify our REG.NAME dictionary in TEX.QCH to use a link expression. This will link to the REG.NAME dictionary in NCY.C. So, the expression in the I-type becomes:

```
C%REG.NAME
```

```
LIST TEX.QCH CTRY C%SHORTNAME REG.NAME C%SUBREG.NAME
TEX.QCH.....   Ctry   Country............   Region name....   Subregion Name......
20131*RS*85    RS     Serbia                Europe            Southern Europe
20124*SB*10    SB     Solomon Islands       Oceania           Melanesia
20124*LC*74    LC     St Lucia              Americas          Caribbean
20111*RS*05    RS     Serbia                Europe            Southern Europe
20104*SB*90    SB     Solomon Islands       Oceania           Melanesia
```

Note that this is really just moving the link expression from being written in the query statement to being written in the dictionary. And of course, it works just the same *except that the dictionary item may apply different conversions and format codes.* We can see this in the above two listings where the REG.NAME dictionary in TEX.QCH used a format code of 15L whereas using C%REG.NAME in the *QMQuery* statements used a format code of 20T from REG.NAME in NCY.C.

Overall, this means that link expressions can be used to simplify dictionary items. Instead of using a TRANS expression, we can use a link expression. Instead of using the complicated double TRANS expression shown earlier, we can link to an item in the next dictionary (NCY.C) which in turn links to the final dictionary (NCY.R). And the chain could continue on to further dictionary items.

So, we could set up dictionary item in our base file (TEX.QCH) that contains a compound TRANS expression which will perform our multi-file lookup; or we could set up a chain of linked dictionary items. Both approaches have some advantages:

- Using explicit TRANS expressions lets you determine exactly what is happening to get the returned data. However, the expression may become complex.
- The translation done with a link always uses the 'X' action code. If you wish to use one of the other action codes, you must use the TRANS function. However, you could still link to this dictionary item.
- Chained links look simple, but you need to follow the chain to find out exactly what is being done.

6.4 Performing Calculations With I-Types

6.4.1 Looking up data in the current file

A common task is to calculate a change or percentage change from one time period to the next. This might be a monthly change or an annual change.

The first part of calculating such a change is to get the values for both the current period and the relevant preceding period. The value for the current period is easy enough – that will be displayed by the relevant dictionary item. For example:

```
SORT TEX.QCH WITH CTRY EQ "US" AND WITH HS EQ "02" AND WITH YEAR GE "2011" BREAK.ON YEAR "'L'" QTR
C%SHORTNAME H%SHORTDESC FOB ID.SUP
Year   Qtr   Country............   Description.........   ......FOB Value
2011   1     United States         Meat                       341,725,265
2011   2     United States         Meat                       431,846,739
2011   3     United States         Meat                       170,331,117
2011   4     United States         Meat                       178,979,452

2012   1     United States         Meat                       347,807,910
2012   2     United States         Meat                       415,144,509
2012   3     United States         Meat                       190,119,391
2012   4     United States         Meat                       226,967,852

2013   1     United States         Meat                       420,003,062

9 record(s) listed
```

The value output for FOB is the value of exports for that quarter. Now, how do we get the value for a preceding period?

We use the TRANS function to do this. However, instead of looking up data in an external file, we are going to look up a different ID in our current file:

```
TRANS('TEX.QCH', PREV.ID, FOB, 'X')
```

We haven't defined PREV.ID so far, so this won't compile if you try to enter it. But in general terms, this says look up the current file (assuming we are working with the TEX.QCH file), using an ID defined in the PREV.ID dictionary, and return the value associated with the FOB dictionary.

Previous year value

Let's consider the situation for looking up the previous year's value. The ID of our TEX.QCH file is in the form YYYYQ*CC*HH where YYYY = year, Q = quarter number, CC is the country identifier, and HH is the HS code. If our current quarter is December 2012, and we are looking at meat exports to the United States, then the ID is 20124*US*02. The ID of the same quarter in the previous year would be 20114*US*02. So, in general terms the ID of the item 12 months earlier is simply the current year number less 1 combined with the current quarter number, country, and HS code. In an I-type, we would express this as:

```
(YEAR – 1) : QTR : '*' : CTRY : '*' : HS
```

or:

```
(YYYYQ – 10) : '*' : CTRY : '*' : HS
```

So, create an I-type with the name of PREV.ID.YR1 using the above formula for LOC. Why use that particular name? We want the ID of the dictionary item to suggest the period of the previous ID – in this case 1 year.

Now, we can create another I-type to look up the value of the FOB value 1 year earlier. We'll call this FOB.PREV.YR1, and it will have an expression of:

```
TRANS('TEX.QCH', PREV.ID.YR1, FOB, 'X')
```

Now:

```
SORT TEX.QCH WITH CTRY EQ "US" AND WITH HS EQ "02" AND WITH YEAR GE "2011" BREAK.ON YEAR "'L'" QTR
CTRY HS FOB PREV.ID.YR1 FOB.PREV.YR1
TEX.QCH.....   Year   Qtr   Ctry   HS Code   ......FOB Value   Prev ID Yr 1   ....Prev Yr FOB
20111*US*02    2011   1     US          02       341,725,265   20101*US*02        282,013,013
20112*US*02    2011   2     US          02       431,846,739   20102*US*02        396,571,574
20113*US*02    2011   3     US          02       170,331,117   20103*US*02        143,751,110
20114*US*02    2011   4     US          02       178,979,452   20104*US*02        195,608,116

20121*US*02    2012   1     US          02       347,807,910   20111*US*02        341,725,265
20122*US*02    2012   2     US          02       415,144,509   20112*US*02        431,846,739
20123*US*02    2012   3     US          02       190,119,391   20113*US*02        170,331,117
20124*US*02    2012   4     US          02       226,967,852   20114*US*02        178,979,452

20131*US*02    2013   1     US          02       420,003,062   20121*US*02        347,807,910

9 record(s) listed
```

Note that the PREV.ID.YR1 dictionary item is correctly displaying the ID for the preceding year. So for the first quarter of 2011, it displays the ID for first quarter of 2010.

Similarly, in the first quarter of 2012 and later, you can see that the lookup value of the FOB value matches those displayed directly 12 months earlier.

Previous quarter value

The way that we derived the ID of the previous year was to deduct one from the year number and continue to use the same quarter number. Consider now how we derive the ID for the previous quarter.

For three quarters of the year, we can use the same year number, and subtract one from the quarter number. However, for the first quarter, we need to return the fourth quarter of the previous year. How do we put that into an I-type expression?

We need to use an IF ... THEN ... ELSE construct. Those familiar with computer programming, or even with formulas in Excel will recognise this. For an I-type in *OpenQM*, it will look like:

```
IF (QTR GT 1) THEN (YEAR : (QTR - 1) : '*' : CTRY : '*' : HS) ELSE
((YEAR – 1) : '4*' : CTRY : '*' : HS)
```

Put that into an I-type dictionary item named PREV.ID.Q1 and test it to see if it works:

```
SORT TEX.QCH WITH CTRY EQ "US" AND WITH HS EQ "02" AND WITH YEAR GE "2012" YEAR QTR CTRY HS
PREV.ID.Q1
TEX.QCH.....   Year   Qtr   Ctry   HS Code   Prev ID Q1..
20121*US*02    2012   1     US          02   20114*US*02
20122*US*02    2012   2     US          02   20121*US*02
20123*US*02    2012   3     US          02   20122*US*02
20124*US*02    2012   4     US          02   20123*US*02
20131*US*02    2013   1     US          02   20124*US*02

5 record(s) listed
```

That looks OK. Now we can create another dictionary item to look up the FOB value in the previous quarter. Call it FOB.PREV.Q1. This will have a similar formula to the one we used to look up the FOB value for the previous year:

```
TRANS(‘TEX.QCH’, PREV.ID.Q1, FOB, ‘X’)
```

Can you see the difference? The only difference is that we are passing the ID for the previous quarter rather than for the previous year. Try it to make sure it reads the data correctly:

```
SORT TEX.QCH WITH CTRY EQ "US" AND WITH HS EQ "02" AND WITH YEAR GE "2012" YEAR QTR CTRY FOB HS
PREV.ID.Q1 FOB.PREV.Q1
TEX.QCH.....   Year   Qtr   Ctry   ......FOB Value   HS Code   Prev ID Q1..   ...Prev Qtr FOB
20121*US*02    2012   1     US         347,807,910        02   20114*US*02        178,979,452
20122*US*02    2012   2     US         415,144,509        02   20121*US*02        347,807,910
20123*US*02    2012   3     US         190,119,391        02   20122*US*02        415,144,509
20124*US*02    2012   4     US         226,967,852        02   20123*US*02        190,119,391
20131*US*02    2013   1     US         420,003,062        02   20124*US*02        226,967,852

5 record(s) listed
```

That looks fine.

6.4.2 Calculating a percentage change

We now have the component parts to calculate a percentage change – namely, a current value, and a preceding value. So how do we calculate the percentage change?

The basic formula for calculating a percentage change is:

```
((CURRENT – PREVIOUS) / PREVIOUS) * 100
((CURRENT / PREVIOUS) – 1) * 100
```

If you are familiar with Excel, you might recognise the formula as:

```
CURRENT / PREVIOUS - 1
```

However, that formula needs to be displayed with a percentage format – which takes a decimal fraction (such as 0.12) and displays it as a percentage (12%). Effectively, it is applying an output conversion that multiplies the value by 100. Therefore, the Excel formula is essentially the same as the second formula listed above.

Now, let's take that general formula and convert it to an I-type formula for use in calculating the percentage change in the FOB value of exports from 1 year earlier. As a first step, we'll try:

```
((FOB / FOB.PREV.YR1) – 1) * 100
```

Let's put that into an I-type named FOB.PREV.YR1% as follows:

```
CT DICT TEX.QCH FOB.PREV.YR1%
DICT TEX.QCH FOB.PREV.YR1%
01: I
02: ((FOB / FOB.PREV.YR1) - 1) * 100
03:
04: 'R'FOB Value²1 Yr % Chg
05: 7R
06: S
07:
```

And see how it goes:

```
SORT TEX.QCH WITH CTRY EQ "US" AND WITH HS EQ "02" AND WITH YEAR GE "2012" YEAR QTR CTRY HS FOB
FOB.PREV.YR1 FOB.PREV.YR1
Year   Qtr   Ctry   HS Code   ......FOB Value   ....Prev Yr FOB   .FOB Value
                                                                  1 Yr % Chg
2012   1     US          02       347,807,910       341,725,265         1.78
2012   2     US          02       415,144,509       431,846,739      -3.8676
2012   3     US          02       190,119,391       170,331,117      11.6175
2012   4     US          02       226,967,852       178,979,452      26.8122
2013   1     US          02       420,003,062       347,807,910      20.7572

5 record(s) listed
```

Now, if we get the calculator out, we can see that the percentage change that is calculated is correct, but we don't want to see all those decimal places. Let's say we want to see one decimal place. Change the conversion in the dictionary to MR1, and re-run the query.

```
SORT TEX.QCH WITH CTRY EQ "US" AND WITH HS EQ "02" AND WITH YEAR GE "2012" YEAR QTR CTRY HS FOB
FOB.PREV.YR1 FOB.PREV.YR1
Year    Qtr   Ctry   HS Code   ......FOB Value   ....Prev Yr FOB   .FOB Value
                                                                    1 Yr % Chg
2012    1     US          02       347,807,910       341,725,265          0.2
2012    2     US          02       415,144,509       431,846,739         -0.4
2012    3     US          02       190,119,391       170,331,117          1.2
2012    4     US          02       226,967,852       178,979,452          2.7
2013    1     US          02       420,003,062       347,807,910          2.1

5 record(s) listed
```

Well, that didn't work! The 1.78 per cent change shown for the first quarter of 2012 has just been reduced to 0.2 per cent, rather than 1.8 per cent.

The `MR1,Z` conversion has actually moved the decimal point. This may seem an odd behaviour, but it is consistent with the way that conversion codes work.

Conversion codes

You will recall when we first set up dictionaries for the `XRATES` file that we used conversion codes of `MR22` and `MR44`. We could have simply written those as `MR2` and `MR4` respectively – because if only specify one digit, then the second digit is assumed to be the same as the first. But what are those conversion codes really saying?

As we have already seen, the first digit is the number of decimal places to display. In the absence of any second digit, this works as follows:

Value	**Conversion**	**Output**
12345	MR,	12,345
12345	MR1,	1,234.5
12345	MR2,	123.45
12345	MR4,	1.2345

That seems straightforward. Remember, in the absence of a second digit, the second digit is assumed to be same as the first. That makes those conversion codes `MR00`, `MR11`, `MR22`, and `MR44` respectively. Which brings us to the question, what is the second digit doing.

The documentation is a little confusing here. It says that the second digit is the number of implied decimal points in the data. That is OK when the two numbers are the same, or where the input value is an integer.

Take `MR44` applied to 12345. If there are 4 implied decimal places, that means the real number is 1.2345 and we want to display to 4 decimal places. That is 1.2345 again.

Now consider `MR24` applied to 12345. The real number is 1.2345 (from above), and we want to display to 2 decimal places. That is 1.23.

What if our input number isn't an integer? How many implied decimal places are there?

A better way to think of this is how much should the number be *descaled* before applying the conversion for decimal places.

For integer numbers, this is pretty much the same as outlined above. If we apply `MR24` to 12345, then we descale the number by 4 decimal places to 1.2345 before displaying to 2 decimal places to get 1.23.

Now, let's consider our issue with the percentage changes above. The first two numbers listed are 1.78 and 3.8676. We want to display these to 1 decimal place, but `MR11` carries out a descaling on these numbers before displaying to 1 decimal place.

We don’t want to descale these numbers – they are already scaled correctly – so the second digit needs to be ‘0’.

Change the conversion code to MR10, and try the query again.

```
SORT TEX.QCH WITH CTRY EQ "US" AND WITH HS EQ "02" AND WITH YEAR GE "2012" YEAR QTR CTRY HS FOB
FOB.PREV.YR1 FOB.PREV.YR1
Year    Qtr    Ctry    HS Code    ......FOB Value    ....Prev Yr FOB    .FOB Value
                                                                        1 Yr % Chg
2012    1      US           02       347,807,910        341,725,265            1.8
2012    2      US           02       415,144,509        431,846,739           -3.9
2012    3      US           02       190,119,391        170,331,117           11.6
2012    4      US           02       226,967,852        178,979,452           26.8
2013    1      US           02       420,003,062        347,807,910           20.8

5 record(s) listed
```

Perfect!

Now, does it work for all the data that we have? Take the YEAR selection criteria off and try again:

```
SORT TEX.QCH WITH CTRY EQ "US" AND WITH HS EQ "02" YEAR QTR CTRY HS FOB FOB.PREV.YR1 FOB.PREV.YR1%
ID.SUP
00000073: Divide by zero error in dictionary expression
   Item: FOB.PREV.YR1%
   Filename: TEX.QCH
   Id: 20091*US*02
```

Oops! We’ve got a ‘divide by zero’ error when processing item 20091*US*02. Let’s think about that.

Item 20091*US*02 looks up item 20081*US*02 to get the preceding year’s value, and then calculated the percentage change from that. In this case, 20081*US*02 doesn’t exist in the file, so that value gets returned as a null value (empty string), and the percentage change calculation tries to divide by this.

The solution to this is to avoid trying to calculate a percentage change if the returned value is zero. We can put this into the I-type formula as follows:

```
      IF (FOB.PREV.YR1 NE '') THEN (((FOB / FOB.PREV.YR1) - 1) * 100) ELSE ''
```

Update the formula and try again:

```
SORT TEX.QCH WITH CTRY EQ "US" AND WITH HS EQ "02" YEAR QTR CTRY HS FOB FOB.PREV.YR1 FOB.PREV.YR1%
ID.SUP
Year    Qtr    Ctry    HS Code    ......FOB Value    ....Prev Yr FOB    .FOB Value
                                                                        1 Yr % Chg
2009    1      US           02       328,653,879
2009    2      US           02       330,030,602
2009    3      US           02       133,820,377
2009    4      US           02       179,094,504
2010    1      US           02       282,013,013        328,653,879          -14.2
2010    2      US           02       396,571,574        330,030,602           20.2
2010    3      US           02       143,751,110        133,820,377            7.4
2010    4      US           02       195,608,116        179,094,504            9.2
2011    1      US           02       341,725,265        282,013,013           21.2
```

Now the percentage change works as expected.

Let’s try this for a one-quarter percentage change. Create a dictionary similar to the one you already have for the annual change. A quick way to do this is to copy the existing dictionary item and then modify the copy:

```
COPY FROM DICT TEX.QCH FOB.PREV.YR1%,FOB.PREV.Q1%
1 record(s) copied.
```

Now, modify the new item, and change the expression to:

```
IF (FOB.PREV.Q1 NE '') THEN (((FOB / FOB.PREV.Q1) - 1) * 100) ELSE ''
```

and change the heading to match. Check that it works in a query statement:

```
SORT TEX.QCH WITH CTRY EQ "US" AND WITH HS EQ "02" YEAR QTR CTRY HS FOB FOB.PR
Year   Qtr   Ctry   HS Code   ......FOB Value   ...Prev Qtr FOB   ..FOB Value
                                                                   1 Qtr % Chg
2009   1     US          02       328,653,879
2009   2     US          02       330,030,602       328,653,879           0.4
2009   3     US          02       133,820,377       330,030,602         -59.5
2009   4     US          02       179,094,504       133,820,377          33.8
2010   1     US          02       282,013,013       179,094,504          57.5
2010   2     US          02       396,571,574       282,013,013          40.6
```

Note that no percentage change is calculated for the first quarter because it could not find a base figure to work from. Thereafter, percentage changes are correctly calculated.

6.4.3 Grouped percentage changes

So far, we are taking one record, looking up a matching record from a previous period, and then calculating a percentage change on the difference between them. That is a good start, but there are many cases where we want to calculate a percentage change between the totals of two groups – for example, what is the percentage increase in exports year on year?

Let's start with a basic query:

```
SORT TEX.QCH WITH CTRY EQ "CN" AND WITH HS EQ "02" AND WITH YEAR GE "2011" BREAK.ON YEAR "'V'"
TOTAL FOB TOTAL FOB.PREV.YR1 FOB.PREV.YR1%
TEX.QCH.....   Year   ......FOB Value   ....Prev Yr FOB   .FOB Value
                                                          1 Yr % Chg
20111*CN*02    2011        56,674,740        38,700,242         46.4
20112*CN*02    2011        78,882,851        37,411,198        110.9
20113*CN*02    2011        39,390,693        26,353,058         49.5
20114*CN*02    2011        40,315,951        33,103,077         21.8
               2011       215,264,235       135,567,575

20121*CN*02    2012        92,274,894        56,674,740         62.8
20122*CN*02    2012       111,799,808        78,882,851         41.7
20123*CN*02    2012        73,815,737        39,390,693         87.4
20124*CN*02    2012       133,828,822        40,315,951        232.0
               2012       411,719,261       215,264,235

20131*CN*02    2013       278,878,964        92,274,894        202.2
               2013       278,878,964        92,274,894


                          905,862,460       443,106,704

9 record(s) listed
```

That has given us the percentage change for each quarter, but we don't have the percentage changes for the each year. To do this, we need to modify our percentage change calculation and use the CALC keyword.

Change the I-type expression in FOB.PREV.YR1% to:

```
IF (TOTAL(FOB.PREV.YR1) NE '') THEN (((TOTAL(FOB) / TOTAL(FOB.PREV.YR1)) - 1) * 100) ELSE ''
```

Essentially, this has simply told the expression to operate on the totals of the fields on any break lines (when used with the CALC keyword). Running the query again produces exactly the same output as our initial attempt. However, if we add the CALC keyword to the query, we get:

```
SORT TEX.QCH WITH CTRY EQ "CN" AND WITH HS EQ "02" AND WITH YEAR GE "2011" BREAK.ON YEAR "'V'"
TOTAL FOB TOTAL FOB.PREV.YR1 CALC FOB.PREV.YR1%
TEX.QCH.....    Year   ......FOB Value   ....Prev Yr FOB   .FOB Value
                                                           1 Yr % Chg
20111*CN*02     2011        56,674,740        38,700,242         46.4
20112*CN*02     2011        78,882,851        37,411,198        110.9
20113*CN*02     2011        39,390,693        26,353,058         49.5
20114*CN*02     2011        40,315,951        33,103,077         21.8
                2011       215,264,235       135,567,575         58.8

20121*CN*02     2012        92,274,894        56,674,740         62.8
20122*CN*02     2012       111,799,808        78,882,851         41.7
20123*CN*02     2012        73,815,737        39,390,693         87.4
20124*CN*02     2012       133,828,822        40,315,951        232.0
                2012       411,719,261       215,264,235         91.3

20131*CN*02     2013       278,878,964        92,274,894        202.2
                2013       278,878,964        92,274,894        202.2


                           905,862,460       443,106,704        104.4

9 record(s) listed
```

Or, if we suppress the detail:

```
SORT TEX.QCH WITH CTRY EQ "CN" AND WITH HS EQ "02" AND WITH YEAR GE "2011" BREAK.ON YEAR "'V'"
TOTAL FOB TOTAL FOB.PREV.YR1 CALC FOB.PREV.YR1% DET.SUP NO.GRAND.TOTAL
Year   ......FOB Value   ....Prev Yr FOB   .FOB Value
                                           1 Yr % Chg
2011       215,264,235       135,567,575         58.8
2012       411,719,261       215,264,235         91.3
2013       278,878,964        92,274,894        202.2

9 record(s) listed
```

Perhaps you don’t want the previous year values showing:

```
SORT TEX.QCH WITH CTRY EQ "CN" AND WITH HS EQ "02" AND WITH YEAR GE "2011" BREAK.ON YEAR "'V'"
TOTAL FOB CALC FOB.PREV.YR1% DET.SUP NO.GRAND.TOTAL
Year   ......FOB Value   .FOB Value
                         1 Yr % Chg
2011       215,264,235         58.8
2012       411,719,261         91.3
2013       278,878,964        202.2

9 record(s) listed
```

Note that this last query is a bit deceptive. The percentage change for 2013 is shown as +202.2 per cent – but the numbers show $278.9m for 2013 and $411.7m for 2012 – which is a decrease. What is happening here is that only the first quarter for 2013 is present, and the percentage change is correctly calculated against the first quarter of 2012.

What about for a longer time period?

```
SORT TEX.QCH WITH CTRY EQ "CN" AND WITH HS EQ "02" BREAK.ON YEAR "'V'" TOTAL FOB CALC
FOB.PREV.YR1% DET.SUP NO.GRAND.TOTAL
000000DA: Divide by zero error in dictionary expression
   Item: FOB.PREV.YR1%
   Filename: TEX.QCH
   Id: 20101*CN*02
```

Well, our test for this error in the dictionary item didn’t work properly. Let’s try another variant. Change the I-type expression in FOB.PREV.YR1% to:

```
IF (TOTAL(FOB.PREV.YR1) GT 0) THEN (((TOTAL(FOB) / TOTAL(FOB.PREV.YR1)) - 1) * 100) ELSE ''
```

WRONG WAY

```
SORT TEX.QCH WITH CTRY EQ "CN" AND WITH HS EQ "02" BREAK.ON YEAR "'V'" TOTAL FOB CALC
FOB.PREV.YR1% DET.SUP NO.GRAND.TOTAL
Year   ......FOB Value   .FOB Value
                         1 Yr % Chg
2009       140,522,200
2010       135,567,575         -3.5
2011       215,264,235         58.8
2012       411,719,261         91.3
2013       278,878,964        202.2

17 record(s) listed
```

Now, we are testing the totalled value against zero rather than null. That makes sense. The null value test worked fine against individual records, because we don't store zero values. But if we add several null values together, we get a total of zero – which is not null – and so it went through to the main calculation and hit the divide by zero error.

Arguably, the test should be 'NE 0' rather than 'GT 0', but as we won't experience any negative exports, both expressions will work just fine.

What about if we add more records into the query?

```
SORT TEX.QCH WITH CTRY EQ "CN" BREAK.ON YEAR "'V'" TOTAL FOB CALC FOB.PREV.YR1% DET.SUP
NO.GRAND.TOTAL
Year   ......FOB Value   .FOB Value
                         1 Yr % Chg
2009     3,613,047,492
2010     4,809,399,673         30.4
2011     5,863,790,222         23.8
2012     6,840,528,216         17.8
2013     2,293,298,141         35.2

1666 record(s) listed
```

Well, that looks OK – but if we get the calculator out, we find that the percentage change in 2010 should be 33.1 per cent. The other figures are wrong too – they are all several percentage points out.

So, what is going on? Well, it doesn't look like a numeric overflow situation (the numbers would probably turn negative if that were the case), but we'll change the dictionary item to eliminate that possibility. Let's create another dictionary item named FOB.CHG.YR1 with an expression of:

```
FOB – FOB.PREV.YR1
```

and we'll change our percentage change expression to:

```
IF (TOTAL(FOB.PREV.YR1) GT 0) THEN (TOTAL(FOB.CHG.YR1) * 100 / TOTAL(FOB.PREV.YR1)) ELSE ''
```

The change we've made here is to reduce the magnitude of the working value when we multiply by 100. This will reduce the possibility of numeric overflow. Running the query after making these changes still gives us the same result.

Well, let's try to deduce how it is getting the results that are being displayed. We'll want a smallish data set, and we'll display the detail records. Let's go back to just displaying Chapter 02 data:

```
SORT TEX.QCH WITH CTRY EQ "CN" AND WITH HS EQ "02" BREAK.ON YEAR "'V'" TOTAL FOB TOTAL
FOB.PREV.YR1 TOTAL FOB.CHG.YR1 CALC FOB.PREV.YR1%
TEX.QCH.....    Year   ......FOB Value   ....Prev Yr FOB   Change 1 Yr FOB   .FOB Value
                                                                             1 Yr % Chg
20091*CN*02     2009       47,540,097                           47,540,097
20092*CN*02     2009       46,094,890                           46,094,890
20093*CN*02     2009       24,926,563                           24,926,563
20094*CN*02     2009       21,960,650                           21,960,650
                2009      140,522,200                  0       140,522,200

20101*CN*02     2010       38,700,242        47,540,097         -8,839,855        -18.6
20102*CN*02     2010       37,411,198        46,094,890         -8,683,692        -18.8
20103*CN*02     2010       26,353,058        24,926,563          1,426,495          5.7
20104*CN*02     2010       33,103,077        21,960,650         11,142,427         50.7
                2010      135,567,575       140,522,200         -4,954,625         -3.5
```

Everything appears correct in this listing. Let's add another Chapter:

```
SORT TEX.QCH WITH CTRY EQ "CN" AND WITH HS EQ "01""02" BREAK.ON YEAR "'V'" TOTAL FOB TOTAL
FOB.PREV.YR1 TOTAL FOB.CHG.YR1 CAL FOB.PREV.YR1%
TEX.QCH.....    Year   ......FOB Value   ....Prev Yr FOB   Change 1 Yr FOB   .FOB Value
                                                                             1 Yr % Chg
20091*CN*01     2009                                                     0
20091*CN*02     2009       47,540,097                           47,540,097
20092*CN*01     2009        6,128,500                            6,128,500
20092*CN*02     2009       46,094,890                           46,094,890
20093*CN*01     2009        5,242,213                            5,242,213
20093*CN*02     2009       24,926,563                           24,926,563
20094*CN*01     2009        3,465,000                            3,465,000
20094*CN*02     2009       21,960,650                           21,960,650
                2009      155,357,913                  0       155,357,913

20101*CN*01     2010        7,084,889                            7,084,889
20101*CN*02     2010       38,700,242        47,540,097         -8,839,855        -18.6
20102*CN*01     2010        5,331,666         6,128,500           -796,834        -13.0
20102*CN*02     2010       37,411,198        46,094,890         -8,683,692        -18.8
20103*CN*01     2010        2,983,112         5,242,213         -2,259,101        -43.1
20103*CN*02     2010       26,353,058        24,926,563          1,426,495          5.7
20104*CN*01     2010       20,402,116         3,465,000         16,937,116        488.8
20104*CN*02     2010       33,103,077        21,960,650         11,142,427         50.7
                2010      171,369,358       155,357,913         16,011,445          5.7
```

Now we are getting somewhere. Checking with the calculator shows that the percentage change on the 2010 total line should be 10.3 per cent. So, how did it calculate 5.7 per cent?

The top line of 2010 stands out because there is no percentage change on the detail line. That is quite correct as we'd get a divide by zero error if we tried to calculate a percentage change there. But has this affected our total? Let's check the percentage change without this line.

```
(16,011,445 – 7,084,889) * 100 / 155,357,913 = 5.7%
```

So, the $7,084,889 of exports weren't included in the total for the 2010 calculation. Let's go back to the I-type to figure out why not. The current expression is:

```
IF (TOTAL(FOB.PREV.YR1) GT 0) THEN (TOTAL(FOB.CHG.YR1) * 100 / TOTAL(FOB.PREV.YR1)) ELSE ''
```

On the surface this looks OK, but the TOTAL functions in this expression actually determine WHEN a data value gets added to a total. Our logic here tests the value of our denominator, and if that is positive then it does its calculation. The bit that isn't obvious is that our numerator value only gets added to its total when that logic branch is executed. Where no value exists for last year (the denominator), then the logic takes the ELSE clause, and there is no TOTAL function there to add the current value to the relevant total. Therefore, when we get to the break line, the total for this year's exports are missing any values for which there were no corresponding exports in the preceding year.

We need to unconditionally TOTAL the values for this year. We don't really need to worry about last year's figures – if they are positive, they will be added in; if they are zero, then they don't have any impact on the total anyway.

Let’s try:

```
(TOTAL(FOB.CHG.YR1) * 100) / (IF (TOTAL(FOB.PREV.YR1) GT 0) THEN TOTAL(FOB.PREV.YR1) ELSE
1)
```

Now, this totals the change in value, and then divides that by either last years value, or by 1. The output from this looks like:

```
SORT TEX.QCH WITH CTRY EQ "CN" AND WITH HS EQ "01""02" BREAK.ON YEAR "'V'" TOTAL FOB TOTAL
FOB.PREV.YR1 TOTAL FOB.CHG.YR1 CALC FOB.PREV.YR1%
TEX.QCH.....   Year   ......FOB Value   ....Prev Yr FOB   Change 1 Yr FOB   .FOB Value
                                                                            1 Yr % Chg
20091*CN*01    2009                                                      0         0.0
20091*CN*02    2009        47,540,097                           47,540,097  4,754,009,
                                                                            700.0
20092*CN*01    2009         6,128,500                            6,128,500  612,850,00
                                                                            0.0
20092*CN*02    2009        46,094,890                           46,094,890  4,609,489,
                                                                            000.0
20093*CN*01    2009         5,242,213                            5,242,213  524,221,30
                                                                            0.0
20093*CN*02    2009        24,926,563                           24,926,563  2,492,656,
                                                                            300.0
20094*CN*01    2009         3,465,000                            3,465,000  346,500,00
                                                                            0.0
20094*CN*02    2009        21,960,650                           21,960,650  2,196,065,
                                                                            000.0
               2009       155,357,913                 0        155,357,913  15,535,791
                                                                            ,300.0

20101*CN*01    2010         7,084,889                            7,084,889  708,488,90
                                                                            0.0
20101*CN*02    2010        38,700,242        47,540,097         -8,839,855       -18.6
20102*CN*01    2010         5,331,666         6,128,500           -796,834       -13.0
20102*CN*02    2010        37,411,198        46,094,890         -8,683,692       -18.8
20103*CN*01    2010         2,983,112         5,242,213         -2,259,101       -43.1
20103*CN*02    2010        26,353,058        24,926,563          1,426,495         5.7
20104*CN*01    2010        20,402,116         3,465,000         16,937,116       488.8
20104*CN*02    2010        33,103,077        21,960,650         11,142,427        50.7
               2010       171,369,358       155,357,913         16,011,445        10.3
```

Well, that gives the correct percentage change on the break line – but is pretty ugly on those lines where there were no exports in the preceding year. We have calculated a ridiculously high percentage change for those lines. Perhaps we can hide them …. Well, yes we can. Let’s add a range check to the expression:

```
OCONV(((TOTAL(FOB.CHG.YR1) * 100)  / (IF TOTAL(FOB.PREV.YR1) GT 0 THEN TOTAL(FOB.PREV.YR1)
ELSE 1)), 'R-1000,1000')
```

This has wrapped our previous expression in an `OCONV` function, with a range checking conversion. This only allows values in the range -1,000 to +1,000. Our output now looks like:

```
SORT TEX.QCH WITH CTRY EQ "CN" AND WITH HS EQ "01""02" BREAK.ON YEAR "'V'" TOTAL FOB TOTAL
FOB.PREV.YR1 TOTAL FOB.CHG.YR1 CALC FOB.PREV.YR1%
TEX.QCH.....   Year   ......FOB Value   ....Prev Yr FOB   Change 1 Yr FOB   .FOB Value
20091*CN*01    2009                                                     0          0.0
20091*CN*02    2009        47,540,097                          47,540,097
20092*CN*01    2009         6,128,500                           6,128,500
20092*CN*02    2009        46,094,890                          46,094,890
20093*CN*01    2009         5,242,213                           5,242,213
20093*CN*02    2009        24,926,563                          24,926,563
20094*CN*01    2009         3,465,000                           3,465,000
20094*CN*02    2009        21,960,650                          21,960,650
               2009       155,357,913                 0       155,357,913

20101*CN*01    2010         7,084,889                           7,084,889
20101*CN*02    2010        38,700,242        47,540,097        -8,839,855        -18.6
20102*CN*01    2010         5,331,666         6,128,500          -796,834        -13.0
20102*CN*02    2010        37,411,198        46,094,890        -8,683,692        -18.8
20103*CN*01    2010         2,983,112         5,242,213        -2,259,101        -43.1
20103*CN*02    2010        26,353,058        24,926,563         1,426,495          5.7
20104*CN*01    2010        20,402,116         3,465,000        16,937,116        488.8
20104*CN*02    2010        33,103,077        21,960,650        11,142,427         50.7
               2010       171,369,358       155,357,913        16,011,445         10.3
```

Excellent … but it still isn't correct.

6.5 Compound I-types

Hiding some of the results is a pretty ugly way of doing things. Is there a better way?

Let's think about that for a moment. Our requirements are that both the change in values, and last year's values get accumulated into the break-point totals. And we don't want "divide by zero" errors or spurious percentage changes.

What we need is to unconditionally total both values, and then have a conditional expression to decide what to display. We can't do this in a single statement – but we can put more than one statement within an I-type. This is known as a compound I-type.

In a compound I-type, statements are separated by semi-colons. Further, you can refer to the results of earlier statements using an @n syntax (where n is the number of the statement within the I-type).

Using these rules, we can write our expression as:

```
TOTAL(FOB.PREV.YR1); TOTAL(FOB.CHG.YR1); IF (@1 GT 0) THEN (@2 * 100 / @1) ELSE ''
```

Now, the output from this is just the same as shown above. And if we now go back to our original statement where we found the errors, that looks like:

```
SORT TEX.QCH WITH CTRY EQ "CN" BREAK.ON YEAR "'V'" TOTAL FOB CALC FOB.PREV.YR1% DET.SUP
NO.GRAND.TOTAL
Year    ......FOB Value   .FOB Value
                          1 Yr % Chg
2009     3,613,047,492
2010     4,809,399,673          33.1
2011     5,863,790,222          21.9
2012     6,840,528,216          16.7
2013     2,293,298,141          31.3

1666 record(s) listed
```

Checking those percentage changes with a calculator, we find they are correct.

Incidentally, if you tried the above query with our earlier I-type where we hid the large values, you would have found that it gave the wrong percentage changes at this level. So, it is a good thing that we abandoned that expression.

Hopefully, this section has highlighted the need to CHECK YOUR RESULTS. Answers may look OK but be quite wrong.

Once you know that you have an incorrect result, it is important to understand why you got that particular result so that you can work toward the correct solution. And just because your current solution gives the correct answer some of the time, doesn't mean it gives the correct answer all of the time.

6.6 Alternate Key Indices

Did you find that you were waiting for what seemed like a long time for your query to finish when doing the above calculations? That is because the query had to read all the items in the TEX.QCH file to find just that small number that were required to answer our query.

For example, in the query:

```
SORT TEX.QCH WITH CTRY EQ "CN" BREAK.ON YEAR "'V'" TOTAL FOB CALC FOB.PREV.YR1% DET.SUP
NO.GRAND.TOTAL
```

… just 1,666 records were selected out of the 411,502 records in the file. On my notebook, that query took just over seven (7.014) seconds, meaning it processed nearly 60,000 records per second. That is pretty impressive, but that is a sequential read – if it was reading in random order that rate would be likely to be significantly slower.

The problem for the query is how to minimise the time it takes to select the records required. Once it has the 1,666 records, it doesn't take long to sort them and display them – but the selection itself takes a long time.

Alternate key indices are one way to speed up the selection process. An alternate key index indexes the file on the value contained in a field, or provided by a dictionary item. For example, to speed up the above query, an index based on the CTRY field would enable the query to instantly select just those records with a CTRY value of "CN". Let's try that.

The basic command to make an index is MAKE.INDEX[25] and has the syntax:

```
      MAKE.INDEX filename field(s) {NO.NULLS}
```

For example:

```
MAKE.INDEX TEX.QCH CTRY
Added index for CTRY
Building index 'TEX.QCH CTRY'...
411502 records processed
Populating index...
```

Now, if we run the above query again, it takes just 0.218 seconds. That is an impressive gain.

Essentially, an alternate key index is a cross-reference file keyed on the values of the field. *QMQuery* automatically takes advantage of alternate key indices where they are available.

In the above query, QMQuery reads the CTRY index for the TEX.QCH file, and gets the CN item. This is a list of item-ids that have a CTRY field of CN. From there, it is a simple matter of reading and processing those records.

Key issues for indexing are:

25 MAKE.INDEX is actually a combination of the CREATE.INDEX and BUILD.INDEX commands. See the online help for more information on these commands.

- which fields should be indexed
- how should we construct queries to ensure that available indices are used.

A file can have up to 32 indices. That should be sufficient to restrict the item selection for most queries.

There are a couple of downsides with indices:

- Each index takes up additional disk space. The basic TEX.QCH file takes up about 14.4 MB. The CTRY index we added above takes up a further 5 MB. If we add a HS index, that takes up another 5.1 MB. If we add a third index, then the total space devoted to indices will exceed the space taken up by the data.
- Indices slow down the process of writing to the file. This is because whenever an item is written to (create or update) or deleted from the file, then each index must be checked and updated if necessary.

How much do indices slow down file updates? The TEX.QCH file was built by importing the records form a CSV file to a holding file via the *AccuTerm* FTD file import utility. A *QMBasic* program was then used to transform the data in the holding file, and write it to TEX.QCH.

Without indices, the process of transferring 24,696 records in the holding file into 24,206 items in TEX.QCH took 1.9 seconds. With one index defined on TEX.QCH, the same process took 6.6 seconds. With two indices defined, the process time increased to 18.6 seconds, while loading with three indices took 79.9 seconds. (In this case, the third index was poorly constructed and had a particularly adverse impact on performance).

In general, the more indices there are to maintain, then the greater the slowdown in update processes. The slowdown is quite apparent when making bulk updates like this. However, if you are only saving a single screen of data, then the slowdown may not be apparent at all.

So, this tells us that indexing a file gives us higher performance in selecting data from the file, at the expense of lower performance in updating the file. Further, the cost increases as we increase the number of indices – therefore, we should use indices sparingly.

Now, what is the best way to use an index. In our example so far, we have specified the exact field value we want. What about when we want a range of field values?

Say, we want to select records for China in the 2012 year. Consider the following two statements:

```
SORT TEX.QCH WITH YEAR EQ "2012" AND WITH CTRY EQ "CN" BY HS BREAK.ON HS TOTAL FOB DET.SUP NO.PAGE
SORT TEX.QCH WITH CTRY EQ "CN" AND WITH YEAR EQ "2012" BY HS BREAK.ON HS TOTAL FOB DET.SUP NO.PAGE
```

The first statement took about 6.4 seconds on my notebook, while the second took less than 0.1 seconds. The second statement is so much quicker than the first because it selects on the indexed field before the non-indexed field.

So, the first rule of using indices is to make sure that any selection puts indexed fields ahead of non-indexed fields.

Now, we'll look at using indices to select a range of values – say 2012 exports for Chapter 02. We'll start with the basic statement:

```
SORT TEX.QCH WITH HS EQ "02" AND WITH YEAR EQ "2012" BY CTRY BREAK.ON C%SHORTNAME TOTAL FOB
DET.SUP NO.PAGE
```

Neither of the selection fields are indexed so this statement takes just over six seconds. Note that we've selected the HS code first. This is because we can reduce our selected records faster this way. We only want one HS Chapter out of 99, while we want one year out of five. By selecting on HS code first, there are far fewer second comparisons required to get our final records.

Now, should we index on YEAR? There are only 5 years in the database, so its a poor choice for indexing. In comparison, there are 17 YYYYQ values. That still isn't great, but we'll index on that in the meantime:

```
MAKE.INDEX TEX.QCH YYYYQ
Added index for YYYYQ
Building index 'TEX.QCH YYYYQ'...
411502 records processed
Populating index...
```

Running the query again gives us a similar time – just over six seconds. If we reverse the selection order (year then HS code), the time increases to 6.5 seconds.

Clearly, the indexing of YYYYQ is not having any effect on this selection. This is despite year being a function of YYYYQ.

Now, let's specify the YYYYQ values directly:

```
SORT TEX.QCH WITH YYYYQ EQ "20121""20122""20123""20124" AND WITH HS EQ "02" BY CTRY BREAK.ON C
%SHORTNAME TOTAL FOB DET.SUP NO.PAGE
```

This is actually worse. Query times are now over 12 seconds.

Has the query used the index? Let's add the REQUIRE.INDEX modifier to the query statement:

```
SORT TEX.QCH WITH YYYYQ EQ "20121""20122""20123""20124" AND WITH HS EQ "02" BY CTRY BREAK.ON C
%SHORTNAME TOTAL FOB DET.SUP NO.PAGE
OR relationship cannot be resolved with an index
Processing terminated: This query cannot be resolved with an index
```

In our earlier statement, the query processor did not know how to handle the selection for YYYYQ with indices, so reverted to scanning the whole file. This is why it took so long. Still, it is surprising that it took nearly twice as long as the query against YEAR that returned the same values.

How else can we specify the range of dates? Let's try:

```
SORT TEX.QCH WITH YYYYQ GE "20121" AND LE "20124" AND WITH HS EQ "02" BY CTRY BREAK.ON C%SHORTNAME
TOTAL FOB DET.SUP NO.PAGE REQUIRE.INDEX
```

This took around 1.6 seconds. As did this:

```
SORT TEX.QCH WITH YYYYQ BETWEEN "20121" "20124" AND WITH HS EQ "02" BY CTRY BREAK.ON C%SHORTNAME
TOTAL FOB DET.SUP NO.PAGE REQUIRE.INDEX
```

So, our query times are affected by the way we write our query. Specifically, the query processor cannot handle a request to select a value that is *"equal to x or y"*. But it can handle an equivalent expression where it is asked to select a value that is *"greater than or equal to x and less than or equal to y"* or *"between x and y"*.

While we have created an index on YYYYQ, this really isn't a good field to index. YYYYQ has only 17 unique values in the file at present. That means that there are 24,206 item-ids in each index value. CTRY and HS are much better fields to index because they have 247 and 99 unique values respectively to index on.

We'll test that assumption now. We'll delete the YYYYQ index, create an index for the HS field, and run an equivalent query:

```
DELETE.INDEX TEX.QCH YYYYQ
Deleted index YYYYQ
MAKE.INDEX TEX.QCH HS
Added index for HS
Building index 'TEX.QCH HS'...
411502 records processed
Populating index...
```

Now, we'll modify our query so that the HS selection is first:

```
SORT TEX.QCH WITH HS EQ "02" AND WITH YEAR EQ "2012" BY CTRY BREAK.ON C%SHORTNAME TOTAL FOB
DET.SUP NO.PAGE
```

This query now runs in a bit less than 0.2 seconds. That compares with 1.6 seconds when we were indexing by YYYYQ.

If you wish to see what indices have been defined for a file, use the LIST.INDEX command:

```
LIST.INDEX filename field(s) {STATISTICS} {DETAIL}

LIST.INDEX filename ALL {STATISTICS} {DETAIL}
```

```
LIST.INDEX TEX.QCH ALL
Alternate key indices for file TEX.QCH
Number of indices = 2

Index name...... En Tp Nulls SM Fmt NC Field/Expression
HS                Y  I  Yes   S  R  N  FIELD(@ID, '*', 3)

CTRY              Y  I  Yes   S  L  N  FIELD(@ID, '*', 2)
```

Or, if you wish to see which files have alternate key indices enabled against them, use the FILE.STAT command and look in the AK column. Note, your display output needs to be at least 132 columns wide to display this report.

Summary

Alternate key indices work by maintaining lists of item-ids for each value of an indexed field. This allows you to quickly select the item-ids for a given value of the indexed field at the expense of slower updates (because of the need to maintain the index items).

A good field to index on is one that you use frequently and has many unique values.

In a query, select on indexed fields before selecting on non-indexed fields.

When selecting on an indexed field, select for either a single value, or a range of values. Do not select for several values as the query processor does not handle this. Use the REQUIRE.INDEX modifier to check that your query properly uses the available indices.

6.7 Using QMBasic in I-types

Using the TEX.QCH file, we were able to look up a previous value by calculating the ID of the item. Let's consider how we would do the equivalent lookups in the IRATES and XRATES files.

The lookups we want to do are for one month previous, and one year previous. Let's start with one month previous.

From any given date, we will define the one month earlier date to be the same day number of the previous month. Now, what is the one month previous date from the 31st of March? There is no 31st of February, but we can reason that the appropriate date is the 28th or 29th of February, or more generally, the last day of February. How do we do this in a lookup? How do we do this for any month in the year?

Let's assume we've solved that problem. We can now determine a date value for one month prior to our base date. Is there a value to look up on that date?

The nature of the data we are dealing with is that there are values on weekdays, but not on weekends or public holidays. So, April 30th 2013 was a Tuesday. One month prior this date is March 30th 2013. This was a Saturday, and has no recorded value. Likewise, there is no data recorded on December 25th or the 1st of January, so we have issues with our lookups one month after these dates. So, how do we handle weekends, public holidays, and any other days that do not have recorded values?

Clearly, this is not a straightforward process of calculating a date value, and looking it up. We need to:

- calculate an initial date value. Check to see whether this is a valid date, and adjust if necessary
- look up the data to see if data is present. If not, then step back through time until data is found, or if there is no preceding data, then return null.

These calculations don't easily fit into an I-type expression. But they can be done easily in a *QMBasic* subroutine (or function), and then we can get the I-type to call the subroutine using the SUBR function.

The function on the next page calculates a datevalue given a starting date and a parameter telling it how far away the new date is.

This should be entered into the *QMBasic* programs file, that we created back in Section 4.4.4.

Now use the WED editor to enter the program:

WED BP DATEVALUE.CALC

```
FUNCTION DATEVALUE.CALC(fromdate, chgval)
*****************************************************************************
* Bp DateValue.Calc - Calculate a date id, that is chgval away from fromdate.
*
* Author : BSS
* Created: 28 Jun 2013
* Updated: 28 Jun 2013
* Version: 1.0.0
*
* Pass:
*  fromdate - an internal date value
*  chgval   - {+|-}nP where P = M or Y
*             e.g. -1M  One month previous
*                  -5M  Five months previous
*                   1Y  One year ahead
*
$CATALOG LOCAL
$MODE UNASSIGNED.COMMON

COMMON /QMQ/ DFORMAT

previd = ''
IF NOT(fromdate AND NUM(fromdate)) THEN RETURN previd

IF (chgval EQ '') THEN chgval = '-1M'
ptype = OCONV(OCONV(chgval, 'MCA'), 'MCU')
pno = OCONV(chgval, 'MC/A')

dd = OCONV(fromdate, 'DD')
mm = OCONV(fromdate, 'DM')
yyyy = OCONV(fromdate, 'DY')
```

```
BEGIN CASE
  CASE (ptype EQ 'M')
    mno = yyyy * 12 + mm + pno
    m1 = MOD(mno, 12)
    y1 = mno // 12
    IF (m1 EQ 0) THEN
      m1 = 12
      y1 -= 1
    END

  CASE (ptype EQ 'Y')
    m1 = mm
    y1 = yyyy + pno

  CASE 1
    RETURN previd
END CASE


IF UNASSIGNED(DFORMAT) THEN
  EXECUTE "DATE.FORMAT INFORM"
  DFORMAT = @SYSTEM.RETURN.CODE
END
IF (DFORMAT) THEN dconv = 'D' ELSE dconv = 'DE'

d1 = dd
LOOP
  prevdate = d1:'-':m1:'-':y1
  previd = ICONV(prevdate, dconv)
  IF (STATUS() EQ 0) THEN EXIT
  d1 -= 1
  IF ((dd - d1) GT 10) THEN
    previd = ''
    EXIT
  END
REPEAT

RETURN previd
*
* ------------------------------------------------------------------------- *
*
END
```

Enter the program, save it, and exit the editor. Now, we need to compile the program:

```
BASIC BP DATEVALUE.CALC
Compiling BP DATEVALUE.CALC
******
0 error(s)
DATEVALUE.CALC added to local catalogue
Compiled 1 program(s) with no errors
```

Let's quickly look at what this program does – or is supposed to do. The main actions are:

- set a default return value
- check that a valid date has been passed
- split the 'chgval' variable into a unit of movement (M or Y) and a number of units
- break the passed date into day, month, and year components
- calculate test values for the month and year based on the movement specified
- check whether the system is set to North American or International date format and adjust the date conversion accordingly. This date format is stored in a named common so that we only need to test this once.
- Start a loop
  - form an international date string based on the test values of month and year

   - try to convert this to an internal date
   - If STATUS() is zero, the conversion was successful and we have a valid date. Exit the loop.
   - Otherwise, go back a day and repeat the loop.

- Return from the program.

There is also a check in the loop to make sure that it does not loop infinitely, but this should never occur anyway.

You may note that there is no check for a valid day after we have taken a day off. And you might think that if we take a day off the first day of the month, then we will reach an impossible day number. That is all true – but it can't happen. If we started with the first of the month, there is ALWAYS a first of the preceding month. The only time when we will get invalid date conversions is at the end of the month where it is possible that our current month has more days than the preceding month.

Now, let's test the program. Create an I-type dictionary item in the XRATES dictionary called PREV.ID.M1. The expression should be:

```
SUBR('DATEVALUE.CALC', @ID, '-1M')
```

Don't bother with a conversion at this stage, and use a format of 7R. Now, let's try it:

```
SORT XRATES DATE PREV.ID.M1
XRATES....    Date........   Prev ID
                             -1 Mth.
15346          05 JAN 2010     15315
15347          06 JAN 2010     15316
etc
15397          25 FEB 2010     15366
15398          26 FEB 2010     15367
15401          01 MAR 2010     15373
15402          02 MAR 2010     15374
etc
15426          26 MAR 2010     15398
15429          29 MAR 2010     15400
15430          30 MAR 2010     15400
15431          31 MAR 2010     15400
15432          01 APR 2010     15401
15437          06 APR 2010     15406
```

Look at the date values returned for the dates from 29 March through to 31 March. These all return 15400. If we look back to the end of February, we see that we don't have date 15400 on file, but 26 February was date 15398, so date 15400 must be 28 February 2010. In other words, the calculated date for one month prior to 29 March 2010 was 28 February 2010. Likewise, the 30th and 31st of March both returned 28 February also.

Now what did we just do? We called a function using the SUBR function. The SUBR function takes a variable number of arguments – we must pass the subroutine (function) name first, followed by any arguments required by the subroutine we are calling[26].

The DATEVALUE.CALC function we just created needs two arguments – a date to start from, and a distance to move. We passed @ID as the initial date value, and '-1M' as the movement value – meaning go back one month. If we passed '-3M', we'd go back 3 months; '-15M'

26 Note: If you wish to call a subroutine using the SUBR function, then you need to write the subroutine to return the answer as the first argument – but you don't include this answer argument in the list of arguments passed in the SUBR call. Because we are using a function, then the arguments passed match the arguments declared by the function. We are using a function here because the arguments match which makes the concept easier to understand.

would go back 15 months; '2M' would go forward 2 months; and '-1Y' would go back one year.

Note that we've built quite a bit of flexibility into the DATEVALUE.CALC function. Even though we've only discussed finding dates for the previous month, we've already allowed for the possibility of finding dates months and even years into the past – or future. Nor have we restricted people to months between one and 12 – any number of months will be handled correctly.

Of course, we could add even more flexibility by adding the ability to handle day movements as well as years and months. Feel free to do so.

Now, we need to look up a currency value based on the date returned by the PREV.ID.M1 dictionary item. We'll create another function to do this, as the same code needs to be used by each currency.

WED BP XRATES.LOOKUP

Now, enter the program from the next page:

```
FUNCTION XRATES.LOOKUP(currency, dateval)
*****************************************************************************
* Bp Xrates.Lookup - Look up the value of currency at dateval. If no value,
*                    then look up previous value.
*
* Author : BSS
* Created: 28 Jun 2013
* Updated: 01 Jul 2013
* Version: 1.0.0
*
* Pass:
*  currency - ISO currency identifier (XRATES dictionary name)
*  dateval  - internal date value
*
$CATALOGUE LOCAL

xval = ''
IF NOT((dateval NE '') AND NUM(dateval)) THEN RETURN xval

OPEN 'DICT','XRATES' TO xrates.dict ELSE RETURN xval
OPEN 'XRATES' TO xrates ELSE RETURN xval

READ xdict FROM xrates.dict, currency ELSE
  READ xdict FROM xrates.dict, UPCASE(currency) ELSE
    RETURN xval
  END
END
amc = xdict<2>
IF NOT((amc NE '') AND NUM(amc)) THEN RETURN xval

xid = dateval
LOOP
  READ xvec FROM xrates, xid ELSE xvec = ''
  xval = xvec<amc>
  IF (xval NE '') THEN EXIT
  xid -= 1
  IF ((dateval - xid) GT 10) THEN
    xval = ''
    EXIT
  END
REPEAT

RETURN xval
*
* ------------------------------------------------------------------------- *
*
END
```

Let's look at what this does. The steps are:

- set a default return value, and check that you have passed a valid date (number)
- open the XRATES file and its dictionary
- read the dictionary definition of the currency you have passed. If it isn't found then return. Get the field number from the dictionary item and check that it is a number.
- Start a loop
    - read the exchange rates for the specified date
    - see if there is an entry in the specified field number. If so, then exit the loop
    - Otherwise, decrement the date value.
    - Check that the new date value is no more than 10 days from the original date value. If OK then do another loop, else exit.
- Return from the program.

Now, we'll compile it:

```
BASIC BP XRATES.LOOKUP
Compiling BP XRATES.LOOKUP
****
0 error(s)
XRATES.LOOKUP added to local catalogue
Compiled 1 program(s) with no errors
```

We'll now use this in an I-type to retrieve the US Dollar value for one month previous. We'll call this USD.PREV.M1 and it will look like:

```
       I
       SUBR('XRATES.LOOKUP', 'USD', PREV.ID.M1)
       MR44,
       US Dollar²Prev mth
       7R
       S
```

Now:

```
SORT XRATES DATE USD PREV.ID.M1 USD.PREV.M1
XRATES....   Date........   US Dollar   Prev ID   US Dollar
                                        -1 Mth.   Prev mth.
15346         05 JAN 2010      0.7344     15315
15347         06 JAN 2010      0.7343     15316
15348         07 JAN 2010      0.7378     15317
15349         08 JAN 2010      0.7325     15318
15352         11 JAN 2010      0.7395     15321
etc
15376         04 FEB 2010      0.7021     15345
15377         05 FEB 2010      0.6869     15346      0.7344
15380         08 FEB 2010      0.6883     15349      0.7325
15381         09 FEB 2010      0.6823     15350      0.7325
15382         10 FEB 2010      0.6961     15351      0.7325
15383         11 FEB 2010      0.6927     15352      0.7395
```

We can see from this listing, that it is looking up the correct exchange rate based on the rules that we outlined at the beginning – namely, if an exchange rate doesn't exist for the nominated date, then it will step back in time until one is found. For example, on the 10th of February, the lookup date is the 10th of January – which doesn't exist. So, it steps back to the 8th of January and returns the value found for that date.

All we need to do now is create a percentage change dictionary item similar to the we created for the FOB percentage changes. We'll call this USD.PREV.M1% and it will look like:

```
       I
       IF (USD.PREV.M1 NE '') THEN (((USD / USD.PREV.M1) – 1) * 100) ELSE ''
       MR10,
       US Dollar²% chg 1 mth
       7R
       S
```

```
SORT XRATES DATE USD PREV.ID.M1 USD.PREV.M1 USD.PREV.M1%
XRATES....   Date........   US Dollar   Prev ID   US Dollar   US Dollar..
                                        -1 Mth.   Prev mth.   % Chg 1 mth
15346         05 JAN 2010      0.7344     15315
15347         06 JAN 2010      0.7343     15316

15376         04 FEB 2010      0.7021     15345
15377         05 FEB 2010      0.6869     15346      0.7344          -6.5
15380         08 FEB 2010      0.6883     15349      0.7325          -6.0
15381         09 FEB 2010      0.6823     15350      0.7325          -6.9
15382         10 FEB 2010      0.6961     15351      0.7325          -5.0
15383         11 FEB 2010      0.6927     15352      0.7395          -6.3
```

If we get the calculator out, we find that these percentage changes are calculated correctly.

Simplifying the process

All of the above seems somewhat complicated. To get a percentage change for a currency, we need to create TWO dictionary items – one to get the currency value for a preceding period, and a second to do the actual calculation. Can we simplify this process so that we only need to create one dictionary item to calculate a percentage change over a given period?

Well, of course we can. But it is useful to have gone through the individual steps, so you can see the process involved.

As always, there are multiple ways of doing things, so let's stop for a moment and consider what we want to do.

We've created two functions – one to calculate a datevalue; and another to look up a currency value at or near that datevalue. If we are going to simplify things, should we amalgamate these functions into our overall calculation?

The DATEVALUE.CALC function is a useful general function to have. I would suggest that you keep this separate so that you can use it in other dictionary items or programs.

The XRATES.LOOKUP function is a bit more specific to the process of calculating percentage changes. Even so, there may be other times when you want to use this type of lookup, so we'll keep this separate too.

That means we are going to create another function that will call both of these functions in its own calculation of a percentage change. And what we need to pass to this new function is a combination of the inputs to the two functions we have already created – a currency identifier, and a period (change) identifier. This function is shown below:

```
FUNCTION XRATES.PC.CHANGE(currency, chgval)
****************************************************************************
* Bp Xrates.Pc.Change - Calculated the percentage change in the currency over
*                       period chgval.
*
* Author : BSS
* Created: 01 Jul 2013
* Updated: 01 Jul 2013
* Version: 1.0.0
*
* Pass:
*  currency - ISO currency identifier (XRATES dictionary name)
*  chgval   - {+|-}nP where P = M or Y
*             e.g. -1M  One month previous
*                  -5M  Five months previous
*                   1Y  One year ahead
*
$CATALOGUE LOCAL

pcchange = ''
OPEN 'DICT','XRATES' TO xrates.dict ELSE RETURN pcchange

READ xdict FROM xrates.dict, currency ELSE
  READ xdict FROM xrates.dict, UPCASE(currency) ELSE
    RETURN pcchange
  END
END
amc = xdict<2>
IF NOT((amc NE '') AND NUM(amc)) THEN RETURN pcchange

CALL DATEVALUE.CALC(xdate, @ID, chgval)
IF (xdate EQ '') THEN RETURN pcchange

CALL XRATES.LOOKUP(prevval, currency, xdate)
IF NOT(prevval AND NUM(prevval)) THEN RETURN pcchange

nowval = @RECORD<amc>
pcchange = ((nowval / prevval) - 1) * 100

RETURN pcchange
*
* ------------------------------------------------------------------------ *
*
END
```

You can see that we have duplicated some of the code from XRATES.LOOKUP where we find the field number associated with a currency identifier. After that, the DATEVALUE.CALC function is called to get the earlier date, then the value of the exchange rate for the passed currency is retrieved by the XRATES.LOOKUP function. Finally, the percentage change is calculated from the two exchange rate values – the one in the current record being processed, and the one we have just looked up.

Note that this calculation simply uses the internal values of the exchange rates without converting them to external format. That isn't particularly important here, but in other circumstances where the data items have different implied decimal precisions, then you need to consider the appropriate degree of scaling to apply to the result.

This function is clearly built around the assumption that it is going to be called from a dictionary item. This is because it is reliant on the variables @ID and @RECORD being set. If you are going to call this function from something other than *QMQuery* (say a *QMBasic* program), then it is your responsibility to ensure those variables are set.

Note that the DATEVALUE.CALC and XRATES.LOOKUP functions have actually been called as subroutines. A function is normally called as:

    answer = function-name(arg1, arg2 etc)

Without going into a technical explanation, we'll simply state that a function can be called as a subroutine using the following syntax:

```
CALL function-name(answer, arg1, arg2 etc)
```

Calling as a subroutine has the advantage that we don't have to declare the function within the program. However, it has the disadvantage that we can't directly use the function as an input to another statement or function. These issues are really beyond the scope of this book, but are covered further in *"Getting Started in OpenQM – Part 2"*.

Now, create a new dictionary item named GBP.PREV.M1% as follows:

```
I
SUBR('XRATES.PC.CHANGE', 'GBP', '-1M')
MR01,
UK Pound²% chg 1 mth
7R
S
```

```
SORT XRATES DATE GBP GBP.PREV.M1%
XRATES....   Date........   UK Pound   UK Pound...
                                       % chg 1 mth
15346         05 JAN 2010     0.4564
15347         06 JAN 2010     0.4591
15348         07 JAN 2010     0.4606
15349         08 JAN 2010     0.4597
15352         11 JAN 2010     0.4604

15376         04 FEB 2010     0.4419
15377         05 FEB 2010     0.4358          -4.5
15380         08 FEB 2010     0.4414          -4.0
15381         09 FEB 2010     0.4381          -4.7
15382         10 FEB 2010     0.4428          -3.7
15383         11 FEB 2010     0.4441          -3.5
```

Once again, the calculator confirms that these percentage changes are correct.

Now we only need one dictionary item per currency to calculate the percentage change in value over a given time period.

6.7.1 Calculating an index value

An index value is similar to a percentage change. It expresses a current value as a proportion of a past value. However, whereas the percentage changes are calculated from a set number of periods in the past, the index value is always expressed relative to a single period in the past – the base period.

The general formula to calculate an index value is:

```
CURRENT * indexbase / BASE
```

where indexbase is a usually value of 100 or 1000. If the current value is the same as the base value, then it will be given an index value of indexbase.

For example, we might want to express the exchange rates relative to their value as at the beginning of February 2010. At that date, the US Dollar exchange rate was 0.7006. One year later, this exchange rate had increased to 0.7724. If we set our base period to a value of 1,000, then the February 2011 value has an index value of 1,102. In other words, the exchange rate had increased by about 10 per cent since the base period.

There is no particular difficulty in putting this into a dictionary item, but think for a moment how to include the base period value:

- We could hard code this value into the dictionary item – but then if we change the base period, we will need to edit the dictionary item. As we have more than one exchange rate, we would need to edit each dictionary with a hard coded exchange rate.
- We could enter the value into a dictionary item that always returns the @ID of the base period. We could then use this to look up the appropriate base value. We would only need to edit this one dictionary item to change the base period for all exchange rates.

This second option gives us greater flexibility, so we will use that. First, create a BASEPERIOD dictionary. This should look like:

```
CT DICT XRATES BASEPERIOD
DICT XRATES BASEPERIOD
01: I
02: 15373
03:
04: Base period
05: 7R
06: S
07:
```

This defines the dictionary as an I-type, but gives it a constant value of 15373 which is the internal date value for the 1st of February 2010. Perhaps you may want this value to be a bit more transparent so that you can understand what date is being referred to when you look at it. In that case, try replacing the expression with:

```
    ICONV('01 FEB 2010', 'D')
```

Now, we create a dictionary to look up the value of the US Dollar in this base period. We will call this USD.BASE:

```
CT DICT XRATES USD.BASE
DICT XRATES USD.BASE
01: I
02: TRANS('XRATES', BASEPERIOD, USD, 'X')
03: MR44,
04: USD Base
05: 7R
06: S
07:
```

Now, we can try these dictionary items to make sure they work:

```
SORT XRATES DATE USD BASEPERIOD USD.BASE
XRATES....   Date........   US Dollar   Base Period   USD Base
15346         05 JAN 2010      0.7344         15373     0.7006
15347         06 JAN 2010      0.7343         15373     0.7006

15369         28 JAN 2010      0.7062         15373     0.7006
15370         29 JAN 2010      0.7048         15373     0.7006
15373         01 FEB 2010      0.7006         15373     0.7006
15374         02 FEB 2010      0.7085         15373     0.7006
15375         03 FEB 2010      0.7131         15373     0.7006
15376         04 FEB 2010      0.7021         15373     0.7006
15377         05 FEB 2010      0.6869         15373     0.7006
```

The BASEPERIOD dictionary item is assigning a value of 15373 (01 FEB 2010) to every record displayed, while the USD.BASE item is looking up the same US Dollar value for every record. Now, we can use this base value in an index calculation. Create another dictionary item called USD.IDX.

```
CT DICT XRATES USD.IDX
DICT XRATES USD.IDX
01: I
02: IF (USD.BASE NE '') THEN (USD * 1000 / USD.BASE) ELSE ''
03: MR0,Z
04: US DollarýIndex
05: 7R
06: S
07:
```

Note that the formula in line 2 checks to make sure there is a base year value before doing the division, and that the conversion in line 3 rounds the answer to zero decimal places. The output of this looks like:

```
SORT XRATES WITH DOM EQ "01""02""03" BREAK.SUP MTH DATE USD USD.IDX ID.SUP
Date........   US Dollar   US Dollar
                           Index....
 01 FEB 2010      0.7006       1,000
 02 FEB 2010      0.7085       1,011
 03 FEB 2010      0.7131       1,018

 01 MAR 2010      0.6992         998
 02 MAR 2010      0.6996         999
 03 MAR 2010      0.6959         993

 01 APR 2010      0.7107       1,014

 03 MAY 2010      0.7286       1,040
```

Check these with a calculator to ensure that the correct index values are being calculated.

Consolidating dictionary items

Once again, we are creating two dictionary items for every index value that we wish to display – the basevalue item, and the index item. Let's see how we might consolidate these.

We could create a callable function as we did with the percentage changes. Or we could simply consolidate the calculations into one I-type. Let's do this consolidation for the Australian dollar:

```
CT DICT XRATES AUD.IDX
DICT XRATES AUD.IDX
01: I
02: IF (TRANS('XRATES', BASEPERIO, AUD, 'X') NE '') THEN (AUD * 1000 / TRANS('XRATES',
      BASEMTH, AUD, 'X')) ELSE ''
03: MR0,Z
04: AU DollarýIndex
05: 7R
06: S
07:
```

Note that the expression extends over 2 lines.

This dictionary item does the exact same calculation as was shown earlier for the US Dollar index, but has internalised the lookups of the base month value. This is a much more difficult formula to read.

But we can actually simplify this expression. We are doing the same translation twice – so a better way to write this would be:

```
TRANS('XRATES', BASEPERIOD, AUD, 'X'); IF (@1 NE '') THEN (AUD * 1000 / @1) ELSE ''
```

This is a compound I-type. The first statement does the translation, while the second one uses the variable `@1` to refer to the result of this translation. This makes the I-type shorter, easier to read, and makes sure that the translation is only done once.

How you develop your dictionary items is up to you. However, it is usually better to keep dictionary items easy to understand. Sometimes, this means writing more dictionary items than you strictly need – but fewer difficult I-types may be counter-productive.

Note that use of the SUBR statement to perform complex calculations is a mixed blessing. Yes – you CAN do complex things – but it isn't apparent from looking at the dictionary item exactly what is happening. You need to dig into the subroutine (or function) to find that out.

You should also note that using nested I-types in a dictionary item (rather than one monolithic calculation) does not incur a performance penalty. *OpenQM* compiles dictionary items as you create them. So when you create a dictionary item which uses multiple I-type expressions, these expressions are all merged at that time to create an efficient calculation[27]. This is another reason for you to use multiple simple dictionary items rather than a lesser number of complex items.

6.7.2 In-line calculations

So far, whenever we have wanted to do a calculation, we have had to create a dictionary item to do the calculation for us. *OpenQM* allows an alternative to this – we can specify the calculation directly in the *QMQuery* sentence.

To specify a calculation directly in the query, we use the EVAL keyword:

```
EVAL “expression” {AS name}
```

The EVAL keyword will often be used in conjunction with the following modifiers:

```
CONV “code”

FMT “spec”

COL.HDG “text”
```

These modifiers give *OpenQM* instructions on how to display the results of the EVAL expression.

EVAL evaluates an expression contained in a quoted string following the keyword. This expression is identical to the formula used in line 2 of an I-type dictionary – and this is the best way to consider the EVAL expression – as an in-line I-type dictionary.

To illustrate the use of EVAL, we will use the IRATES file.

Say we want to find the margin between 90 day and 10 year interest rates. We could use an expression like:

```
SORT IRATES DATE YRS10 DAYS90 EVAL "YRS10 - DAYS90" ID.SUP
Date.......   10 Yr Govt Bonds   90 Day Bank Bill   YRS10 - DAYS90
05 JAN 2010               6.13               2.80             3.33
06 JAN 2010               6.07               2.79             3.28
07 JAN 2010               6.06               2.78             3.28
08 JAN 2010               6.06               2.78             3.28
11 JAN 2010               6.05               2.76             3.29
12 JAN 2010               6.10               2.78             3.32
```

As you can see, the EVAL expression correctly calculated the difference between the two interest rates. You should also note two other things:

- The heading of the column is the expression that was evaluated

27 This does not always hold true for other multi-value databases. In traditional PICK-style databases, dictionary items are either compiled at execution time, or are interpreted at execution, both of which create performance penalties for nesting dictionary items.

- The data displayed has had an output conversion applied to it – i.e. it has been formatted to 2 decimal places

The first point is obvious enough, but what about the second? Where did it get the conversion code from? The documentation states that:

"The default display format and conversion are taken from the first field referenced by the expression. If no fields are referenced, the format defaults to 10L with no conversion. Alternative format or conversion may be specified with the FMT or CONV field qualifiers."

Therefore, in this case, the format and conversion codes were taken from the `YRS10` dictionary item. The format code in this case was `7R`. If you look at the column heading, you can see that the field was actually 14 characters wide because the heading text was wider than the format code field width. This is consistent with normal dictionary items.

Clearly, the conversion and format codes are correct for this expression, but we may wish to use a different column heading. In this case, we use an alternate column heading as shown earlier in Section 5.5.3:

```
SORT IRATES DATE YRS10 DAYS90 EVAL "YRS10 - DAYS90" COL.HDG "'R'Margin²10Y-90D" ID.SUP
Date.......   10 Yr Govt Bonds   90 Day Bank Bill   .Margin
                                                    10Y-90D
05 JAN 2010               6.13               2.80      3.33
06 JAN 2010               6.07               2.79      3.28
07 JAN 2010               6.06               2.78      3.28
08 JAN 2010               6.06               2.78      3.28
11 JAN 2010               6.05               2.76      3.29
12 JAN 2010               6.10               2.78      3.32
```

An example of the use of the `CONV` modifier is given in 5.5.4. The use of `FMT` is essentially similar.

6.8 Output to O/S Level Files

OpenQM can output the results of query statements direct to an O/S level file. This is useful for importing the results of queries into tools such as Excel.

There are two parts to this export process. Firstly, the format of the exported data is defined using the `CSV` or `DELIMITER` keywords. Secondly, the destination of the output file is defined using the `TO` keyword.

For example:

```
SORT XRATES WITH YEAR EQ "2012" DATE USD GBP AUD JPY EUR CSV "<TAB>" TO C:\TEMP\XRATES.CSV ID.SUP
251 record(s) listed
```

Viewing the output from this statement in a text editor shows:

```
Date          US Dollar     UK Pound      Aus Dollar    Jap Yen       Euro
04 JAN 2012   0.7906        0.5052        0.7619        60.64         0.6057
05 JAN 2012   0.7875        0.5042        0.7598        60.43         0.6085
06 JAN 2012   0.7807        0.5038        0.7606        60.21         0.6106
09 JAN 2012   0.7800        0.5056        0.7637        60.00         0.6139
10 JAN 2012   0.7872        0.5094        0.7689        60.49         0.6167
```

The `CSV` keyword has two optional parameters. Only one of these has been used in the statement above. The format of `CSV` is as follows:

```
CSV {mode} {"delimiter"}
```

Mode may be either 1 (default), 2 or 3. Mode 1 produces output that conforms to RFC 4180 which defines the comma separated value format. In this mode, output fields are not quoted unless they contain either a double quote character or the delimiter. Embedded double quotes are replaced by two consecutive double quote characters.

Mode 2 quotes all non-null fields except for numeric fields that do not contain a comma.

Mode 3 quotes all fields.

The delimiter is a comma by default, but may be set to an alternative character by quoting it. For example:

```
CSV “~”
```

will set the delimiter to a tilde (~). Setting the delimiter to a TAB character is accomplished using the special string: “<TAB>”.

The format of the TO keyword is as follows:

```
TO pathname {NO.QUERY | APPENDING}
```

where pathname is the full pathname of the output file.

To write the output to a file that is defined within *OpenQM* (either a directory or a hashed file), you can use the AS keyword instead of TO:

```
AS filename id {NO.QUERY | APPENDING}
```

If the output file exists, *OpenQM* will prompt you for permission to overwrite it. Answering ‘N’ to this prompt will terminate the query.

If you use the NO.QUERY option, then *OpenQM* will overwrite any existing file without prompting.

Use of the APPENDING option will result in the query output being appended to any existing file without any prompting.

If you omit the TO or AS clause, output will go to the screen:

```
SORT XRATES WITH YEAR EQ "2012" AND WITH DOM EQ "20" DATE USD GBP AUD JPY EUR CSV ID.SUP
Date,US Dollar,UK Pound,Aus Dollar,Jap Yen,Euro
20 JAN 2012,0.8026,0.5184,0.7706,61.87,0.6191
20 FEB 2012,0.8396,0.5297,0.7792,66.88,0.6369
20 MAR 2012,0.8262,0.5199,0.7787,68.91,0.6241
20 APR 2012,0.8147,0.5073,0.7877,66.55,0.6199
20 JUN 2012,0.7962,0.5064,0.7823,62.94,0.6278
20 JUL 2012,0.8033,0.5111,0.7707,63.23,0.6544
20 AUG 2012,0.8066,0.5140,0.7740,64.14,0.6538
20 SEP 2012,0.8297,0.5114,0.7921,65.03,0.6358
20 NOV 2012,0.8190,0.5149,0.7872,66.65,0.6402
20 DEC 2012,0.8339,0.5132,0.7959,70.28,0.6303
10 record(s) listed
```

The DELIMITER keyword also generates a delimited report. It’s format is:

```
DELIMITER “string”
```

The output destination is controlled by the TO or AS keywords as for the CSV reports.

```
SORT XRATES WITH YEAR EQ "2012" AND WITH DOM EQ "20" DATE USD GBP AUD JPY EUR DELIMITER "<+>"
ID.SUP
Date<+>US Dollar<+>UK Pound<+>Aus Dollar<+>Jap Yen<+>Euro
20 JAN 2012<+>0.8026<+>0.5184<+>0.7706<+>61.87<+>0.6191
20 FEB 2012<+>0.8396<+>0.5297<+>0.7792<+>66.88<+>0.6369
20 MAR 2012<+>0.8262<+>0.5199<+>0.7787<+>68.91<+>0.6241
20 APR 2012<+>0.8147<+>0.5073<+>0.7877<+>66.55<+>0.6199
20 JUN 2012<+>0.7962<+>0.5064<+>0.7823<+>62.94<+>0.6278
20 JUL 2012<+>0.8033<+>0.5111<+>0.7707<+>63.23<+>0.6544
20 AUG 2012<+>0.8066<+>0.5140<+>0.7740<+>64.14<+>0.6538
20 SEP 2012<+>0.8297<+>0.5114<+>0.7921<+>65.03<+>0.6358
20 NOV 2012<+>0.8190<+>0.5149<+>0.7872<+>66.65<+>0.6402
20 DEC 2012<+>0.8339<+>0.5132<+>0.7959<+>70.28<+>0.6303
10 record(s) listed
```

6.9 Reformatting Data Files

There are frequently times when we want to use summary data for some purpose, but all the data we are collecting is at a detail level. While we could translate between files, the level difference is going to cause problems somewhere along the way. What we need is a way to reformat our detailed data into a summary level file.

QMQuery provides a way to do this. Essentially, we replace our standard LIST and SORT verbs with their reformatting equivalents REFORMAT and SREFORMAT. And that is almost all there is to it.

The first step in carrying out a reformat (or file restructure) of this nature is to create the new file to contain the reformatted data.

```
CREATE.FILE XRATES.MONTH
```

and as a first step to provide a dictionary, we will copy the XRATES dictionary across:

```
COPY FROM DICT XRATES TO DICT XRATES.MONTH ALL
Record '@ID' already exists - '@ID' not copied.
35 record(s) copied.
```

The ID of our new file has to be unique for the month. We also want an ID that we can calculate easily so that we can use it in lookups. On this basis, we'll use a concatenation of the year and month number e.g. 201307 for July 2013. We need to generate this number within our source file (XRATES), so create an I-type named YYYYMM in the dictionary of XRATES with the following expression:

```
    YEAR * 100 + MTHNO
```

The next step is to create a *QMQuery* statement using LIST or SORT which displays the data *as you want it to appear in the new data file.* Quite literally, the REFORMAT verbs take the output from the *QMQuery* statement and write it to the file you specify. Our first attempt at this command is shown below.

```
SORT XRATES BREAK.ON YYYYMM "'V'" AVG USD NO.NULLS AVG GBP NO.NULLS AVG AUD NO.NULLS AVG JPY
NO.NULLS AVG EUR NO.NULLS DET.SUP
YYYYMM.   US Dollar   UK Pound   Aus Dollar   Jap Yen   Euro...
201001       0.7277     0.4501       0.7959     66.38    0.5092
201002       0.6974     0.4455       0.7869     62.93    0.5094
201003       0.7033     0.4670       0.7713     63.66    0.5178
201004       0.7124     0.4644       0.7685     66.52    0.5304
201005       0.6992     0.4761       0.8019     64.36    0.5557
201006       0.6928     0.4696       0.8105     62.96    0.5665
```

Essentially, this statement averages the values for each month, and suppresses the daily data so that it displays the only the monthly averages. Note the use of the NO.NULLS modifiers to ensure we get valid averages.

That looks OK, so now we'll replace the SORT verb with SREFORMAT and see what happens:

```
SREFORMAT XRATES BREAK.ON YYYYMM "'V'" AVG USD NO.NULLS AVG GBP NO.NULLS AVG AUD NO.NULLS AVG JPY
NO.NULLS AVG EUR NO.NULLS DET.SUP

Output file: XRATES.MONTH
873 record(s) reformated, 0 record(s) overwritten
```

So, we were prompted for a file name, to which we responded: XRATES.MONTH. Alternatively, we could have specified an output file in the query command itself with:

```
        TO filename
```

Then we received a message indicating that the reformat had completed. Let's see what *QMQuery* reports from the file:

```
SORT XRATES.MONTH USD AUD GBP JPY EUR
XRATES.MONTH    US Dollar    Aus Dollar    UK Pound    Jap Yen    Euro...
201001             0.0001        0.0001      0.0000       0.66     0.0001
201002             0.0001        0.0001      0.0000       0.63     0.0001
201003             0.0001        0.0001      0.0000       0.64     0.0001
201004             0.0001        0.0001      0.0000       0.67     0.0001
201005             0.0001        0.0001      0.0000       0.64     0.0001
```

Well, that is clearly incorrect. Let's look in one of the items to see what actual data is there.

```
CT XRATES.MONTH 201005
XRATES.MONTH 201005
1: 0.6992
2: 0.4761
3: 0.8019
4: 64.36
5: 0.5557
```

From this listing, we can see that the REFORMAT command has included all the decimal points from the display output in the data. This is contrary to normal practice in the multi-value world where the data is stored without commas and decimal places. Then, when we listed the item using *QMQuery*, the dictionary items applied further output conversions to the data – even though it was already in external format.

Let's clear the data in our file before we try to fix the problem:

```
        CLEAR.FILE DATA XRATES.MONTH
```

What can we do about this? There are two options here – either create new dictionary items without conversion codes for use with REFORMAT, or specify an alternative conversion in the query statement for each field.

Alternate conversions are specified in a similar manner to the alternative column headings mentioned in Section 6.7.2. This requires the display specification of the field to be in the following format:

```
        field CONV "conv-spec"
```

The conversion specification we want for each of the output fields is simply "MR0", meaning a right-justified decimal number. Therefore, our REFORMAT command becomes:

```
SREFORMAT XRATES BREAK.ON YYYYMM "'V'" AVG USD NO.NULLS CONV "MR" AVG GBP NO.NULLS CONV "MR" AVG
AUD NO.NULLS CONV "MR" AVG JPY NO.NULLS CONV "MR" AVG EUR NO.NULLS CONV "MR" DET.SUP
```

and the new data in the FX.MONTHLY file looks like:

```
CT XRATES.MONTH 201005
XRATES.MONTH 201005
1: 6992
2: 4761
3: 8019
4: 6436
5: 5557
```

Checking the output from this data through a normal *QMQuery* statement:

```
SORT XRATES.MONTH USD AUD GBP JPY EUR
XRATES.MONTH    US Dollar   Aus Dollar   UK Pound   Jap Yen   Euro...
201001             0.7277       0.7959     0.4501     66.38    0.5092
201002             0.6974       0.7869     0.4455     62.93    0.5094
201003             0.7033       0.7713     0.4670     63.66    0.5178
201004             0.7124       0.7685     0.4644     66.52    0.5304
201005             0.6992       0.8019     0.4761     64.36    0.5557
```

This matches the output from our earlier query on the source data file. At this stage, the FX.MONTHLY file is getting near the stage when it can be used in conjunction with the XRATES file to report on foreign exchange volumes and rates.

The key thing to remember when using the REFORMAT and SREFORMAT verbs is that they write to the destination file *almost exactly as QMQuery would display the data on the screen.* Therefore, we need the data to display in the internal format that you want written to the new file[28].

There are two other things to be aware of. Firstly, REFORMAT will overwrite any existing records in the destination file, so you need to ensure that your new data structure is captured correctly, and that you do not accidentally have later data overwriting earlier data.

Secondly, REFORMAT can write back to the source file. This could happen accidentally if you specify the wrong filename in the command or in response to the filename prompt. Given that this could overwrite data, you should be careful to specify the correct file name.

Alternatively, you could use this feature to restructure the data in an existing file. However, because of the likelihood of data being overwritten in this instance, you need to ensure that your statement is exactly correct before running it. Copying the data to a backup file first would be a good idea.

6.10 Working with Multi-Values

6.10.1 Multi-valued files, dictionary items and associations

Supposedly, this is a book on multi-value databases. But so far, we haven't used any multi-values in our examples.

This actually underlines the flexibility of multi-value databases – you aren't forced into using particular features. In fact, multi-value databases work quite well using the same file structures as relational databases.

However, we need to cover how to query multi-values, and consider where multi-values should be used.

The classic example of multi-value usage is for an invoice. This idea was covered in Section 1.1.1, where a the layout of an invoices file was shown. This is reproduced below:

28 Older versions of *OpenQM* also wrote items to the destination file for GRAND.TOTAL and the item count. Therefore, you needed to use the NO.GRAND.TOTAL and COUNT.SUP modifiers. Space characters in the ID could also be written to file, making the item-id different to what you would see.

Invoice number	*Date*	*Customer number*	*Product ID*	*Quantity*	*Price*
12345	24 Apr 2007	9854	9854	2	15.00
12346	24 Apr 2007	6234	6234 4921	1 1	32.50 23.90
12347	25 Apr 2007	4921	5651 5694 6234	3 2 5	12.50 3.50 32.50

This table will convert easily to an *OpenQM* file. Before we do that, however, we will examine this table in more detail.

The invoice number will form the item-id of the file. This makes the (usually reasonable) assumption that the invoice number will be unique.

The date of the invoice is shown here in external format. However, this will be stored in internal format as a sequential date number, with output conversion used for reporting purposes.

The customer details are referenced simply via the customer number. Lookups can occur to get the customer's name and address from the CUSTOMERS file as appropriate.

These first three fields are all single valued. That is, they each contain only one value. (Of course, the item-id has to be single valued).

The next three fields are all multi-valued. That is, they can contain more than one value. When we create the dictionary items for these fields, we will need to specify that these fields are multi-valued so that *OpenQM* knows how to process them.

The next point to note is that these three multi-valued are associated. That is to say, each value in the Product ID field is associated with (or related to) matching values in the Quantity and Price fields. This means that the order of the values in each field is important.

OpenQM needs to know about these associations so that it can properly report on these fields. This is what the ASSOC prompt means when you are creating dictionary items using the MODIFY editor.

Associations are created by entering a phrase item into the dictionary that details the names of the fields in the association. Phrases were introduced in Section 5.7.1 as a shortcut means of providing display defaults.

We need to decide the names of our dictionary items and of the association before we start creating these items – because they reference each other. While we can always go back and edit these items later, it helps to define things first.

Start by creating an invoices file:

```
CREATE.FILE INVOICES
```

Now, let's create an association named INVDET which references the fields PRODID, QUANTITY, and PRICE. Once you've created this, it should like like this:

```
CT DICT INVOICES INVDET
DICT INVOICES INVDET
1: PH
2: PRODID QUANTITY PRICE
```

Now, create the dictionary items using the following definitions:

```
ID          Type  Loc  Conv    Name                Format  S/M   Assoc
DATE        D     1    D4/     Date                10R     S
CUSTID      D     2    MR      Customer ID         7R      S
PRODID      D     3    MR      Product ID          7R      M     INVDET
QUANTITY    D     4    MR,Z    Quantity            7R      M     INVDET
PRICE       D     5    MR2,Z   Price               9R      M     INVDET
```

Note the PRODID, QUANTITY, and PRICE dictionaries are all defined as multi-valued in the S/M field, and specify the INVDET association in the ASSOC field.

6.10.2 Entering multi-valued data

Once this has been done, we can use the MODIFY editor to enter some data. We will enter the same data as is shown in the table above. Type in:

```
MODIFY INVOICES
```

and enter 12345 at the item-id prompt. You will then see a screen display similar to that shown below:

```
MODIFY INVOICES

INVOICES (? for list): 12345
  1: Date       =
  2: Customer ID=
  3> Product ID, Quantity, Price
```

Note that the third line displays all the related data together. This is using the information in the INVDET association to group these fields together.

Now, enter in the data for the item.

The date was 24 Apr 2007. This can be entered in any appropriate date format and *OpenQM* will input convert this for storage. It will also output convert the result into the format we specified in the dictionary (D4/). Therefore, entering ‘24 apr 2007’ will result in a display of 24/04/2007 (or 04/24/2007 if you are using American date formats). We could equally have entered the date as 24/4/07 or ‘apr 24 2007’ and the final display would have been the same.

The Customer ID is straightforward in this case as we are not checking that it exists elsewhere. In a proper application, any Customer ID entered here would be checked against the CUSTOMERS file to ensure that this customer exists. The MODIFY editor can carry out such a check for you – see the documentation for details – but normally you will have a custom built data entry application to this for you.

Once you have entered the Customer ID, the screen changes to look like:

```
MODIFY INVOICES

INVOICES (? for list): 12345
     Product ID Quantity Price....
```

and the prompt at the bottom of the screen says Product ID:

Enter the Product ID from the table above, followed by the Quantity and Price as you are prompted. The data is displayed under the appropriate field heading after each entry. Once you have completed entry of the related data, the prompt returns to Product ID again, so that you can enter another line of related data.

```
MODIFY INVOICES

INVOICES (? for list): 12345
     Product ID Quantity Price....
  1: 9854       2        15.00
```

As we don't have any more data to enter for this invoice, simply press Enter at the Product ID prompt, then enter again at the `Modify item` prompt (see below), and FI at the `Action` prompt to file the item.

The Modify item prompt looked like:

```
Modify item(n/In/Dn/E/?):
```

n refers to the line number of one of the sets of related data that has been entered.

Entering a line number lets you modify the related data in that line.

Entering In lets you insert a set of related data at line n.

Entering Dn deletes the set of related data at line n.

Entering E lets you continue entering new related at the end of the existing data sets.

Entering ? shows a short set of prompts.

Now enter the next two invoices.

You can see how *OpenQM* stores the data by displaying the multi-valued item on the screen:

```
CT INVOICES 12347
INVOICES 12347
1: 14360
2: 4921
3: 5651²5694²6234
4: 3²2²5
5: 1250²350²3250
```

Note how fields 3, 4, and 5 each have multiple values, delimited by a value mark (²).

6.10.3 Querying multi-valued data

Display your entered data as follows:

```
SORT INVOICES DATE CUSTID PRODID QUANTITY PRICE
INVOICES..   Date......   Customer   Product   Quantity   Price....
                          ID......   ID.....
12345        24/04/2007       9854      9854          2       15.00
12346        24/04/2007       6234      6234          1       32.50
                                        4921          1       23.90
12347        25/04/2007       4921      5651          3       12.50
                                        5694          2        3.50
                                        6234          5       32.50

3 record(s) listed
```

This shows each invoice with its associated multiple lines of detail data. It is relatively simple from here to build an `I-type` dictionary item to calculate the extended value of each detail line of the invoice, and then to use *QMQuery* to calculate the total value of each invoice. The `I-type` dictionary item looks like:

```
I
QUANTITY * PRICE
MR2,
Extended²Value
9R
M
```

And the output of this looks like:

```
SORT INVOICES BREAK.ON @ID TOTAL EXTVALUE GRAND.TOTAL "Total'U'" DET.SUP
INVOICES..   Extended.
             Value....
12345            30.00
12346            56.40
12347           207.00
             =========
Total           293.40

3 record(s) listed
```

Now, what about getting the value of the invoices broken into the value of each product? This is where we use the BY.EXP keyword (or its descending sort counterpart BY.EXP.DSND).

The BY.EXP keyword “explodes” multi-valued data to break it out of its association with a particular invoice. You can then group the exploded data into Product ID order (for example).

```
SORT INVOICES BY.EXP PRODID BREAK.ON PRODID ENUM @ID FMT "7R" TOTAL QUANTITY TOTAL EXTVALUE
GRAND.TOTAL "Totals'U'" DET.SUP
Product ID   INVOICES   Quantity   Extended.
                                   Value....
      4921          1          1       56.40
      5651          1          3      207.00
      5694          1          2      207.00
      6234          2          6      263.40
      9854          1          2       30.00
             ========   ========   =========
Totals              6         14      763.80

3 record(s), 6 value(s) listed
```

The above query does the following:

- Explodes the data and groups by Product ID
- Counts the number of invoices that each product appears on
- Counts the quantity of each product sold
- Calculates the value of each product sold, and the total value of products sold.

The only new thing in this query is the use of ENUM (or ENUMERATE) to count the number of invoices. Note that this has literally counted the invoice item-id’s as we can be certain that this is a unique value, and that we have used an FMT modifier to force a right-justified display.

Note also that the item count at the end of the query has been expanded to show the count of values displayed by the query.

Now, let’s consider selection of multi-valued data. Normal selection of data used the WITH keyword. Let’s try that:

```
SORT INVOICES WITH PRODID EQ "6234" DATE CUSTID PRODID TOTAL QUANTITY PRICE TOTAL EXTVALUE
INVOICES..   Date......   Customer ID   Product ID   Quantity   Price....   Extended.
                                                                            Value....
12346        24/04/2007          6234         6234          1       32.50       32.50
                                              4921          1       23.90       23.90
12347        25/04/2007          4921         5651          3       12.50       37.50
                                              5694          2        3.50        7.00
                                              6234          5       32.50      162.50

                                                           12                  263.40

2 record(s) listed
```

This has correctly selected the two records that contain the Product ID of 6234, but the query has returned all invoice detail lines – whether or not they contain the specified Product ID.

This isn't necessarily wrong – you may want to know what else your customers are buying if they have bought Product ID 6234.

To return only the matching values (as distinct from the matching records), we need to use the WHEN keyword:

```
SORT INVOICES WHEN PRODID EQ "6234" DATE CUSTID PRODID TOTAL QUANTITY PRICE
INVOICES..   Date......   Customer ID   Product ID   Quantity   Price....
12346        24/04/2007          6234         6234          1      32.50
12347        25/04/2007          4921         6234          5      32.50

                                                            6

2 record(s) listed
```

This has returned only the selected Product ID's, and their matching multi-valued data (quantity and price).

Now, let's add the extended price to this listing:

```
SORT INVOICES WHEN PRODID EQ "6234" DATE CUSTID PRODID TOTAL QUANTITY PRICE TOTAL EXTVALUE
INVOICES..   Date......   Customer ID   Product ID   Quantity   Price....   Extended.
                                                                            Value....
12346        24/04/2007          6234         6234          1      32.50        32.50
                                                                                23.90
12347        25/04/2007          4921         6234          5      32.50        37.50
                                                                                 7.00
                                                                               162.50

                                                            6                  263.40

2 record(s) listed
```

Well, that isn't right. We are showing the totals for the invoice detail lines that weren't selected. The problem here is that our EXTVALUE dictionary item isn't part of the association. Use MODIFY and add the INVDET association to EXTVALUE and try the report again:

```
SORT INVOICES WHEN PRODID EQ "6234" DATE CUSTID PRODID TOTAL QUANTITY PRICE TOTAL EXTVALUE
INVOICES..   Date......   Customer ID   Product ID   Quantity   Price....   Extended.
                                                                            Value....
12346        24/04/2007          6234         6234          1      32.50        32.50
12347        25/04/2007          4921         6234          5      32.50       162.50

                                                            6                  195.00

2 record(s) listed
```

That's better. We could have achieved the same thing by explicitly making EXTVALUE part of the INVDET association in the *QMQuery* statement. To check this, remove the association from the EXTVALUE dictionary item and try the following:

```
SORT INVOICES WHEN PRODID EQ "6234" DATE CUSTID PRODID TOTAL QUANTITY PRICE TOTAL EXTVALUE ASSOC
INVDET
INVOICES..   Date......   Customer ID   Product ID   Quantity   Price....   Extended.
                                                                            Value....
12346        24/04/2007          6234         6234          1      32.50        32.50
12347        25/04/2007          4921         6234          5      32.50       162.50

                                                            6                  195.00

2 record(s) listed
```

Or, we could have associated the EXTVALUE field with one of the other multi-value fields. This implicitly makes it part of any association that field belongs to:

```
SORT INVOICES WHEN PRODID EQ "6234" DATE CUSTID PRODID TOTAL QUANTITY PRICE TOTAL EXTVALUE
ASSOC.WITH QUANTITY
INVOICES..   Date......   Customer ID   Product ID   Quantity   Price....   Extended.
                                                                            Value....
12346        24/04/2007          6234         6234          1       32.50       32.50
12347        25/04/2007          4921         6234          5       32.50      162.50

                                                            6                  195.00

2 record(s) listed
```

6.11 Outer Joins

In an SQL view of the world, the above invoices example would use two tables – a header table, and a detail table. To show the overall invoice, the header record would be selected from the header table, and then the detail lines for that invoice would be selected using an outer join on the invoice number.

In the multi-value world, it is more common to select from the detail records back to the header records. However, *OpenQM* does provide the OUTERJOIN function to go the other way.

In this example, we are going to start with the XRATES.MONTH file to display the average exchange rate for the month, and then display each day in the month from the XRATES file.

We need to do two things to make this work:

- We need to link the XRATES.MONTH file to the XRATES file using the OUTERJOIN function; and
- We need to create and alternate key index on XRATES.

We need the alternate key index because the OUTERJOIN function uses that index. We'll create that first.

When we created the XRATES.MONTH file, we created a dictionary item named YYYYMM in XRATES to form the @ID for XRATES.MONTH. We now want to index XRATES using that dictionary:

```
MAKE.INDEX XRATES YYYYMM
Added index for YYYYMM
Building index 'XRATES YYYYMM'...
873 records processed
Populating index...
```

Now, we create the link between the files. That looks like:

```
CT DICT XRATES.MONTH D
DICT XRATES.MONTH D
1: L
2: OUTERJOIN('XRATES','YYYYMM',@ID)
3: XRATES
```

Our link says: Look up the value @ID in the YYYYMM index in the XRATES file, and return the list of ID's. So, let's give it a try:

```
SORT XRATES.MONTH USD D%DATE D%USD
XRATES.MONTH   US Dollar   Date........   US Dollar
201001            0.7277    05 JAN 2010      0.7344
                            06 JAN 2010      0.7343
                            07 JAN 2010      0.7378
                            08 JAN 2010      0.7325
                            11 JAN 2010      0.7395
                            12 JAN 2010      0.7422
                            13 JAN 2010      0.7385
                            14 JAN 2010      0.7398
                            15 JAN 2010      0.7427
```

Now, the DATE and USD dictionary items in the XRATES file are actually single-valued, but they have displayed here as multi-valued. If they hadn't displayed properly, we could have tried using the MULTI.VALUE (or MULTIVALUED) qualifier:

```
LIST XRATES.MONTH USD D%DATE MULTIVALUED D%USD MULTIVALUED
XRATES.MONTH   US Dollar   Date........   US Dollar
201001            0.7277    05 JAN 2010      0.7344
                            06 JAN 2010      0.7343
                            07 JAN 2010      0.7378
                            08 JAN 2010      0.7325
                            11 JAN 2010      0.7395
                            12 JAN 2010      0.7422
                            13 JAN 2010      0.7385
                            14 JAN 2010      0.7398
                            15 JAN 2010      0.7427
```

Can we make the monthly average figure repeat down the list of daily values? No. Or at least, not easily.

If we wanted to use TRANS to look up the daily exchange rates rather than using a dictionary link, the dictionary to look up the US Dollar exchange rate would look like:

```
I
TRANS('XRATES', OUTERJOIN('XRATES', 'YYYYMM', @ID), USD, 'X')
MR44,
US Dollar²Daily
7R
M
```

Note that we have defined this as a multi-valued item.

6.12 Cross-Tabs (or Pivot Tables)

A cross-tab is where the data is arranged into a two-way (or more) classification. For example, the export data may be arranged to display countries in rows and years in columns. The intersection cells would show exports to that country for that year.

There is no QMQuery command to create this type of report …. but we can create a set of dictionary items to simulate this type of report. The 2009 dictionary item (named FOB.2009.M) looks like:

```
I
IF YEAR EQ 2009 THEN FOB ELSE ''
MR16,
2009 $m
9R
S
```

Create equivalent items for 2010, 2011, 2012, and 2013.

Now we can run queries like:

```
SORT TEX.QCH WITH HS EQ "02" AND WITH C%SUBREG EQ "013""021" BY C%SUBREG BY CTRY BREAK.SUP C
%SUBREG.NAME BREAK.ON C%SHORTNAME TOTAL FOB.2009.M TOTAL FOB.2010.M TOTAL FOB.2011.M TOTAL
FOB.2012.M TOTAL FOB.2013.M NO.GRAND.TOTAL DET.SUP
Country............   2009 $m..   2010 $m..   2011 $m..   2012 $m..   2013 $m..
Belize                      0.0         0.3         0.0         0.0         0.0
Costa Rica                  0.0         0.0         0.0         0.0         0.0
Guatemala                   0.0         0.0         0.0         0.0         0.0
Honduras                    0.0         0.0         0.0         0.0         0.0
Mexico                     39.6        42.4        36.6        19.7         7.4
Nicaragua                   0.0         0.0         0.0         0.0         0.0
Panama                      0.0         0.0         0.0         0.0         0.0
El Salvador                 0.0         0.0         0.0         0.0         0.0
                      ---------   ---------   ---------   ---------   ---------
                           39.6        42.7        36.6        19.7         7.4
Bermuda                     2.7         2.4         3.4         2.9         0.8
Canada                    239.5       208.6       248.8       189.8        46.9
Greenland                   0.0         0.0         0.0         0.0         0.0
St Pierre                   0.0         0.0         0.0         0.0         0.0
United States             971.6     1,017.9     1,122.9     1,180.0       420.0
                      ---------   ---------   ---------   ---------   ---------
                        1,213.8     1,228.9     1,375.0     1,372.8       467.7

221 record(s) listed
```

Now, to check that these totals are correct, we can run queries such as:

```
SORT TEX.QCH WITH CTRY EQ "CA" AND WITH HS EQ "02" BREAK.ON YEAR TOTAL FOB.M DET.SUP
Year   ...FOB $m
2009       239.5
2010       208.6
2011       248.8
2012       189.8
2013        46.9

           933.6

17 record(s) listed
```

This shows that the Canadian totals match those shown in our cross-tab report. You can repeat this query for the other countries to satisfy yourself that those totals are correct too.

This method of creating cross-tabs works well as long as the number of categories isn't too great. Clearly, it would be tedious to create a dictionary item for every country (240+ countries) or HS code (99 Chapter totals).

Years were chosen for the above report because there aren't too many years in the table. But most trade statistics in New Zealand are reported on June or September ending years. Therefore, to show the above table on a "trade year" basis, we would need to create a new set of "year" dictionaries for each type of year – i.e. one set for March years; another set for June years; and a third set for September years; in addition to the calendar year dictionary items we have already created.

This proliferation of dictionary items make this method of creating cross-tab reports less than ideal. However, short of writing a program to generate the cross-tab for you, this is the method that we have.

7 Design Issues

7.1 Meaningful Data in Item-ids

There was a brief discussion in Section 4.2 on the use of meaningful data in item-id's. The essential elements outlined there were that the meaningful data must be unique, and must not change through time.

The requirement for uniqueness is obvious enough – this is identifier that will be used to find the record in the database. It has to be unique for this to happen. Relational databases have the same requirement for their primary keys.

The reason that we don't want this value to change is two-fold:

- To change a primary key, we must first read the existing record, and then write it back to the database with the new primary key. Finally, we need to delete the record with the original primary key. The delete operation makes the whole process messy.
- More critically, when databases store lists of related records in other files, they store the primary key of the related data. If we change the primary key, then we need to change that value in any lists of related data which include that key. In other words, we may need to update many records in multiple data files, simply because we've changed a key value.

Given that changing keys is awkward, why do we often use meaningful data?

One reason that we would use meaningful data is that it helps us as humans look up data in the database. If we wanted to see New Zealand's meat exports to China in the third quarter of 2012, we could run a query like:

```
LIST TEX.QCH '20123*CN*02' YEAR QTR C%SHORTNAME FOB ID.SUP
```

We could deduce that was the correct ID to specify because the data was meaningful to us. Of course, we could have run a similar query even if we had used sequential numbers as the item-id's:

```
LIST TEX.QCH WITH CTRY EQ "CN" AND WITH HS EQ "02" AND WITH YEAR EQ "2012" AND WITH QTR EQ "3"
YEAR QTR C%SHORTNAME FOB ID.SUP NO.INDEX
```

The above statement may even be easier for someone to create than by specifying a slightly arcane item-id, even if it is a little more verbose. So, this is not a compelling argument for meaningful data in the ID.

On the other hand, the speed at which those two queries execute is notably different. The first query – where we specified the ID of the item – returned the answer almost instantly (8ms). The second query – where we had to search the file – took around 6.6 seconds on my notebook.

Of course, we can index the TEX.QCH file to reduce the search times. We created them earlier, but if they don't exist the we can create them now:

```
MAKE.INDEX TEX.QCH CTRY HS
Added index for CTRY
Added index for HS
Building index 'TEX.QCH CTRY'...
411502 records processed
Populating index...
Building index 'TEX.QCH HS'...
411502 records processed
Populating index...
```

Now, if we re-run the second query (without the NO.INDEX modifier), the answer is returned in less than 0.1 seconds (63ms). That is much quicker than was achieved by searching the entire file, but still not as quick as going directly to the relevant item.

So, performance is much better if we can provide the specific item-id(s) we want to retrieve. And we have a much better chance of providing specific item-ids if the data is meaningful.

Let's also add an observation that for data to be meaningful to us, it must be relatively small, or made up of groups of small bits of data. The ID for TEX.QCH is made up of three pieces of data which, when taken together, uniquely identify the data element. Each of those three groups is quite recognisable to us:

- The first group is a year and quarter identifier. We can quickly change this identifier to any other year and quarter.
- The second group is a country identifier. While there are over 250 countries identified in the NCY.C file, we are usually only interested in a small subset of these, and have no problems remembering the identifier for those countries. In a program, it is not difficult to offer the user the choice of all countries with appropriate lookups to speed the selection.
- The final group is an HS code. We've only used the Chapter codes here which are two digit codes. Once again, we are usually only interested in a small proportion of the available codes, and for general users, selection tools can be made available to simplify the choice.

Overall then, it was easy to generate the specific ID that we wanted by assembling the different identifiers and splicing them together with asterisks as delimiters.

Now, let's consider why we used a delimited ID rather than simply concatenating the three elements together. That would have given us an ID of 20123CN02 – which is still unique, and we can still extract the various components from the ID. However, delimiters do provide a degree of visual separation of the elements; and delimiters allow the elements to be of variable length.

Are those elements likely to change length?

Well, once we get to the year 10,000, the year and quarter identifier will gain another digit. Perhaps you think that is far enough away that we need not worry about it. Perhaps you are right, but that is also how systems come to embed rigidities in their structure – such as the Y2K issue of 2 digit years always being assumed to be in the 20th century (i.e. 19xx) because the turn of the century was too far away (and cost of storing 2 extra digits too high) to worry about.

Let's briefly consider the dictionary items used to break the YYYYQ element into its constituent YEAR and QTR components. YEAR used the expression YYYYQ[1,4] while QTR used the expression YYYYQ[5,1]. These both work while YYYYQ is only 5 digits long, but both will fail once we arrive at year 10,000. Yet it isn't difficult to structure them so that they will always work. YEAR could use YYYYQ[1, LEN(YYYYQ) - 1], while QTR could use YYYYQ[1].

Will the 2 character country codes change to 3 characters? Well, the whole point of the ISO 3166 alpha-2 standard is that country codes are 2 characters in length. Perhaps if the number of countries increases, the ISO 3166 alpha-3 standard will be used more.

A more worrying problem for key design is that countries split and change their names. Accordingly, their identifiers may change too.

For example, 'BU' used to be the code for Burma, but that has now changed to 'MM' (Myanmar). 'CS' used to be the code for Czechoslovakia, but the Czech Republic now uses 'CZ' while 'CS' was transferred to Serbia and Montenegro. Serbia has since become 'RS' while Montenegro became 'ME'.

The above problems are a good case for NOT using the key structure used in TEX.QCH.

Why? Because we potentially need to re-key all affected items in TEX.QCH when a country identifier changes. Initially, you may choose to continue using the old code, or use the new code for new entries while keeping the existing entries with the old code. However, if the old country code is re-assigned, then you are forced to change the keys of your existing data.

It is important to recognise that you are always going to have issues with country identifiers – whatever structure you choose. But if you choose to include a country identifier in the ID (primary key) of the file, then you are creating structural issues within the database design (because table lookups are achieved by looking up records via their item-id).

What about the HS Chapter identifiers? Are they a good choice for an ID? Well, at the Chapter level, the data is highly aggregated. These chapters are unlikely to change.

Of course, HS Codes come in a range of sizes – namely 2, 4, 6 and 10 digits long – with increased detail with each increase in digits. For example:

02	Meat and edible meat offal
0201	Meat of bovine animals; fresh or chilled
0201.30	Meat; of bovine animals, boneless cuts, fresh or chilled
0201.30.00.01	Meat; of bovine animals, beef cuts according to the NZ Meat Producers' Board definition, of cow, steer and heifer, boneless, fresh or chilled

The codes are standardised up to the 6-digit level, but become country specific at the 10-digit level.

Now, the TEX.QCH file will handle all the above codes without modification. The HS dictionary item will extract whatever HS code we enter in the ID because it is simply extracting the third field delimited by asterisk characters. Therefore, if we added 6-digit HS codes to the file, we could use a query like:

```
LIST TEX.QCH WITH YEAR EQ “2012” AND WITH QTR EQ “3” AND WITH CTRY EQ “CN” AND WITH HS EQ
“0201.30” YEAR QTR C%SHORTNAME H%SHORTDESC QTY FOB ID.SUP
```

As you can see, this is just a simple extension of the earlier query where we extracted the HS Chapter total. Of course, this level of detail would add significantly to the data the file would contain too.

But, detailed HS codes change through time, even if the meaning of the Chapter totals do not. Therefore, if we wanted to use this file to store more detailed HS codes, we could potentially have the problem of changing key values.

Are there other reasons to use meaningful data in the item-id?

We used item-id’s to look up related data both in external files and in the current file for different time periods. We could do this easily because we could derive the related item’s ID from our current item-id. Let’s consider this in greater depth.

We looked up data in the interest rates file to be displayed alongside data from our primary file of exchange rates. We did this in two ways. Firstly, in section 6.3.1, we wrote a dictionary item that used the TRANS function to look up data in the remote file. Secondly, in section 6.3.2, we created a link between the exchange rates file and the interest rates file so that we could use any of the dictionary items in the interest rate file in our query on exchange rates.

Both of these methods relied on the fact that the item-id in the interest rates file was the same as that used in the exchange rates file. [It is also important to recognise that the item-id’s don’t have to be same, but there must be some data in the primary file that can be transformed to the item-id in the secondary file to enable the lookup]. By using meaningful data that was consistent between the two files, we provided an easy way to look up data in the remote file.

7.2 Sequential Item ids

In section 6.4, we looked up prior period data in both the XRATES and the TEX.QCH files. Once again, this was done by calculating the ID of the item we needed to look up from some existing data.

Could this have been done using sequential numbers? Well, yes …. but not in the same way. We should not expect to be able to *calculate* a related item-id if the files use sequential numbers for their item-ids. Instead, we would have to make use of indices on the related file to find the record(s) that match our relationship criteria.

To see how this might work, we’ll transform the data into TEX.QCH into a file using sequential ID’s. First, we’ll create a file to hold the transformed data:

```
CREATE.FILE TEX.QCH.N MINIMUM.MODULUS 5120
Created DICT part as TEX.QCH.NA.DIC
Created DATA part as TEX.QCH.NA
Added default '@ID' record to dictionary
```

Now, enter the following program:

```
WED BP TEX.QCH.N.FILL

PROGRAM TEX.QCH.N.FILL
******************************************************************************

OPEN 'TEX.QCH' TO tex.qch ELSE STOP 201, 'TEX.QCH'
OPEN 'TEX.QCH.N' TO tex.qch.n ELSE STOP 201, 'TEX.QCH.N'

CRT 'Filling TEX.QCH.N ... ':

SSELECT tex.qch
id = 0
LOOP
  READNEXT qch ELSE EXIT

  READ rec FROM tex.qch, qch ELSE rec = ''
  q = FIELD(qch, '*', 1)
  c = FIELD(qch, '*', 2)
  h = FIELD(qch, '*', 3)
  qty = rec<1>
  fob = rec<2>

  id += 1
  nrec = ''
  nrec<1> = q
  nrec<2> = c
  nrec<3> = h
  nrec<4> = qty
  nrec<5> = fob
  READU dummy FROM tex.qch.n, id ELSE NULL
  WRITE nrec ON tex.qch.n, id

  IF (MOD(id, 100) EQ 0) THEN CRT @(23):id 'R,#11':
REPEAT

CRT
CRT 'Complete'
STOP

END
```

Once you have entered it, save it, exit from WED, and compile it:

```
BASIC BP TEX.QCH.N.FILL
Compiling BP TEX.QCH.N.FILL
***
0 error(s)
Compiled 1 program(s) with no errors
```

Essentially, this program sorts the existing data file into ID order, then loops through the data items. It reads each record, makes a new record from the data, and writes that new record to the TEX.QCH.N file with a sequential number as the ID.

The program isn't catalogued, so we need to invoke it using the RUN command:

```
RUN BP TEX.QCH.N.FILL
Filling TEX.QCH.N ...      411,500
Complete
```

Now, enter the following dictionary items:

```
SORT DICT TEX.QCH.N
@ID.........   TYPE   LOC..........   CONV..   NAME........   FORMAT   S/M   ASSOC...
@ID            D      0                        TEX.QCH.N      10L      S
YYYYQ          D      1                        YYYYQ          5R       S
CTRY           D      2                        Ctry           2L       S
HS             D      3                        HS Code        2R       S
QTY            D      4               MR,      'R'Quantity    11R      S
FOB            D      5               MR,      'R'FOB Value   15R      S
YEAR           I      YYYYQ[1,4]               Year           4R       S
QTR            I      YYYYQ[5,1]               Qtr            1L       S
```

These just define the fields in the file.

Copy the `C` and `H` link items into the dictionary from the `TEX.QCH` dictionary. Now we can link to the country and HS code descriptions. [We don't need to change the link items, because they use the `CTRY` and `HS` fields as the "link", and both of these fields were defined above].

Now, let's index the file so that we can find the records quickly:

```
MAKE.INDEX TEX.QCH.N CTRY HS
Added index for CTRY
Added index for HS
Building index 'TEX.QCH.N CTRY'...
411502 records processed
Populating index...
Building index 'TEX.QCH.N HS'...
411502 records processed
Populating index...
```

And we'll test this with a query similar to that used near the start of this section:

```
LIST TEX.QCH.N WITH CTRY EQ "CN" AND WITH HS EQ "02" AND WITH YYYYQ EQ "20123" YEAR QTR C
%SHORTNAME FOB
TEX.QCH.N.    Year   Qtr   Country............   ......FOB Value
343198        2012   3     China                         73,815,737

1 record(s) listed
```

It should return from this query quickly. Our query has used all of the indexed fields, and *QMQuery* uses those indices to locate the record(s) quickly.

Now, if we want to look up the previous year's equivalent value for this record, we essentially need to run a very similar query to this, and do it within a dictionary item. The query would be similar to:

```
LIST TEX.QCH.N WITH CTRY EQ "CN" AND WITH HS EQ "02" AND WITH YYYYQ EQ "20113" YEAR QTR C
%SHORTNAME FOB
TEX.QCH.N.    Year   Qtr   Country............   ......FOB Value
246374        2011   3     China                         39,390,693

1 record(s) listed
```

This tells us that to calculate the one-year percentage change for Chapter 02 exports to China for the third quarter of 2012 (record 343198), we need to look up record 246374. How can we do this?

We know that QMQuery can look up the field for us once it has the item-id. So, our task is to derive the item-id.

Even though the file now has a sequential number as its `ID`, the underlying primary key of the file remains the combination of quarter, country, and HS code. In other words, there should only ever be one record within the database with a given combination of quarter, country, and HS code. Therefore, if we create an I-type that creates that combination of fields, and index on that I-type, we can use the index to find any record in the database.

We'll call the I-type `QCH` and give it the expression:

```
      YYYYQ : ‘*’ : CTRY : ‘*’ : HS
```

And to test that it works, we’ll do a listing from the database:

```
LIST TEX.QCH.N YYYYQ CTRY HS QCH
TEX.QCH.N.    YYYYQ   Ctry   HS Code   QCH.........
3592          20091   CD          64   20091*CD*64
11990         20091   LK          34   20091*LK*34
15139         20091   NF          47   20091*NF*47
36407         20092   LS          49   20092*LS*49
```

Note that what we’ve done is to recreate the item-id we used in TEX.QCH.

Now, we’ll create an index using this I-type:

```
MAKE.INDEX TEX.QCH.N QCH
Added index for QCH
Building index 'TEX.QCH.N QCH'...
411502 records processed
Populating index...
```

Now, we need a method to use this index. We need to be able to query it to find the record that meets our criteria of quarter, country, and HS code. We’ll use a function to do this again, and use the SUBR function within an I-type to call it.

```
FUNCTION TEX.QCH.N.ITEMID(yyyyq, ctry, hs)
*****************************************************************************
* Bp Tex.Qch.N.Itemid - Get the item-id of TEX.QCH.N for the passed values.
*
* Author : BSS
* Created: 06 Jul 2013
* Updated: 06 Jul 2013
* Version: 1.0.0
*
* Pass:
*  yyyyq - year and quarter
*  ctry  - ISO 3166-1 alpha-2 country identifier
*  hs    - HS code (2 digit chapter level)
*
$CATALOGUE LOCAL

id = ''
IF (yyyyq EQ '') OR (ctry EQ '') OR (hs EQ '') THEN RETURN id

OPEN 'TEX.QCH.N' TO tex.qch.n ELSE RETURN id

qch = yyyyq : '*' : ctry : '*' : hs
SELECTINDEX 'QCH', qch FROM tex.qch.n TO 9
IF STATUS() OR NOT(@SELECTED) THEN RETURN id
idcnt = @SELECTED
READLIST idlist FROM 9

BEGIN CASE
  CASE (idcnt EQ 1)
    id = idlist
  CASE (idcnt GT 1)
    CRT idcnt:' items selected for ':qch
END CASE

RETURN id
*
* ------------------------------------------------------------------------ *
*
END
```

This function accepts three arguments – quarter, country and HS code. It assembles these into the form that has been indexed, and then reads that index item. The select-list that is returned is then transformed into a dynamic array. The CASE statement checks to ensure that only one item-id has been returned. If there is more than one, then an error message is displayed.

Now, we'll put this into a dictionary item named PREV.ID.YR1 with an expression of:

```
      SUBR('TEX.QCH.N.ITEMID', (YYYYQ - 10), CTRY, HS)
```

We'll add this to the end of our earlier query to test it:

```
LIST TEX.QCH.N WITH YYYYQ EQ "20123" AND WITH CTRY EQ "CN" AND WITH HS EQ "02" YEAR QTR
C%SHORTNAME FOB PREV.ID.YR1
TEX.QCH.N.   Year   Qtr   Country............   ......FOB Value   Prev ID Yr 1
343198       2012   3     China                      73,815,737         246374

1 record(s) listed
```

That ID matches the one we found earlier with our related query for the third quarter of 2011.

Now that we have that ID, the rest of the process is similar to that when we calculated the percentage changes for TEX.QCH in Section 6.4.3. Copy the FOB.PREV.YR1, FOB.CHG.YR1, and FOB.PREV.YR1% items from the TEX.QCH dictionary to the TEX.QCH.N dictionary, then update the expression in FOB.PREV.YR1 to:

```
      TRANS('TEX.QCH.N', PREV.ID.YR1, FOB, 'X')
```

The other dictionary items are correct, but need to be recompiled. From the command prompt, type:

```
CD TEX.QCH.N
Compiling PREV.ID.YR1
Compiling YEAR
Compiling QTR
Compiling QCH
Compiling FOB.PREV.YR1%
Compiling FOB.CHG.YR1
Compiling FOB.PREV.YR1
```

Now, we should be able to run our query to show percentage changes. Note that a sort order has been added as we can no longer rely on the default item-id sort to put the records in year and quarter order.

```
SORT TEX.QCH.N WITH CTRY EQ "CN" AND WITH HS EQ "01""02" BY YEAR BY QTR BREAK.ON YEAR "'V'" QTR
CTRY HS TOTAL FOB TOTAL FOB.PREV.YR1 TOTAL FOB.CHG.YR1 CALC FOB.PREV.YR1% ID.SUP
Year   Qtr   Ctry   HS Code   ......FOB Value   ....Prev Yr FOB   Change 1 Yr FOB   .FOB Value
                                                                                   1 Yr % Chg
2009   1     CN          01                                                     0
2009   1     CN          02        47,540,097                          47,540,097
2009   2     CN          01         6,128,500                           6,128,500
2009   2     CN          02        46,094,890                          46,094,890
2009   3     CN          01         5,242,213                           5,242,213
2009   3     CN          02        24,926,563                          24,926,563
2009   4     CN          01         3,465,000                           3,465,000
2009   4     CN          02        21,960,650                          21,960,650
2009                              155,357,913                 0       155,357,913

2010   1     CN          01         7,084,889                           7,084,889
2010   1     CN          02        38,700,242        47,540,097        -8,839,855        -18.6
2010   2     CN          01         5,331,666         6,128,500          -796,834        -13.0
2010   2     CN          02        37,411,198        46,094,890        -8,683,692        -18.8
2010   3     CN          01         2,983,112         5,242,213        -2,259,101        -43.1
2010   3     CN          02        26,353,058        24,926,563         1,426,495          5.7
2010   4     CN          01        20,402,116         3,465,000        16,937,116        488.8
2010   4     CN          02        33,103,077        21,960,650        11,142,427         50.7
2010                              171,369,358       155,357,913        16,011,445         10.3
```

This matches our earlier listing from TEX.QCH. So, that is good. What about total exports to China?

```
SORT TEX.QCH.N WITH CTRY EQ "CN" BY YEAR BY QTR BREAK.ON YEAR "'V'" TOTAL FOB CALC FOB.PREV.YR1%
DET.SUP NO.GRAND.TOTAL
Year    ......FOB Value   .FOB Value
                          1 Yr % Chg
2009      3,613,047,492
2010      4,809,399,673         33.1
2011      5,863,790,222         21.9
2012      6,840,528,216         16.7
2013      2,293,298,141         31.3

1666 record(s) listed
```

Once again, this matches the earlier listing. The percentage changes are correct, so everything is good there.

The other thing that is notable about these queries is that performance is quite good. Running the above query that reported on all exports to China took less than half a second (722ms) on my notebook. In that time, it has read not only the 1,666 records to get the FOB value, but it has read an equivalent number of index items, and an equivalent number of previous year items. However, the original TEX.QCH file is still quicker – an equivalent query on that file took 339 milliseconds.

In both cases, the performance has come from the indexing on the files. Compare the above times with that noted prior to the indexing of the TEX.QCH file – around 6.5 seconds to find a relatively small set of records.

7.3 Meaningful vs Sequential Item-ids

We have seen that we can structure our data files to use either meaningful data in the item-id or a sequential number. We explored the use of sequential item-ids because of the issues associated with changing a record's primary key. But, are there any similar issues associated with sequential item-ids that we need to be aware of?

Well, we had to jump through a couple more hoops to enable lookups within our data file. In particular, we had to write a *QMBasic* program to read the index to find our selected item.

You may also think it curious that the unique index value was identical to the meaningful data ID. So we've replaced a meaningful data ID with an index of exactly the same construction ….

There are two things to remember here:

- We only removed meaningful data from the ID of the TEX.QCH file. We didn't remove meaningful data from the elements that previously made up that ID. So, we haven't removed meaningful data from our selection criteria – we've only changed how we use it.
- We are now able to change any of the meaningful elements without changing the ID of the record.

Let's go back to that first point – what could we have done with the meaningful data stored in the file? Well, given that we have already identified that country names and identifiers change periodically, we should use a number as the country identifier.

On that basis, we might use the ISO 3166-1 numeric codes for the country ID (in NCY.C), with the alpha-2 code (the current ID) becoming a field. Therefore, we would use country 156 in our queries (and indices) rather than country CN.

Then, if China changed its name and 2-character country code at some point in the future, we wouldn't need to do anything at all in the TEX.QCH.N file – because it would reference country 156, and that would not change. All we'd need to do would be to change the name and identifier in the NCY.C file, and all our lookup queries would show the new country name.

There is another issue with sequential ID's that we need to be aware of: Potentially, we could end up with multiple entries for one or more of our "unique" data combinations. What do we mean by that?

Let's go back to our TEX.QCH.N file. We saw above that record 343198 contained the data for Chapter 02 exports to China in the 3rd quarter of 2012. There is nothing to stop us creating a new record (say 1343198) and entering the same identifiers for quarter, country, and HS code. Let's do that to prove the point:

```
COPY FROM TEX.QCH.N 343198,1343198
1 record(s) copied.

LIST TEX.QCH.N WITH YYYYQ EQ "20123" AND WITH CTRY EQ "CN" AND WITH HS EQ "02" FOB
TEX.QCH.N.   ......FOB Value
1343198           73,815,737
343198            73,815,737

2 record(s) listed
```

This should not be allowed to happen. The combination of quarter, country, and HS code should be unique, and when we use those directly in the ID, we enforce that uniqueness – because we can only ever store one record against a given ID.

Let's get rid of that extra record:

```
DELETE TEX.QCH.N 1343198
1 record(s) deleted
```

Now, how do we avoid multiple entries like this? Well, we are getting beyond the scope of this book – but one way would be to set up a pre-write trigger on the TEX.QCH.N file. The trigger would check the index of the file to find the item-id associated with the quarter, country, and HS code defined in the record. If the index value did not exist, then it would allow the write to the file; if the item-id was the same as the item currently being written, then it would allow the write (overwriting the existing item); if the item-id was different from the item currently being written, then it would disallow the write.

7.4 What is the "Best" ID for the Exports File?

So far, we have really considered two ID structures for the exports file. The first uses three elements of meaningful data spliced together, while the second uses a sequential number. Those ID's for equivalent items are shown below:

```
20123*CN*02

343198
```

The biggest problem with the first of these is that country codes change periodically. We don't have that issue with the second item-id, but it requires some extra steps to allow lookups to related items; it requires that we index the file using the unique combination of elements; and we need to guard against creating multiple entries for nominally unique combinations of elements.

Given that the major issue is with country codes, why don't we simply replace these in the first structure with a numeric code? That would make the first item-id:

```
20123*156*02
```

This may not totally get us away from changes in country codes, but it is a good step in the right direction. With this structure, we can't create multiple records for the same unique combination. Related item lookups are relatively easy. We don't need to index the file (although indexing will still improve performance).

What lessons can we take from this?

- You need to think carefully about your item-ids
- It is helpful to construct your ID from multiple small elements which together make up a unique combination
  - This is better than using a single sequential number to embody all uniqueness
  - Each element in the ID should not be subject to change

8 Concluding Comments

This book has only scratched the surface of *OpenQM*. There are whole sections of the product that have not been covered at all. And even for those features that have been covered, there are many options that have either been omitted or only referenced in passing.

Some of these missing features are covered in “Part 2” to this book. Even so, there are features that have been omitted (such as PROC) due to a lack of relevance to modern database usage.

In the meantime, there are plenty of general reference books available on databases in general, database design, and programming. Most of the concepts contained in those books can be used in *OpenQM*.

OpenQM also comes with well maintained documentation and online help. These are invaluable sources of information.

Finally, the best way to learn is to simply use the product. Create an account specifically for trying things out. Create some test files, write some dictionary items, write some basic programs, and ask questions in the available online forums.

Once you become familiar with the environment and the tools available, you will find it is remarkably quick and simple to develop applications. And if you then return to the world of relational databases, you will long for the simplicity of the multi-value model!

On the other hand, you may well long for some of the development tools that are available for relational databases to be available for multi-value databases. Well, some equivalents are available. Products such as DesignBais and mv.NET provide frameworks whereby standard development languages can access multi-value databases. But that is another book beyond Part 2!

Comments on this book are welcome.

Have fun as you learn more about *OpenQM*.

9 Questions and Answers

9.1 General

I want to try OpenQM. Where can I get a copy?

You can download the latest versions of *OpenQM* from www.openqm.com.

If you want to get the GPL version, you need to register on the site. Registrations are not required for downloading the commercial versions.

Alternatively, you could download the forked version of *OpenQM* called ScarletDME. Go to https://github.com/geneb/ScarletDME for this.

Do I have to pay to try OpenQM?

There are four ways to get a free copy of *OpenQM* to try:

- The simplest way is to download the commercial version, and apply a personal licence to it. You do this by typing PERSONAL into the licence number field of the licensing screen
- You can get a 30-day 4-user evaluation licence at no charge by filing in the form on the web site. From the menu, choose ‘Sales and Downloads’, then ‘Get QM Today’.
- You can download the GPL version. This is only available for Linux. You need to register with Ladybridge Systems to get access to the download area for the GPL version.
- Download ScarletDME from the link outlined above.

No free support is available for the Personal or GPL versions.

I don’t understand how something works in OpenQM. Where can I get help?

Try the *OpenQM* online help first. Simply type HELP from the command prompt and the help screens will appear. [Help is installed automatically for Windows versions – you need to download the help file for other operating systems].

The *OpenQM* documentation is essentially identical to the help files, but the documentation may be easier for some people.

Ask a question on the *OpenQM* group at Google:
https://groups.google.com/forum/?hl=en#!forum/openqm

Ask a question on the PICK and MultiValue Databases Google group:
https://groups.google.com/forum/?hl=en&fromgroups#!forum/mvdbms

Commercial licensees could send an email to *support@ladybridge.com* for assistance.

I’ve found a bug. What should I do?

Send an email to: *support@ladybridge.com*. Ladybridge is very responsive to problems, and will often fix bugs within a matter of hours.

9.2 Installation and Configuration

What software do I need to connect to OpenQM?

AccuTerm is the recommended terminal emulator. However, any telnet client should be able to connect. You can also connect Visual Basic and other Windows applications to *OpenQM* using the *QMClient* interface. Details of using this interface are not included in this book.

Where do I get AccuTerm?

You can download *AccuTerm* from www.asent.com. If you have a commercial licence for *OpenQM*, you will have been supplied with a licence number for *AccuTerm*. Otherwise, you will need to obtain a licence from AccuSoft Enterprises.

I don’t want to use AccuTerm. What else can I use?

Any telnet client should connect to *OpenQM*. Other terminal emulators you could use include:

➢ Anzio	www.anzio.com/product/anziowin
➢ Putty	www.chiark.greenend.org.uk/~sgtatham/putty/
➢ Viaduct	www.viasystemsinc.com/external/Viaduct.htm
➢ Windows Telnet	You need to enable this: Control Panel | Programs and Features | Turn Windows features on or off | Telnet Client
➢ Wintegrate	u2.rocketsoftware.com/products/u2-wintegrate/at-a-glance

Putty is free, while Windows telnet is bundled with Windows. Each of the others are commercial products. Viaduct and Wintegrate have multi-value features.

I can’t connect to OpenQM. What should I check?

By default, *OpenQM* connects to port 4242 using a telnet protocol. Make sure that your telnet client (terminal emulator) is set to use this port.

The port used by *OpenQM* can be changed. Check the configuration parameters in the qmconfig file (see Section 2.3.2). Check the PORT setting. If this is empty, then *OpenQM* should be using port 4242.

Check that the connection is not being blocked by a firewall.

If you are using the GPL version, and you have not set up the QMSrvr service, then you will need to log in using Telnet on port 23 or SSH on port 22, cd to your *OpenQM* account folder, and then invoke *OpenQM*. Make sure that you have a telnet or ssh server installed on the Linux machine.

I can connect, but I see strange characters on the screen and data doesn't appear on the screen correctly. What is wrong?

You need to set *OpenQM* and your terminal emulator to the same terminal definition. Type TERM at the command prompt to find *OpenQM*'s current terminal setting. See if this matches the emulation setting of your terminal emulator (in *AccuTerm*, you find this by selecting 'Tools | Settings | Terminal' from the menubar).

See Sections 2.5.2 and 2.5.3 for more information.

The arrow keys on my keyboard don't work for editing the commands on the command stack. What is wrong?

If you are using *AccuTerm*, then make sure that you are using one of the special *AccuTerm* terminal definitions. These are pre-configured to use the arrow keys for editing. These terminal definitions end in '-at' e.g. vt420-at.

Otherwise, check that the control keys work for editing as defined in the online help section 'The Command Editor'. If the control keys work, then you will need to program your keyboard to send those key strokes when you use the arrow keys.

When I try to connect, I get a message saying 'User limit reached'. What does this mean?

OpenQM is licensed on the number of concurrent connections. This message means that the *OpenQM* server is already connected to its maximum number of licensed connections. You will need to disconnect one of those current connections before you can log on, or you can purchase additional licenses for your system.

How do I update the license on my system?

Log to the QMSYS account, and type in: UPDATE.LICENCE

Update the licence details using the information supplied by Ladybridge Systems or your *OpenQM* dealer. You will need to log off and on before the new licence settings become operative.

OpenQM has lots of options available. How do I find out which ones are active?

Just type OPTION at the command prompt. This will list all options and tell you the status of each.

Similarly, you can type CONFIG at the command prompt to find the status of each of the configuration parameters.

9.3 Accounts

How do I find out what accounts are on the system?

From the command prompt, type:

```
SORT QM.ACCOUNTS
```

This tells you which accounts are registered with *OpenQM*.

- This does not mean that all of these accounts are actually present.
- Nor does it mean that there are not other accounts that are not registered.

As always, it is good practice to make sure that the QM.ACCOUNTS file is kept consistent with the accounts on the system. This will be done automatically if you maintain your accounts with the CREATE.ACCOUNT and DELETE.ACCOUNT verbs.

How do I log into an account that is not registered in QM.ACCOUNTS?

If you are already logged into an *OpenQM* account, use the LOGTO verb:

```
LOGTO account_path
```

For example:

```
LOGTO /home/qb/db/qbtest
```

If you are not already logged into *OpenQM*, then you need to change to the account folder, then invoke *OpenQM*:

```
cd /home/qb/db/qbtest
qm
```

How do I add an account to the QM.ACCOUNTS register?

Use one of the editors to create an item in the QM.ACCOUNTS register. The ID is the name of the account; the first field is the path to the account; and the second field is an optional description of the account. For example:

```
CT QM.ACCOUNTS QMINTRO
QM.ACCOUNTS QMINTRO
1: E:\QM\QMINTRO
2: Account for "Getting Started 1"
```

9.4 Files

How do I add a file description so that it appears in the file listings?

The list file commands (LISTF, LISTQ etc) read their information from the VOC file. To add a file description, you simply need to edit the VOC item for the file. For example, change the following VOC item:

```
F
XRATES
XRATES.DIC
```

to:

```
F Exchange rates
XRATES
XRATES.DIC
```

Now, the description ‘Exchange rates’ will appear when you list files using the LISTF commands.

How do I enter data into files?

Section 4.4 of this book showed how to use the *AccuTerm* utilities to import data from a CSV file. You can create CSV files using any spreadsheet application.

AccuTerm provides the ability to upload a number of other source data formats including an Access database, and direct from Excel. *QMBasic* programs can also be transferred from text files on the client PC to program items on the *OpenQM* server.

You can use the *OpenQM* editors, the MODIFY editor in particular, to enter individual data items. MODIFY uses the file dictionary to provide appropriate prompts for data entry, and does input conversion so that date and money fields are appropriately stored within the *OpenQM* files.

The UPDATE.RECORD command is mostly intended to provide a way to update numerous records quickly, but can also be used to update or create a single item in a file. For example, the following statement creates a new record in TEX.QCH.N with an item-id of '-1'.

```
UPDATE.RECORD TEX.QCH.N -1 YYYYQ,20151 CTRY,"X5" HS,02 FOB,12345 CREATING
One * represents 10 records.

1 record updated
```

Finally, you can use custom written programs to enter data. Custom written programs allow a much greater degree of data validation on entry, and can be made much more user-friendly than the editors. The development of programs has not been covered in this book.

I want to delete a lot of data in a file. Is there a quick way to do this?

To delete ALL of the data in the file, use the CLEAR.FILE verb.

```
CLEAR.FILE DATA filename
```

If you don't want to delete all of the data, then it helps if you are able to select all of the records that you want to delete using *QMQuery* (e.g. in a SORT statement). Then you can use the select-list functionality of the DELETE verb to delete the data:

```
SELECT filename WITH selection-criteria
DELETE filename
```

Note that no item-id's were specified in the DELETE statement. In this case, the DELETE verb will delete all records that were returned by the preceding SELECT statement.

Be careful using this form of the DELETE verb. Test the SELECT statement thoroughly as a SORT or LIST statement first to make sure you are only selecting the records that you want to delete.

How do I back up my data?

Make sure everyone is logged off from the system. Back up the directory tree containing your *OpenQM* data using whatever Operating System backup utilities you like.

Note that this backup strategy can only restore entire data files – it cannot restore selected items from within an *OpenQM* file (unless it is a DIRECTORY file).

See the section in the online help titled 'Backup and Restore' for further information.

How do I restore just some of the items from a backup file to my working copy?

Let's say that the file is named MYFILE. Save your backup copy of MYFILE to somewhere convenient – say C:\Temp. Now, create a VOC entry named MYFILE.BAK that looks like:

```
F
C:\Temp\MYFILE
MYFILE.DIC
```

Note that this is using your existing dictionary for MYFILE. Now, all you need to do is COPY the (selected) items from MYFILE.BAK to MYFILE. For example:

```
SELECT MYFILE.BAK WITH DATE EQ "30 JUN 2013"
COPY FROM MYFILE.BAK TO MYFILE
```

To only copy items that exist in MYFILE.BAK but not MYFILE:

```
SELECT MYFILE.BAK
NSELECT MYFILE
COPY FROM MYFILE.BAK TO MYFILE
```

or:

```
COPY FROM MYFILE.BAK TO MYFILE ALL
```

This last version works because COPY does not overwrite items unless you specify the OVERWRITING option. Therefore, existing items are left alone, and only new items are added.

Finally, remove your backup file using a file manager, and delete the VOC entry that points to it:

```
DELETE VOC MYFILE.BAK
```

I have an OpenQM file from a Windows installation. How do I put this onto a Linux installation of OpenQM? Or vice-versa.

Simply save it into the Linux directory tree, and create a VOC entry for the file that points to its saved location. The internal format of *OpenQM* files is the same regardless of the operating system.

If your Linux installation is the GPL version, then you will need to rename the file components from '~0' and '~1' (and others if present) to '%0' and '%1'.

Recent versions of *OpenQM* will automatically create '%' equivalents of the '~' files. It is best to remove the '~' files once this has done. Alternatively, you could simply rename the files when you transfer them between the systems.

9.5 QMQuery

How do I find out what dictionary items are available?

Use *QMQuery* to show you the dictionary items:

```
SORT DICT filename
```

When I list items using QMQuery, the item-id wraps onto a second line. Why doesn't this display correctly?

When *OpenQM* creates a file and its dictionary, it simply assigns default characteristics to the @ID dictionary item. These defaults may not match the data you use for the item-id. So simply MODIFY the @ID dictionary item to match the data you are using. In this case, you would want to increase the field width, but you may also want to change the field justification (see next question).

OpenQM is sorting data into the wrong order – 11 is sorting before 2 – what is happening?

You have specified a left-justified format for this field, or left the format field blank. Fields should be defined as left justified for text data and right-justified for numeric data.

I get an error message which says: 0 record(s) listed. 'xxxx' not found. What does this mean?

This literally means that *OpenQM* has looked in the dictionary for an element specified in your command, and has not found it there. This usually means either:

- You have specified a dictionary item that doesn't exist
- You have mis-specified a relational comparison in the selection criteria.

Check which word *OpenQM* has specified as not being found. If you think that word is a dictionary item, then probably you spelt the word wrongly, or used some variation of the word that is not in the dictionary. Check the contents of the dictionary to find the correct word.

If the word was a test value in the selection criteria, then you have mis-specified the comparison. Usually this occurs when you haven't specified a relational operator between a field name and the value you are testing for. For example:

```
SORT XRATES WITH YEAR 2006 USD

0 record(s) listed
'2006' not found
```

This is a valid syntax in some multi-value databases. The assumption is made that if no relational operator is specified, then an 'equals' is implied. e.g.

```
SORT XRATES WITH YEAR EQ 2006 USD
```

You can enable the assumption of an implied equals by setting the PICK.IMPLIED.EQ option. You can do this by adding the following line to the LOGIN item:

```
OPTION PICK.IMPLIED.EQ ON
```

I want my QMQuery output to go direct to Excel. How do I do that?

Section 6.8 discusses how to write *QMQuery* output direct to an Operating System level file. If you use the CSV output option with a tab delimiter, this can be read by Excel.

A fully formatted Excel output direct from *QMQuery* has been discussed on the *OpenQM* Google group, and is a possible future enhancement of *QMQuery*.

There are utilities available to create Excel spreadsheets direct from *OpenQM*. See for example:

SysCtrl account at: www.rushflat.co.nz

NebulaXLite at: nebula-rnd.com/products/xlite.htm

My saved select-lists disappeared. What happened to them?

If you used the CLEAN.ACCOUNT command, this will have cleared the $SAVEDLISTS file. Therefore, if you want permanent select-lists, you should store them somewhere other than in the default $SAVEDLISTS file. This requires a little extra work:

Say your file for permanent select-lists is named SLISTS. We need to copy our standard select list from the $SAVEDLISTS file to the SLISTS file. Then, to use the stored select-list, we use the QSELECT command rather than GET.LIST. This is shown below:

```
SSELECT CUSTOMERS
10 record(s) selected to list 0
::SAVE.LIST CUSTOMERS
10 records saved to select list 'CUSTOMERS'
COPY.LIST CUSTOMERS TO SLISTS
1 record(s) copied
QSELECT SLISTS CUSTOMERS
10 record(s) selected to select list
::
```

10 Quick Reference

10.1 Accounts

CREATE.ACCOUNT *accountname pathname*

DELETE.ACCOUNT *accountname*

10.2 Files

Creating and deleting files

CREATE.FILE *filename* {DIRECTORY}

CREATE.FILE DICT *dictname*

CREATE.FILE DATA *dictname , dataname* {DIRECTORY}

DELETE.FILE *filename*

DELETE.FILE DATA *dictname , dataname*

DELETE.FILE DICT *dictname*

Listing files in account

LISTF	List local and remote files referenced in the VOC
LISTFL	List local files referenced in the VOC
LISTFR	List remote files referenced in the VOC
LISTQ	List Q-pointers referenced in the VOC

Editing items in files

ED {DICT} *filename item-id*

MODIFY {DICT} *filename item-id*

SED {DICT} *filename item-id*

WED {DICT} *filename item-id*

List the contents of an item

CT {DICT} *filename item-id*

LIST.ITEM {DICT} *filename item-id*

Delete an item

DELETE {DICT} *filename item-id*

10.3 VOC Entries

10.3.1 F-type

VOC entries control much of the system. Therefore, care should be taken when adding or editing these items. Some items may be freely edited – others should be left to *OpenQM* to manage.

An F-type defines a file within *OpenQM*. You should leave *OpenQM* to manage the F-type entries in the VOC.

F-type entries can be listed with:

LISTF	List all files in the account
LISTFL	List local files
LISTFR	List remote files

10.3.2 Q-type

A Q-type is a pointer to a file in another account, or on another server. Q-types usually need to be manually entered into the VOC using one of the available editors. Their structure is:

Field	**Contents**	**Example**
1	Q {descriptive text}	Q
2	Account	QMSYS
3	File name	BP
4	{Server name}	

Q-type entries can be listed using the LISTQ command.

10.3.3 Verbs

Verbs are the first word of the commands you give to *OpenQM*. You should not edit or manually enter these items. Verbs can be listed using the LISTV command.

10.3.4 Keywords

Keywords are part of the commands given to *OpenQM*, but are not verbs. You should not edit or manually enter these items. Keywords can be listed using the LISTK command.

10.3.5 Paragraphs and sentences

Paragraphs and sentences are stored commands, saved by the user. The distinction between paragraphs and sentences is primarily one of length – a sentence is a single command, whereas a paragraph comprises multiple commands and can include some high-level programming. Sentences are listed using the LISTS command, while Paragraphs are listed using the LISTPA command.

10.3.6 Other entries

The VOC may also contain a number of other entries. These include phrases and X-type entries as described below under dictionary structures. Phrases may be listed using the LISTPH command, but you will need to query the VOC to find the X-type entries.

Other VOC entries include menus, procs, and remote items. See the online help for more information on these items.

10.4 Dictionary Structures

Dictionary entries are primarily for use by *QMQuery*, although they also play an important role in documenting the file structures.

All dictionary entries are identified by *OpenQM* by the first character (or sometimes two) in the first field. Users may enter whatever they like after this for their own descriptive purposes.

10.4.1 Displaying dictionary entries

SORT DICT *filename*

10.4.2 D-type

D-type dictionary entries describe the data as it is stored in the file.

Field	Contents	Example
1	D {descriptive text}	D
2	Field number	1
3	Conversion code	MR44,
4	Display name	US Dollar
5	Format specification	7R
6	S/M flag (single or multi-valued)	S
7	Association	

10.4.3 I-type

I-type dictionary items are indirect. They calculate new values from the data stored in the file. Calculations can combine data from multiple fields and multiple data files.

Field	Contents	Example
1	I {descriptive text}	I
2	Expression	FIELD(@ID, '*', 1)
3	Conversion code	
4	Display name	YYYYQ
5	Format specification	5R
6	S/M flag (single or multi-valued)	S
7	Association	

10.4.4 Links

Links provide a link between one file and another, allowing the first file to use the dictionary items defined in the second file.

Field	Contents	Example
1	L {descriptive text}	L
2	ID expression	@ID
3	File name	IRATES

10.4.5 Other

X-type items are ignored by *QMQuery*. Developers may use these items for whatever they wish.

Field	Contents	Example
1	X {descriptive text}	X
2	User data	Comments

10.4.6 Phrases

Phrases allow frequently used groups of query words to be saved as a single word. *QMQuery* expands this single word back to the full group of words.

Field	Contents	Example
1	PH {descriptive text}	PH
2	Phrase expansion	DESC FMT ‘60T’

10.5 Alternate Key Indices

CREATE.INDEX *filename* *field(s)* {NO.NULLS}

BUILD.INDEX *filename* *field(s)* | ALL

MAKE.INDEX *filename* *field(s)* {NO.NULLS}

LIST.INDEX *filename field(s)* | ALL {STATISTICS} {DETAIL}

10.6 QMQuery

10.6.1 General syntax

verb {DICT} *filename* {USING {DICT} *filename*} {*selection.clause* } {*sort.clause* } {*display.clause* } {*record.id...*} {FROM *select.list.no*} {TO *select.list.no*} {*modifiers*}

10.6.2 Verbs

Verbs to display data

LIST	Display records in specified order
SORT	Display records in specified order with an additional item-id sort
LIST.LABEL	Create mailing labels in specified order
SORT.LABEL	Create mailing labels in specified order with an additional item-id sort

LIST.ITEM	Displays selected items in internal format
SORT.ITEM	Displays selected items in internal format

Verbs to select data

SELECT	Create a list of of item-ids in specified order
SSELECT	Create a list of of item-ids in specified order with an additional item-id sort
SHOW	An interactive means of selecting item-ids for inclusion in a select list
SEARCH	Create a list of of item-ids in unsorted order where the item contains a specified text string

Other verbs

REFORMAT	Restructure the selected data by writing it into a new file
SREFORMAT	Restructure the selected data by writing it into a new file
COUNT	Count items meeting the specified criteria
SUM	Sums the fields in the items meeting the specified criteria

10.6.3 Selection clause

WITH {EVERY} *condition* {AND | OR *condition*}

WHEN *condition*

where:

condition is:	*field operator value*
or:	*field1 operator field2*

and operator is one of the terms in the following list:

EQ	=	EQUAL		
NE	#	NOT	<>	><
LT	<	LESS	BEFORE	
LE	<=	=<		
GT	>	GREATER	AFTER	
GE	>=	=>		
LIKE	MATCHES	MATCHING		
UNLIKE	NOT.MATCHING			
SAID	SPOKEN	~		
NO				
BETWEEN				

In multi-valued items, the WITH clause returns all items matching the specified condition(s), whereas the WHEN clause returns only those values within the item matching the condition(s).

10.6.4 Sort clause

BY *field* {BY *field* ... }

BY.DSND *field* ...

BY.EXP *field* ...

BY.EXP.DSND *field* ...

10.6.5 Display clause

{*prefix*} *display-item* {*suffix*} ...

where these items are defined as follows:

Prefix	**Display-item**	**Suffix**
AVG	D-type item	CONV "*code*"
BREAK.ON "*text*"	I-type item	FMT "*spec*"
BREAK.SUP "*text*"	A/S type item	COL.HDG "*text*"
CALC	EVAL expr {AS xx}	ASSOC "*name*"
CUMULATIVE	Fn	ASSOC.WITH *field*
ENUM		DISPLAY.LIKE *field*
MAX		SINGLE.VALUE
MEDIAN		MULTI.VALUE (or MULTIVALUED)
MIN		NO.NULLS
MODE		
PCT {n}		
RANGE		
TOTAL		

where Fn references field numbers e.g. `F1`, `F2`, `F3` etc.

10.6.6 Modifiers

COL.HDR.SUP
COL.SUP
COUNT.SUP
DBL.SPC
DET.SUP
GRAND.TOTAL "*text'options'*"
HDR.SUP
ID.ONLY
ID.SUP
LPTR
NEW.PAGE
NO.GRAND.TOTAL

11 Appendix 1 – Installation

11.1 Windows

Download latest version of OpenQM

The installation executable can be downloaded from www.openqm.com. The Windows executable contains all necessary components such as PDF documentation, the on-line help system, and the *QMClient* dll that enables you to connect *OpenQM* to Visual Basic. This executable has a name in the form:

qm_v-p-r.exe

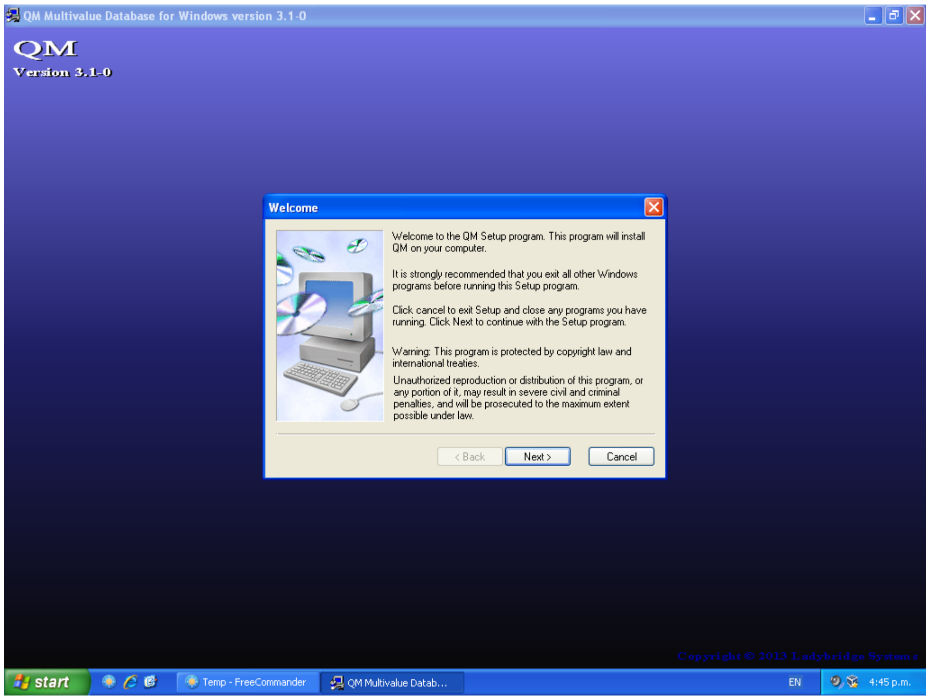
where:	v = version
	p = point
	r = release

For example, the Windows version of *OpenQM* 3.1-0 has an executable name of: *qm_3-1-0.exe* and is about 12.5 MB in size.

The latest version of *OpenQM* should be downloaded and placed in a suitable folder on the target computer (e.g. C:\Temp).

Start the installation

Using your file manager, navigate to the folder where the *OpenQM* executable has been placed, and double-click on the executable to start the installation process. The following screen should appear:



Welcome to the QM Setup program. This program will install QM on your computer.

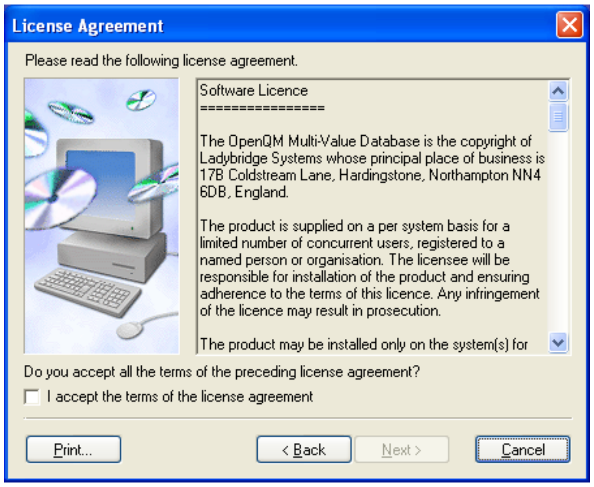
It is strongly recommended that you exit all other Windows programs before running this Setup program.

Click cancel to exit Setup and close any programs you have running. Click Next to continue with the Setup program.

Warning: This program is protected by copyright law and international treaties.

Unauthorized reproduction or distribution of this program, or any portion of it, may result in severe civil and criminal penalties, and will be prosecuted to the maximum extent possible under law.

Click on 'Next'. The 'License Agreement' dialog box will appear:



Please read the following license agreement.

Software Licence
================

The OpenQM Multi-Value Database is the copyright of Ladybridge Systems whose principal place of business is 17B Coldstream Lane, Hardingstone, Northampton NN4 6DB, England.

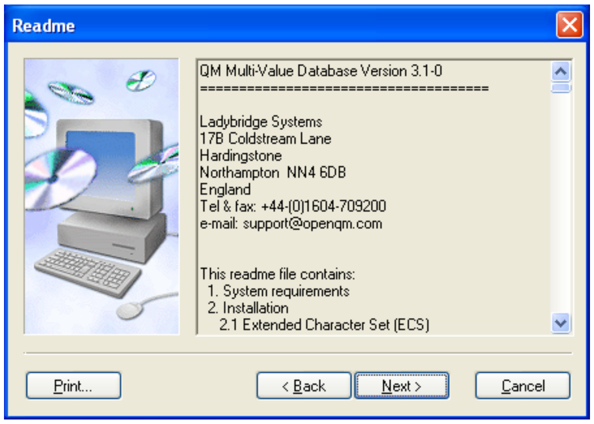
The product is supplied on a per system basis for a limited number of concurrent users, registered to a named person or organisation. The licensee will be responsible for installation of the product and ensuring adherence to the terms of this licence. Any infringement of the licence may result in prosecution.

The product may be installed only on the system(s) for

Do you accept all the terms of the preceding license agreement?

☐ I accept the terms of the license agreement

Click in the check box to accept the license agreement, and then click on 'Next'.

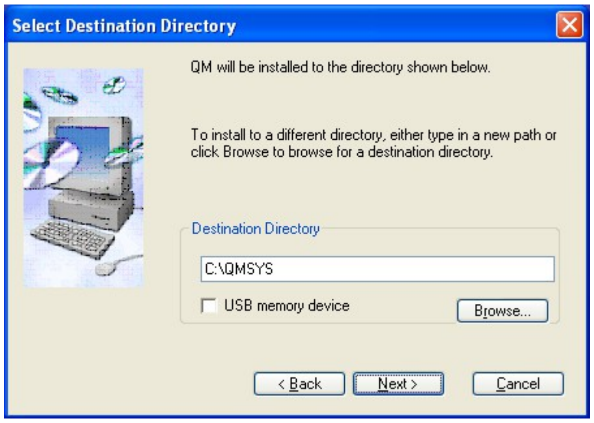


QM Multi-Value Database Version 3.1-0
=====================================

Ladybridge Systems
17B Coldstream Lane
Hardingstone
Northampton NN4 6DB
England
Tel & fax: +44-(0)1604-709200
e-mail: support@openqm.com

This readme file contains:
1. System requirements
2. Installation
2.1 Extended Character Set (ECS)

The Readme file is displayed. Click on 'Next'.



QM will be installed to the directory shown below.

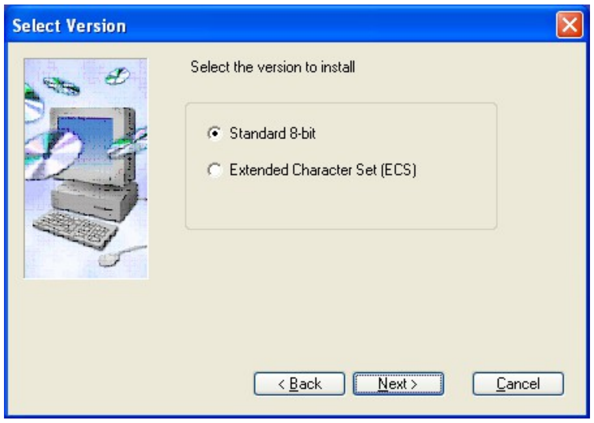
To install to a different directory, either type in a new path or click Browse to browse for a destination directory.

Destination Directory

C:\QMSYS

USB memory device

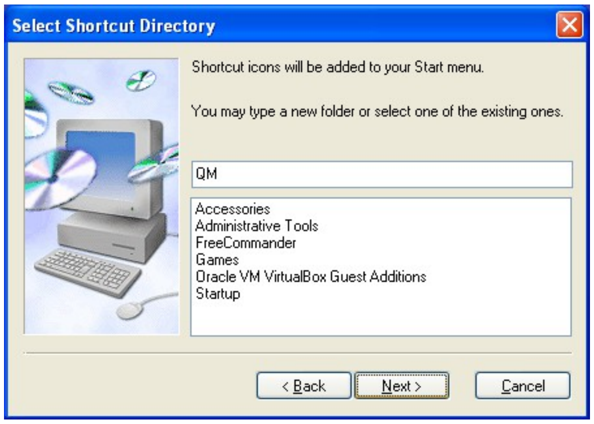
It is recommended that the default destination directory be accepted. The USB memory device option allows *OpenQM* to be installed onto a USB key. Leave this option unchecked, and click on 'Next'.



Select the version to install

[✓ Standard 8-bit]
Extended Character Set (ECS)

Select the version to install. Unless you have need to store data in UniCode, then choose the 'Standard 8-bit' installation. Click on 'Next'.



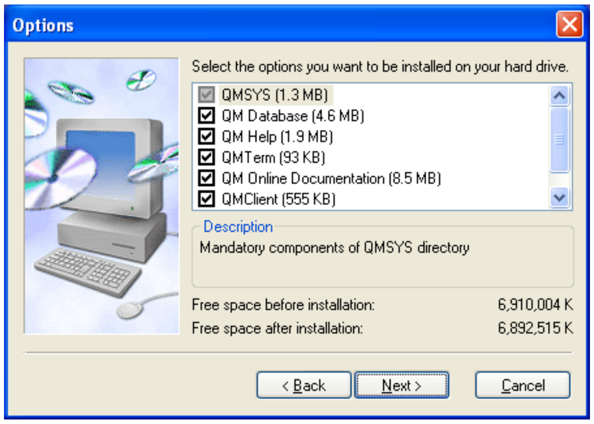
Shortcut icons will be added to your Start menu.

You may type a new folder or select one of the existing ones.

QM

Accessories
Administrative Tools
FreeCommander
Games
Oracle VM VirtualBox Guest Additions
Startup

Select where to place the program start icons. It is recommended that this be left at the default setting. Click on 'Next'.



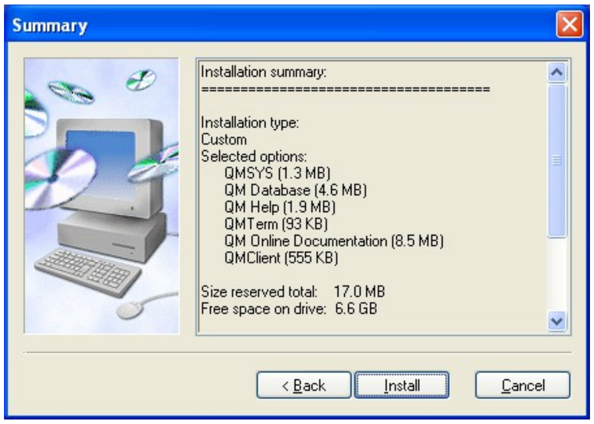
Select the options you want to be installed on your hard drive.

- [✓] QMSYS (1.3 MB)
- [✓] QM Database (4.6 MB)
- [✓] QM Help (1.9 MB)
- [✓] QMTerm (93 KB)
- [✓] QM Online Documentation (8.5 MB)
- [✓] QMClient (555 KB)

Description
Mandatory components of QMSYS directory

Free space before installation:	6,910,004 K
Free space after installation:	6,892,515 K

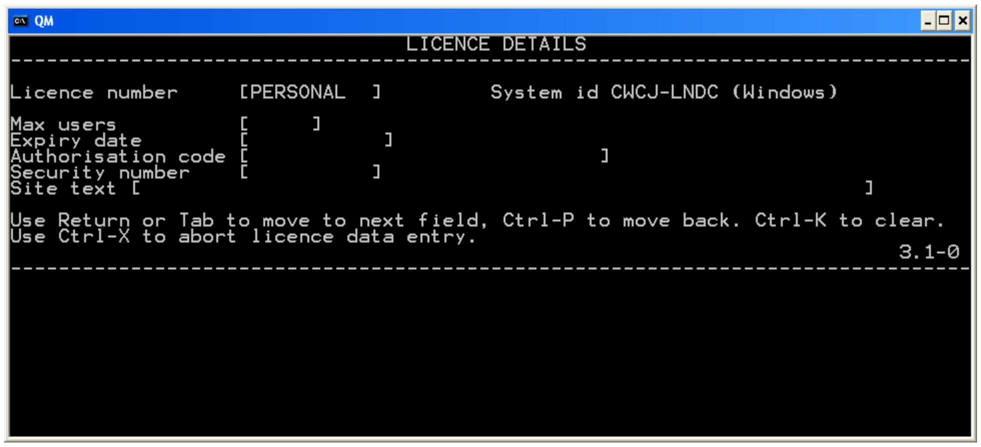
Choose which components of *OpenQM* to install. It is recommended that all components be selected and installed. Click on 'Next'.



```
Installation summary:
=====================================

Installation type:
Custom
Selected options:
    QMSYS (1.3 MB)
    QM Database (4.6 MB)
    QM Help (1.9 MB)
    QMTerm (93 KB)
    QM Online Documentation (8.5 MB)
    QMClient (555 KB)

Size reserved total:    17.0 MB
Free space on drive:  6.6 GB
```

A summary of the installation options is now shown. Click on 'Next'.

A status screen will appear showing the progress through the installation. Once installation is complete, the following licence details screen will appear.

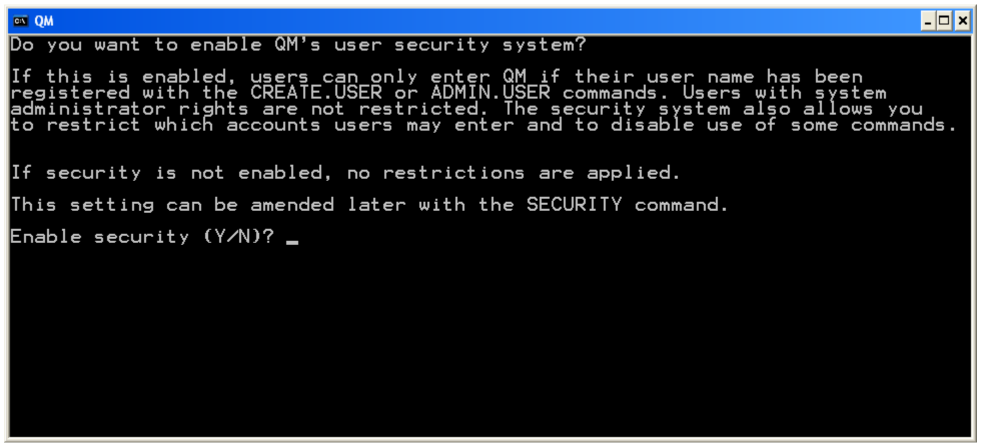


```
                                 LICENCE DETAILS
--------------------------------------------------------------------------------

Licence number     [PERSONAL  ]        System id CWCJ-LNDC (Windows)

Max users          [     ]
Expiry date        [           ]
Authorisation code [                             ]
Security number    [          ]
Site text [                                                            ]

Use Return or Tab to move to next field, Ctrl-P to move back. Ctrl-K to clear.
Use Ctrl-X to abort licence data entry.
                                                                          3.1-0
--------------------------------------------------------------------------------
```

If you have purchased a licence for *OpenQM*, then fill in the licence details exactly as provided by the dealer or distributor.

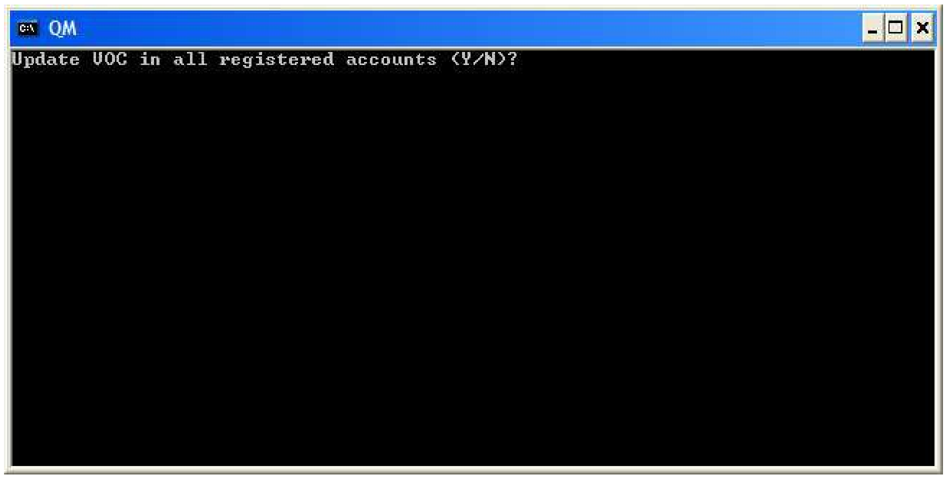
If you have not purchased a licence, then enter ‘PERSONAL’ in the licence number field. You cannot fill in any other fields with a personal licence.

The licence details will be processed once you move off the last field (or first field for the personal licence).



```
Do you want to enable QM's user security system?

If this is enabled, users can only enter QM if their user name has been
registered with the CREATE.USER or ADMIN.USER commands. Users with system
administrator rights are not restricted. The security system also allows you
to restrict which accounts users may enter and to disable use of some commands.

If security is not enabled, no restrictions are applied.

This setting can be amended later with the SECURITY command.

Enable security (Y/N)? _
```

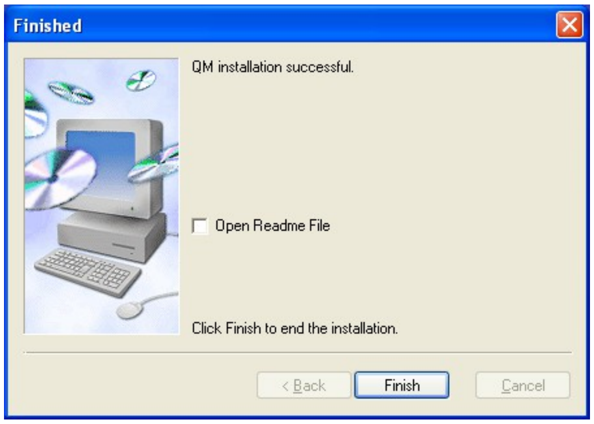
You are then asked if you wish to enable security. At this point, answer ‘N’. You can turn this on later if necessary.



```
Update VOC in all registered accounts (Y/N)?
```

The update accounts screen will now appear. This ensures that all accounts have the latest definitions of the standard commands. This can be run manually from within *OpenQM* using the UPDATE.ACCOUNT command. Answer 'Y' to the prompt.

If you have a firewall on your machine, it may warn you that QmSrvr is attempting to access the internet. Allow access, and instruct your firewall to continue to allow access.



QM installation successful.

Click Finish to end the installation.

Click on 'Finish' to complete the installation. Optionally, you can choose to view the Readme file, but this is not required.

11.2 Linux

Download the latest version of OpenQM from www.openqm.com. Note: This is named as a '.txt' file, so your browser will open it by default. Right-click on the link, choose 'Save As', and save it somewhere on your system.

Open a terminal session in the folder where you saved the text file. Type:

```
ls -l
total 1832
-rw-rw-r-- 1 brian brian 1870705 Jul  9 15:48 qm_3-1-0.txt
```

This shows the file is not executable. Type:

```
chmod +x qm_3-1-0.txt
ls -l
total 1832
-rwxrwxr-x 1 brian brian 1870705 Jul  9 15:48 qm_3-1-0.txt
```

We now need to run the file. This must be done in superuser mode. On Ubuntu based distributions, we do this via sudo. On other distributions, you may have to log in as root.

```
sudo ./qm_3-1-0.txt
[sudo] password for brian:
This script will install the QM multivalue database.
Before continuing, please verify the following...
  You must be running with superuser rights.
  QM must not be running (Use 'qm -stop' with all QM users off)
Continue with installation (y/n)?
```

Answer ‘Y’. The software will be unpacked.

```
Press return to view the software licence
```

Scroll through the software licence until you get to the end.

```
Do you agree with the terms of this licence?                         Answer Y
QMSYS directory path (/usr/qmsys):                                   Press enter

There are two versions of QM available for install:
The standard 8-bit version and the ECS (extended character set) version.
Install ECS version (default N)?                                     Press enter
```

The software will be installed. You will then get to the licensing screen:

```
                                 LICENCE DETAILS
--------------------------------------------------------------------------------

Licence number     [          ]        System id HLQJ-KTWM (Linux)

Max users          [     ]
Expiry date        [           ]
Authorisation code [                            ]
Security number    [          ]
Site text [                                                         ]

Use Return or Tab to move to next field, Ctrl-P to move back. Ctrl-K to clear.
Use Ctrl-X to abort licence data entry.
                                                                       3.1-0
--------------------------------------------------------------------------------
```

Enter your licence details. If you don’t have a licence yet, enter `PERSONAL` in the licence number field. You are then taken to the security screen:

```
Do you want to enable QM's user security system?

If this is enabled, users can only enter QM if their user name has been
registered with the CREATE.USER or ADMIN.USER commands. Users with system
administrator rights are not restricted. The security system also allows you
to restrict which accounts users may enter and to disable use of some commands.

If security is not enabled, no restrictions are applied.

This setting can be amended later with the SECURITY command.

Enable security (Y/N)?
```

Answer ‘N’ here. You can set up security later if you need to.

```
QM has been restarted
Installation complete
```

Check that the system is operational:

```
cd /usr/qmsys
qm
[ QM Rev 3.1-0   Copyright Ladybridge Systems, 2013 ]
[ 0103035549  For personal use only ]
:
```

That is OK. Exit from the OpenQM prompt by typing: **OFF**

You can delete the installation file now if you wish.

11.3 USB Stick

11.3.1 Installation

The easy way

Download the 'USB demonstration' zip file from www.openqm.com. Unzip the contents of this file onto a USB stick.

The next time you insert the USB stick into a Windows PC, an *AccuTerm* window should pop up. Click on the 'Launch AccuTerm 7 Demo' link to start *AccuTerm* and OpenQM.

If you have autorun disabled, then browse to the USB stick in your file manager, and double-click on 'startdemo.bat'.

This demonstration uses *OpenQM* 2.12-4 and *AccuTerm* 7. Neither of these is the latest version, but are still perfectly functional. It is also easy to update them to the current versions.

Normal installation

Installing *OpenQM* on a USB stick is almost identical to installing on Windows.

It is advisable to prepare the USB stick prior to starting the installation. From version 3.1-0 onwards, this is supposedly not required – but I still needed to carry out this step.

Download the USBConfig tool from the Downloads page at www.openqm.com. Make sure that your USB stick is NOT connected to your computer, then run usbconfig.exe.

- The program will warn you that the USB should not be connected. Answer 'Y' to continue.
- Plug the USB stick into the computer, and note the drive letter that is assigned to it.
- Enter the drive letter into the program. The program will initialise the stick.

Create a folder on the USB stick for *OpenQM*. Call this QMSYS.

Now run the *OpenQM* installation program. When you get to the 'Select Destination Directory' dialog box, check the 'USB memory device' checkbox, and then browse to the QMSYS folder on your USB stick. Click on 'Next'.

The rest of the installation is identical to the Windows installation.

11.3.2 Using OpenQM on a USB Stick

If you installed the USB demonstration package, then everything is already set up. All you need to do is double-click on the 'startdemo.bat' file to start *OpenQM* and *AccuTerm*.

Otherwise, to make a network connection to *OpenQM* on a USB stick, you need to start the 'QMUSBSrvr.exe' program in the QMSYS\bin folder. This will open a command window that must stay open throughout your *OpenQM* session.

Note that if you installed the USB demonstration package, then OpenQM will be listening on port 4240 rather than the standard 4242. Similarly, *QMClient* connections are made on 4241 instead of 4243. Presumably, this was done so that any USB installation would not conflict with an OpenQM installation on your hard drive. These ports are defined in the qmconfig file. See Section 2.3.2 and the online help for more information.

11.4 GPL Version (Linux)

Full instructions for installing the GPL version of *OpenQM* can be found at:

- www.rushflat.co.nz
- www.geneb.org/qm/fedora_notes.txt
- www.billabong-services.co.uk/anji

11.5 AccuTerm

11.5.1 General considerations

AccuTerm has a number of functions. These include:

- acting as a front-end onto the QM database (terminal emulation)
- providing data transfer facilities between multi-value databases and the host operating system, or between different multi-value databases
- providing a GUI development environment for multi-value databases
- providing facilities allowing Windows programs to access data held in multi-value databases
- providing facilities allowing multi-value programs to control Windows applications.

This sweep of functionality requires components on both the client and the server – even when the client and the server are on the same machine. The general process for installation is to install the client software, and then use the *AccuTerm* utilities to install the server components.

Note that *AccuTerm* is only available on Windows. A version for Linux has been mentioned in the AccuSoft forums, but no timeframe for delivery has been announced. Reportedly, *AccuTerm* runs well under WINE, although you need to either change the default fonts, or register the *AccuTerm* fonts within WINE.

If you have purchased a commercial licence for *OpenQM*, you will have received an authorisation code to unlock the *AccuTerm* software.

If you are using the personal licence for *OpenQM*, you will either need to purchase an *AccuTerm* licence, or use the personal version of *AccuTerm*. Alternatively, you could use some other terminal emulation program to attach to *OpenQM*.

A single user licence for *AccuTerm* costs US$129 for download. Note, that this is more than the cost of an *OpenQM* licence which comes bundled with an *AccuTerm* licence. Note, however, that the bundled version of *AccuTerm* is licensed *"only for connection to the Commercial QM Bundle"*. Therefore, you should not use the bundled version of *AccuTerm* for connection to other environments.

The personal version of *AccuTerm* is free, but is restricted to local connections – i.e. it cannot operate over a network connection. The personal version is restricted to non-commercial use only.

You need to register with AccuSoft Enterprises to get an unlock code for any version of *AccuTerm* (unless one was provided with your *OpenQM* licence).

11.5.2 Installation steps

Download the latest version of AccuTerm

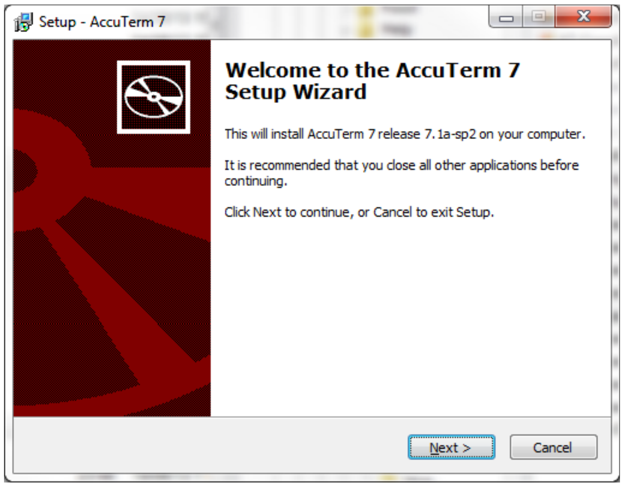
AccuTerm can be downloaded from AccuSoft Enterprises at www.asent.com

The latest version of *AccuTerm* is version 7.1a sp2 and has a file name of *atw71asp2.exe* and is about 23 MB in size.

Place the downloaded file in a suitable folder on the computer (e.g. C:\Temp).

Start the installation

Using your file manager, navigate to the folder where the *AccuTerm* executable has been placed, and double-click on the executable to start the installation process. The following screen should appear:



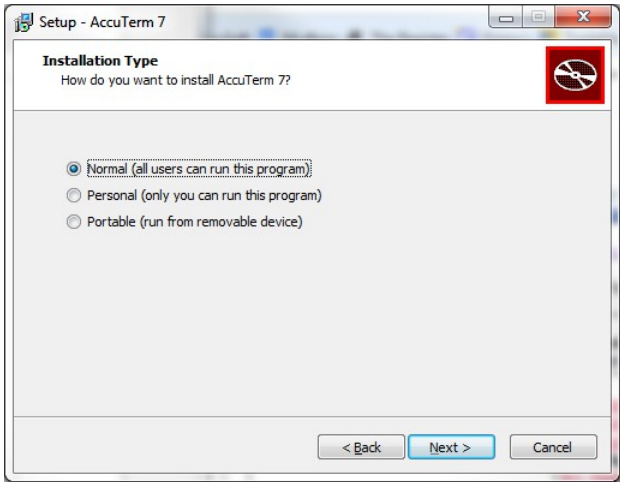
Welcome to the AccuTerm 7 Setup Wizard

This will install AccuTerm 7 release 7.1a-sp2 on your computer.

It is recommended that you close all other applications before continuing.

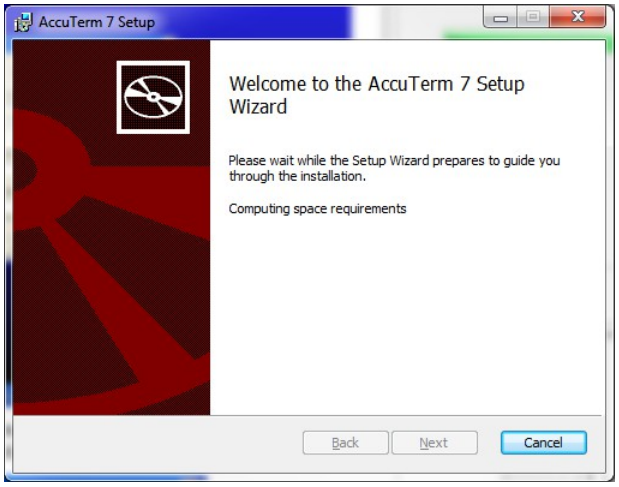
Click Next to continue, or Cancel to exit Setup.

Click on 'Next'.



Normal (all users can run this program) [✓]
Personal (only you can run this program)
Portable (run from removable device)

If you are installing onto a USB stick, select the 'Portable' option. Otherwise, select the 'Normal' option. Click on 'Next'.

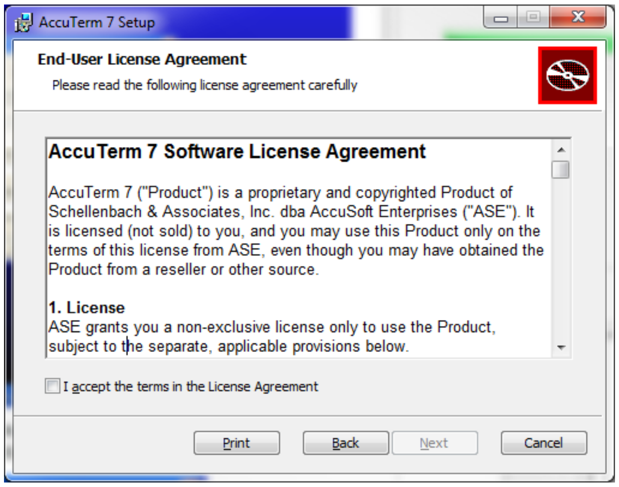


Welcome to the AccuTerm 7 Setup Wizard

Please wait while the Setup Wizard prepares to guide you through the installation.

Computing space requirements

Click on 'Next'.


AccuTerm 7 Setup
End-User License Agreement
Please read the following license agreement carefully
Print
Back
Next
Cancel

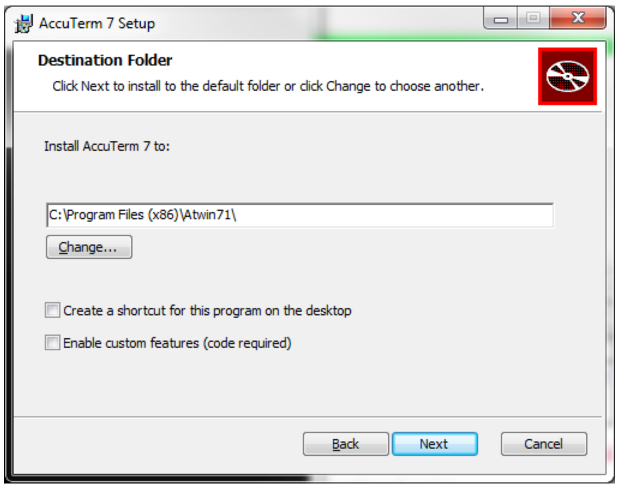
AccuTerm 7 Software License Agreement

AccuTerm 7 ("Product") is a proprietary and copyrighted Product of Schellenbach & Associates, Inc. dba AccuSoft Enterprises ("ASE"). It is licensed (not sold) to you, and you may use this Product only on the terms of this license from ASE, even though you may have obtained the Product from a reseller or other source.

1. License
ASE grants you a non-exclusive license only to use the Product, subject to the separate, applicable provisions below.

I accept the terms in the License Agreement

Accept the license agreement and then click on 'Next'.


AccuTerm 7 Setup

Destination Folder

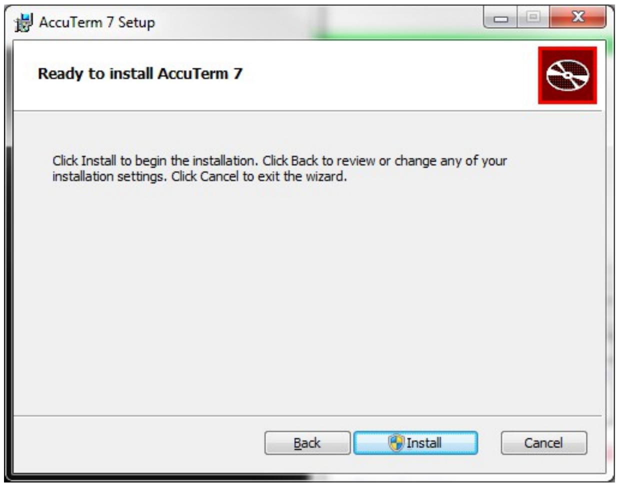
Click Next to install to the default folder or click Change to choose another.

Install AccuTerm 7 to:

C:\Program Files (x86)\Atwin71\

Change...
Create a shortcut for this program on the desktop
Enable custom features (code required)
Back
Next
Cancel

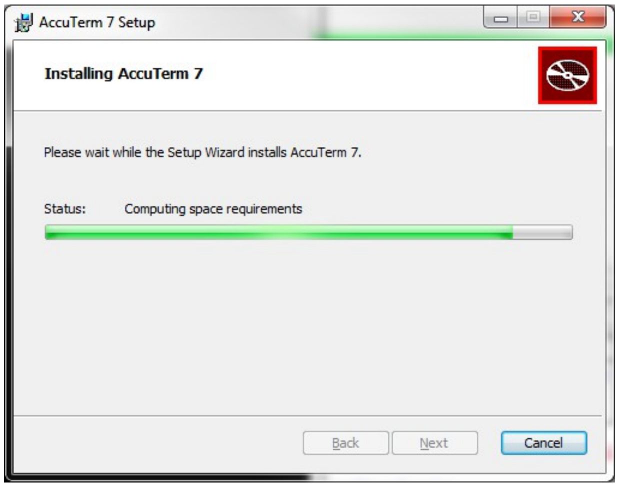
It is recommended that you accept the default destination directory. If you want, you can check the 'Create a shortcut for this program on the desktop'. Click on 'Next'.


AccuTerm 7 Setup

Ready to install AccuTerm 7

Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.

Click on 'Install'.

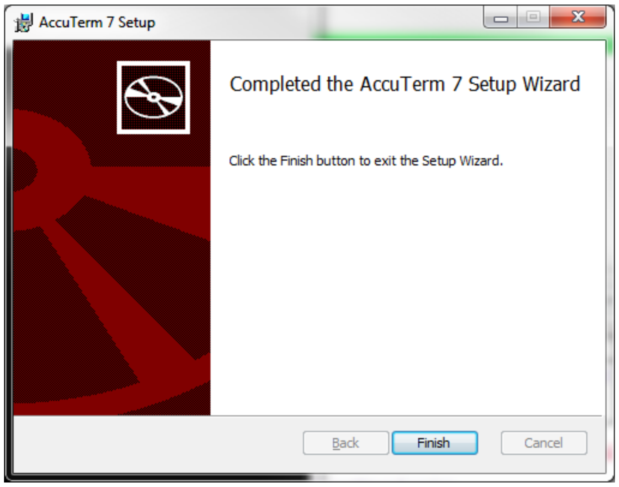

AccuTerm 7 Setup

Installing AccuTerm 7

Please wait while the Setup Wizard installs AccuTerm 7.

Status: Computing space requirements

The installation proceeds, and when complete, the following screen is shown:



Completed the AccuTerm 7 Setup Wizard

Click the Finish button to exit the Setup Wizard.

Click on 'Finish' to exit from the installation process.

11.5.3 Licensing

To enter your *AccuTerm* activation code, choose 'Help | Enter activation code' from the menu. Fill in the details, and click on the 'Activate' button.

11.5.4 Loading the AccuTerm multi-value host progams

You should start from the command prompt in the QMSYS account. If you aren't sure if you are in the QMSYS account, type **WHO** from the command prompt. This will respond with your line number and account name:

```
WHO
1 QMSYS
```

The first step is to create an account for the *AccuTerm* programs – substitute your own preferred account location for the path shown below (E:\QM\ACCUTERM):

```
CREATE.ACCOUNT ACCUTERM E:\QM\ACCUTERM
Create new directory for account (Y/N)? y
Creating VOC...
Creating $HOLD...
Creating $SAVEDLISTS...
Creating private catalogue directory...
Adding to register of accounts...
```

Now log to the account:

```
LOGTO ACCUTERM
```

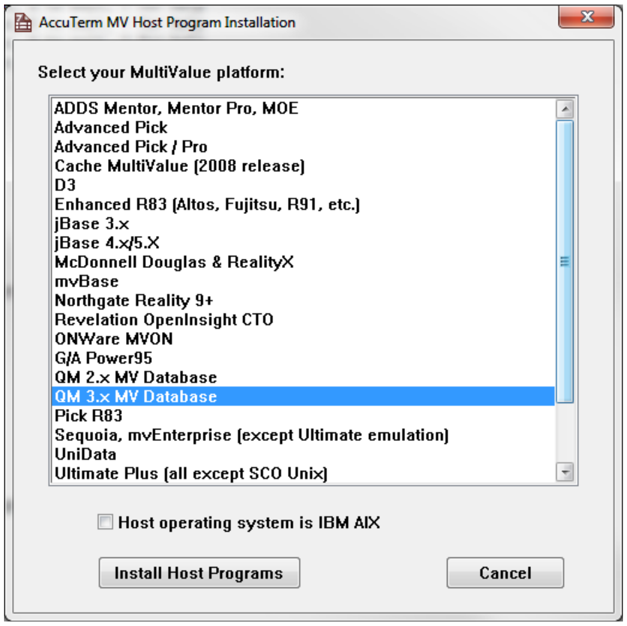
and check that case inversion is disabled:

```
PTERM DISPLAY
Break trapping: On  (char ^C)
Erase key: On  (char ^H)
Case inversion: Off
Input mark translation is off
Input newline:  CR
Output newline: CRLF
Binary mode : On
Prompt: ':' '::'
```

If case inversion is set to 'On', then type:

```
PTERM CASE NOINVERT
```

Now, select 'MultiValue | Host Programs | Install' from the *AccuTerm* menu bar, and click on 'Yes' in the dialog box that appears. Some text will scroll pass on the screen as *AccuTerm* assesses which multi-value database it is dealing with, then the following screen appears:



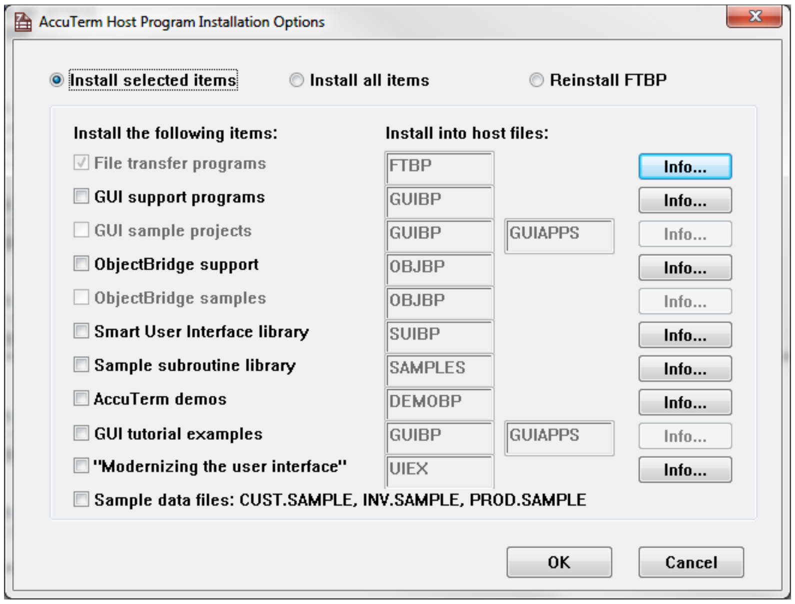
Select your MultiValue platform:

- ADDS Mentor, Mentor Pro, MOE
- Advanced Pick
- Advanced Pick / Pro
- Cache MultiValue (2008 release)
- D3
- Enhanced R83 (Altos, Fujitsu, R91, etc.)
- jBase 3.x
- jBase 4.x/5.X
- McDonnell Douglas & RealityX
- mvBase
- Northgate Reality 9+
- Revelation OpenInsight CTO
- ONWare MVON
- G/A Power95
- QM 2.x MV Database
- QM 3.x MV Database ✓
- Pick R83
- Sequoia, mvEnterprise (except Ultimate emulation)
- UniData
- Ultimate Plus (all except SCO Unix)

☐ Host operating system is IBM AIX

AccuTerm has correctly determined that this version 3.x of *OpenQM*. Click on the 'Install Host Programs' button.

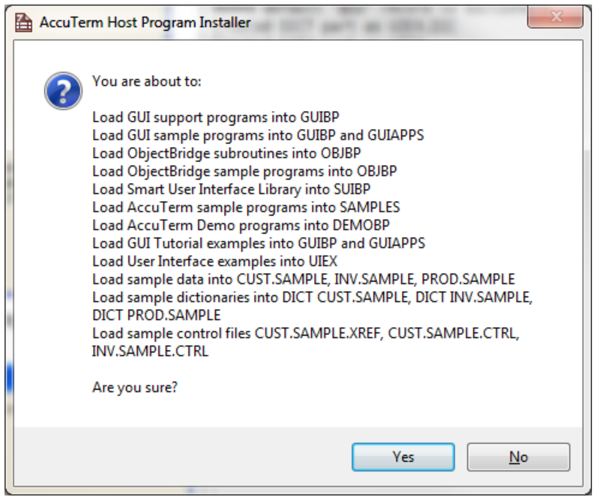
More text will scroll past on the screen as an installation program is loaded into the *AccuTerm* account. Once this is complete, the following options screen appears:



[✓ Install selected items] Install all items Reinstall FTBP

Install the following items:	Install into host files:		
✓ File transfer programs	FTBP		Info...
☐ GUI support programs	GUIBP		Info...
☐ GUI sample projects	GUIBP	GUIAPPS	Info...
☐ ObjectBridge support	OBJBP		Info...
☐ ObjectBridge samples	OBJBP		Info...
☐ Smart User Interface library	SUIBP		Info...
☐ Sample subroutine library	SAMPLES		Info...
☐ AccuTerm demos	DEMOBP		Info...
☐ GUI tutorial examples	GUIBP	GUIAPPS	Info...
☐ "Modernizing the user interface"	UIEX		Info...
☐ Sample data files: CUST.SAMPLE, INV.SAMPLE, PROD.SAMPLE			

Select option 'Install all items', and then click 'OK'. You are then prompted to create each file that does not exist in the ACCUTERM account. Finally, the following screen appears:

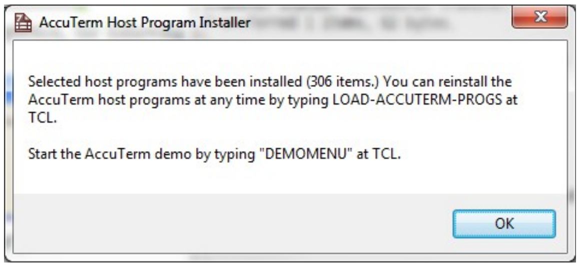

AccuTerm Host Program Installer

You are about to:

Load GUI support programs into GUIBP
Load GUI sample programs into GUIBP and GUIAPPS
Load ObjectBridge subroutines into OBJBP
Load ObjectBridge sample programs into OBJBP
Load Smart User Interface Library into SUIBP
Load AccuTerm sample programs into SAMPLES
Load AccuTerm Demo programs into DEMOBP
Load GUI Tutorial examples into GUIBP and GUIAPPS
Load User Interface examples into UIEX
Load sample data into CUST.SAMPLE, INV.SAMPLE, PROD.SAMPLE
Load sample dictionaries into DICT CUST.SAMPLE, DICT INV.SAMPLE, DICT PROD.SAMPLE
Load sample control files CUST.SAMPLE.XREF, CUST.SAMPLE.CTRL, INV.SAMPLE.CTRL

Are you sure?

Click on ‘Yes’, and the installation will continue. On completion, the following screen will be displayed:


AccuTerm Host Program Installer

Selected host programs have been installed (306 items.) You can reinstall the AccuTerm host programs at any time by typing LOAD-ACCUTERM-PROGS at TCL.

Start the AccuTerm demo by typing "DEMOMENU" at TCL.

Click on ‘OK’. You will be returned to the command prompt.

11.5.5 Configuring the AccuTerm Server Components

AccuTerm requires that the server name be set before some of its functionality will operate correctly. To do this, type **FTSETUP** from the command prompt:

```
FTSETUP


AccuTerm Account, File Transfer & Server Configuration


(1) Account Setup

(2) Kermit Configuration

(3) FT File Transfer Configuration

(4) FTD Data Transfer Configuration

(5) Pick-to-Pick File Transfer Configuration

(6) Server Configuration

Enter option (1-6) or X to exit:
```

Choose option 6:

```
Enter option (1-6) or X to exit: 6

Current parameter settings are:
    1. Host server name.............
    2. Allow server READ/READNEXT/REL Yes
    3. Allow server WRITE/ADD/DELETE. Yes
    4. Allow server CONV............ Yes
    5. Allow server EXECUTE/CALL..... Yes

Enter parameter number (1-5) to modify:
```

To enter the server name, type '1', and then enter the server name when prompted. The server name can take the form of either an actual name or an IP address:

```
Enter parameter number (1-5) to modify: 1

    1. Host server name?127.0.0.1
```

The server settings will be re-displayed, and the modification prompt repeated. Press enter to exit from this menu.

```
Enter parameter number (1-5) to modify:

Save configuration changes (Y/N) ? Y
Configuration saved...
```

The initial FTSETUP menu will now be re-displayed.

Before the *AccuTerm* utilities can be used in any account, that account needs to be activated from the *AccuTerm* account, using the FTSETUP program. At this stage, we only have two accounts in our *OpenQM* system – QMSYS and ACCUTERM. We should activate *AccuTerm* for use in the QMSYS account. To do this, choose option '1' from the FTSETUP menu:

```
Enter option (1-6) or X to exit: 1

This option will create the necessary file pointers
and copy the cataloged program items to the account
you specify.


Enter the name of the account you want to set up: QMSYS

Setup of QMSYS complete. Press <Enter> to continue
```

When you press enter, you will prompted for another account to setup. Press enter again and you will be returned to the FTSETUP main menu. Type **X** to exit from the menu.

11.5.6 Reboot

Theoretically, rebooting is not necessary. However, sometimes *AccuTerm* can't find some of its fonts without a reboot. So rebooting just takes care of any outstanding issues.

Log off from *OpenQM* by typing **OFF** at the command prompt. Then close *AccuTerm*, and reboot your computer normally.

12 Index