

QL EXPERT
(an expert system shell)

for the Sinclair QL Computer

USER GUIDE

by Francesco Balena

Compware QL Expert: Issue 1 December 7, 1987

© Copyright 1987 Francesco Balena

© Copyright 1987 Compware

All rights reserved

This documentation and the software to which it relates are copyright and must not be loaned or passed on to any third party, or modified, copied, stored or reproduced in any form or on any medium without the written consent of Compware, except for the making of working copies of the software by the purchaser. The purchaser may make up to five copies of the software for backup and for their own exclusive use on a single Sinclair QL or compatible computer. If the purchaser wishes to make further copies of the software for use on further computers, even if within the same establishment, then they must purchase further licenses (one for each additional computer) from Compware. Under no circumstances may the purchaser make copies of the documentation. It is the responsibility of the purchaser to ensure that the copyright is not infringed by preventing unauthorised copying of the software and documentation licensed to him.

Copyright Infringement

As a measure to discourage infringement of the copyright, each copy of *QL Expert* is encoded with a unique number ensuring that the origin of illegal copies can be traced.

Compware, 57 Repton Drive, Haslington, Crewe, CW1 1SA.

Sinclair, QDOS, & QL are registered trade marks of Sinclair Research Limited.

TABLE OF CONTENTS

- 1 Introduction
 - 1.1 How To Use This Manual
- 2 What You Get
- 3 Getting Started
 - 3.1 Making A Working Copy
 - 3.2 The BOOT Program
 - 3.3 QL Expert Installation
- 4 Knowledge Engineering With QL Expert
 - 4.1 What Is An Expert System?
 - 4.2 How Does An Expert System Work?
 - 4.3 Introducing QL Expert's Facilities
 - 4.3.1 Rules
 - 4.3.2 The Ordinary Define Rule
 - 4.3.3 The Selective Define Rule
 - 4.3.4 The Functional Define Rule
 - 4.3.5 Constructing Menus
 - 4.3.6 The Input Rule
 - 4.3.7 The Output Rule
 - 4.3.8 Goal Evaluation
 - 4.3.9 Providing Multiple Paths
 - 4.3.10 Special Input Options
 - 4.4 Designing An Expert System
 - 4.5 Programming Hints
 - 4.5.1 Building The Rule Base
 - 4.5.2 Conserving Memory
 - 4.5.3 Maximising Performance
 - 4.5.4 Debugging
 - 4.5.5 Preparing For Use
- 5 Creating And Interrogating A Rule Base
 - 5.1 QL Expert Commands And Menus
 - 5.1.1 On-Line Help
 - 5.1.2 QL Expert Menus
 - 5.2 Summary Of Editor Facilities

5.3	Examples Of Using The Editor
6	Rule Types And Programming Syntax
6.1	Kinds Of Rule
6.2	Goal Names
6.3	Goal Types
6.4	Notation Used In Syntax Definitions
6.5	Preliminary Definitions
6.6	Rule Types And Syntax
6.6.1	Ordinary Define Rule
6.6.2	Selective Define Rule
6.6.3	Functional Define Rule
6.6.4	Input Rule
6.6.5	Output Rule
6.6.6	PRINT, FPRINT And REPORT Statements
7	Special Functions
8	Error Messages
8.1	General Errors
8.2	Screen Editor Errors
8.3	I/O Errors
8.4	User Input Errors
8.5	Run-time Errors
8.6	Miscellaneous Errors
8.7	Informative Messages
9	Distributing QL Expert Rule Bases
10	What To Do If Things Go Wrong
11	Bibliography
Appendix	Problem Report Form

1. Introduction

QL Expert is a complete environment for the programming, debugging and operation of rule based expert systems. Although the author has produced a serious and very powerful tool, both the program and documentation have been designed on the assumption that many users will actually be experimenting with expert system design for the very first time. Experienced designers will be able to appreciate the power provided by *QL Expert*, but at the same time, the extensive descriptions and copious examples in the manual, and the superb on-line help should enable even the most inexperienced user to learn how to use this product to its full potential.

1.1. How To Use This Manual

The manual has been carefully designed to satisfy the needs of users with widely differing abilities. For the experienced programmer who finds things easy to pick up and remember, it provides clear and concise definitions of rule types, tables of commands for quick reference and a section explaining error messages.

For the user who is either new to this form of programming or finds it more difficult to get to know a new piece of software, there are separate sections explaining what an expert system is, how to go about designing one and how to use the different facilities provided by *QL Expert*, as well as step by step examples of using the editor to create a simple rule base.

All users should find the on-line help useful. It is accessed by pressing the F1 function key and quickly provides concise information about all aspects of *QL Expert*. The help is context sensitive, so for example if you are having a problem while using the editor, the help facility will automatically provide you with help relevant to the editor.

The manual is organised around the following main chapters:-

Chapter 3 - Getting Started

Explains how to back-up, configure and run the software. Many facilities can be customised using an installation program which is explained in this chapter, including the ability to set the default program and data drives for use from floppy discs and so on.

Chapter 4 - Knowledge Engineering With *QL Expert*

This is an introductory chapter which explains what an expert system is, what facilities are provided by *QL Expert* how to go about designing an expert system. It discusses each of *QL Expert's* facilities in turn and explains in simple terms how they are intended to be used. A collection of programming hints is also included.

Chapter 5 - Creating And Editing A Rule Base

This chapter begins by explaining the simple system of menus used to control *QL Expert* and includes tables summarising each menu and the functions of each menu command.

At the end of the chapter, it demonstrates how to use these commands by explaining the process of creating a rule base with step-by-step examples.

Chapter 6 - Rule Types And Programming Syntax

This is a reference chapter and contains the formal definitions of rule syntax and brief explanations of *QL Expert* constructs and how rules are evaluated.

2. What You Get

With this manual you should have received a single microdrive cartridge containing the following files:

- BOOT** - a simple program to automatically load *QL Expert*.
- EXPERT** - the expert system (compiled using *Digital Precision's Turbo* compiler).
- EXPERT_BIN** - some extended commands required by *QL Expert*.
- EXPERT_HOB** - the help file.
- EXTERT_DAT** - data required by the **INSTALL** program.
- INSTALL** - a program to modify the defaults used by *QL Expert*.
- BACKUP** - a simple to use general purpose backup program.
- CAR_EXP** - a rule base used to illustrate different rule types.
- HUMAN_EXP** - a rule base used to teach use of the editor.
- QLCON_EXP** - a simple rule base example.
- README_DOC** - a Quill file containing any updates or errata to the manual.

3. Getting Started

This chapter explains the very simple procedure necessary to start the *QL Expert* program. One of the golden rules of computing, usually learnt by bitter experience, is of course that before you try any program out, you should make a backup copy.

Please please don't just press on and use your master cartridge, make a backup copy by following the instructions of section 3.1.

Although you can use the working copy as it is, you can also modify the defaults used by *QL Expert* using the *INSTALL* program. This procedure is explained in section 3.3.

3.1. Making A Working Copy

Before you can get started, you will need to make a working copy of *QL Expert* on a new cartridge; the *BACKUP* program has been provided to help you do this. First, make sure you have a blank formatted microdrive to hand. Then, insert the *QL Expert* cartridge in *mdv1_* and type "lrun mdv1_backup". The backup program will load and ask you for the source device, so type "mdv1_". It will then ask for the name of the destination device, so insert your blank medium in *mdv2_* and type "mdv2_". The backup process will proceed, printing the name of each file as it is copied. When complete, remove the master and store it in a safe place.

3.2. The BOOT Program

The simplest way to load *QL Expert* is to use the program called *BOOT*. You can either type "lrun mdv1_boot", or auto-boot by inserting the cartridge in *mdv1_*, resetting your QL and pressing F1 or F2 as normal.

Once *QL Expert* has been loaded in this way, you can proceed to use all the facilities described in this manual. Chapter 5 provides an introduction to the editor, taking you step by step through creation of your first rule base. However, you will probably find it useful to read the general introduction to expert systems and their design in chapter 4 before attempting to create a rule base.

3.3. QL Expert Installation

This section explains how to configure various features of *QL Expert* including the default drive used for file access and the amount of memory for storing rules etc. Hence, if you want to use *QL Expert* with floppy discs, or make use of the memory available from a memory expansion, you should follow the procedure described below. The *INSTALL* program can also configure *QL Expert* to automatically load a particular rule base whenever the program is run.

Installation Procedure

- (1) Assuming that you have made a working copy of *QL Expert* on microdrive, insert this into *mdv1_*. If you have not already made a working copy, please do so. (It is silly to use the original cartridge, which is in fact protected and so cannot easily be written to.) Type *lrun mdv1_install*, and wait for the program to load.
- (2) You will be asked which drive holds the program overlays, so answer *mdv1_*.
- (3) Next, you will be asked if you want to install the program on a different drive. If you wish to use *QL Expert* from floppy discs, answer yes.
- (4) The next prompt gives you the option of modifying the data space used by *QL Expert*. If you have a memory expansion, you may wish to increase this. The default size is 8192 bytes which is enough room for about 100 rules. If for example, you had a 512K memory expansion, you might change the size to 500,000 which leaves a few kilobytes spare for use by the operating system. If you use a value larger than the available memory, *QL Expert* will use as much memory as possible. In practice, unless you really need to use all the available extra memory, it will be best not to allocate too much of your extra memory for use by the program in this way. The more memory you leave for use by the operating system, the better the performance of your QL will be when reading and writing files. If you want to leave the data space unchanged, just hit <ENTER> without giving a numeric value.
- (5) The number of levels of recursion used by *QL Expert* is the next parameter that can be modified. This determines the "depth" of the rule base that can be evaluated. One level of recursion is equivalent to evaluating a rule containing sub-goals that are evaluated entirely by asking questions of the user. Two levels of recursion permits a rule to contain sub-goals that can themselves be defined by rules. Three levels of recursion allows the sub-goals to be defined in terms of sub-sub-goals which are defined by rules and so on. The default number of levels of recursion is fifteen, and should not be increased carelessly, as it affects the amount of memory required. Again, you can leave it unchanged by just hitting <ENTER>.
- (6) The next question asks for the default device to be used for access to user data files (i.e. rule bases). If you are using microdrives, you will probably want this to be *mdv2_*. If you have a single floppy disc drive, you will probably want to use *flp1_*. Whatever the case, enter the name of the device required.
- (7) You can now alter the default device to be used for "logging" the details of a rule base interrogation session. (Logging is activated through the *design* command described in section 5.1.2.) This would normally be the device used for your printer, (e.g. *ser1*) or a file name. Just hitting <ENTER> prevents the use of a default, but will still allow you to select the device when you enable logging.
- (8) The next prompt is for the default sound status which will normally be "on". If you want sound to be "on" by default, type a "1". A "0" will cause the sound to be "off" by default. You can always alter the default temporarily using the *design* command.
- (9) The default border colour is the next setting requested, and is normally 7 (white). Type a number from 0 to 7 to select your preference. This might be used to help you distinguish between versions of *QL Expert* with different installation options.
- (10) If you want *QL Expert* to look for and load a default rule base whenever the program is run, you can set this with the next option. Simply type the name of the default rule base

(without a device name prefix), or enter a blank line if you don't wish to use this automatic start up feature.

- (11) If you responded earlier by indicating that you would like *QL Expert* to be installed onto a different medium, the next prompt will ask you to insert a blank formatted medium into the drive specified. When you press <ENTER>, the installation program will copy a set of *QL Expert* files to the new medium configuring as requested. If you did not request installation on a particular medium, then the files will be installed on your working copy immediately.

4. Knowledge Engineering With The Expert

This section offers a brief introduction to "knowledge engineering" and the kinds of task appropriate to an expert system.

Knowledge engineering is the programming of skills or experience into a special kind of database known as an expert system. Programming is done by knowledge engineers who construct a *rule base* that embodies the collective knowledge of experts in a particular field. The most striking difference between an expert system and an ordinary database is that the expert system is not passive, but will actively seek information which is most relevant to the particular circumstances which it finds.

4.1. What Is An Expert System?

An expert system is a combination of computer program and information database that mimics the processes which a human expert would go through in analysing a problem. That is, it actively gathers information about a problem, makes an analysis and presents its findings based on the information gathered.

For example, when diagnosing an illness, a doctor would usually be presented with a very limited amount of information, such as "it hurts here", and asks questions which will often result in a quick diagnosis. Each piece of information gathered may suggest new possibilities or tend to eliminate others and so contributes to the direction of the interrogation. Subsequent questioning will be designed to confirm or eliminate these possibilities, and will vary depending on the answers obtained. An expert system works in a similar way to this, although it is not capable of dealing with situations that are not covered by its "experience" in the way that a doctor might make intelligent deductions upon encountering an entirely new set of symptoms. This is because the doctor is capable of drawing on and applying a much wider base of experience than was originally provided by his formal training. He is able to learn from his own experience through dealing with patients.

In the case of an expert system, a set of *rules* are defined which correspond to the experience and training of human experts, and which enable it to mimic the activity of such experts. Whereas the doctor will be capable of a lot more "intelligent" tasks, such as playing chess or deciding which golf club to use, an expert system is generally limited to a very specific area.

Examples of the use of expert systems do in fact include medical diagnosis, where information about symptoms, medical history etc. is provided in response to questions asked by the system. Another well known example is their use in deciding where to drill for oil.

There are several reasons why expert systems have been developed in preference to employing human experts. In some cases, they may be cheaper because once an expert system knowledge base has been developed, it can be replicated for relatively little cost, reducing the need to use human experts. Other reasons are that an expert system may in fact be able to produce

"better" results than the humans whose expertise has been programmed into it. This might be because an expert system can combine the knowledge of several experts, or because it allows everyone access to the knowledge and experience of *the* best (human) expert available. Expert systems also have the advantage of being consistent, and not subject to moods or temporary blind spots. They are of course only as good as the information programmed into them.

4.2. How Does An Expert System Work?

Just as there are different programming languages, there are different ways of implementing an expert system, but there are some fundamental features that will be found in most current systems. Usually, such systems mimic the human approach to solving a problem.

Earlier, we noted how a doctor quickly identifies a number of likely diagnoses and proceeds to gather information designed to confirm or eliminate each one. In this way, he builds up an assessment of the relative likelihood of each, and can home in on the most likely for more thorough evaluation. Each possible diagnosis can be thought of as leading him to identify a number of subordinate hypotheses that need to be evaluated in order to assess each possible diagnosis. These subordinate hypotheses can be similarly broken down to provide a tree like structure of evaluations. By eliminating unlikely options early, the doctor need only thoroughly evaluate the most likely "trees" and can come to a decision very efficiently. This can be described as a *goal driven* approach because it involves the identification of goals and the gathering of information in order to assess the relative likelihoods of the different goals. How close it is to actual thought processes is unclear, but it is definitely a useful technique for automating some tasks which utilise human experts.

In an expert system, the tree of hypotheses is replaced by a tree of rules which define how to evaluate goals. The expert system tries to determine the "value" of a goal, using the structure of the rule base and the values of other goals in the system. Ultimately, these values all depend on replies made by a user to questions put by the expert system. How rules are defined and evaluated is very dependent on the implementation of the expert system "shell" (i.e. the tool used for building expert systems - e.g. *QL Expert*).

In operation, the expert system merely functions as a computer program. It does not reason in the way that a doctor might, and will fail when it encounters a situation not envisaged during its design. In fact, there is nothing to prevent an expert system from being written in a conventional programming language, but to do so would be very inefficient because a conventional language does not provide facilities to simplify the task. In some ways, this is analogous to programming in assembly language instead of using a high level language such as Superbasic or Pascal. Assembly language may provide added flexibility, but this may not be required and will be achieved at the expense of vastly reduced productivity when applied to a complex task.

As techniques improve, expert systems will no doubt become capable of learning more from experience and, through reasoning, to cope with novel situations. For the time being though, they are not as mystical or intelligent as much of the hype has lead people to believe. They are useful and extremely interesting tools, but have well defined limitations.

The process of extracting the knowledge required to apply an expert system to a particular problem and building it into the rule base, is termed "knowledge engineering". The next section introduces the facilities provided by *QL Expert* to help the knowledge engineer with this task.

4.3. Introducing QL Expert's Facilities

In section 4.2, we described the *goal driven* approach to problem solving and said how this was a useful technique for automating some such tasks. *QL Expert* is designed to make use of this powerful but relatively easily understood method. It therefore provides facilities for defining rules which describe how goals should be evaluated, and enables rules to be defined in terms of the results of sub-goals. In addition, a large number of features have been designed to allow a high degree of freedom in the implementation of the goal driven system.

4.3.1. Rules

The following sections introduce the different kinds of rule supported by *QL Expert* and explain how they are intended to be used. They provide a discursive introduction and explanation of facilities, most of which are given a more formal definition elsewhere in the manual. No consideration is given to the design of a rule base, i.e. to how to implement a goal driven approach. Design is dealt with in a subsequent section and so here we consider only the facilities provided and how they can be used, rather than how to arrive at a useful definition for a particular rule.

4.3.2. The Ordinary Define Rule

The following represents a typical example of the *ordinary* kind of *define* rule:-

```
RULE# 10
  DEFINE Riesling
    Want dry white
    Want inexpensive
  OR
    Want medium white
    Want inexpensive
```

The first line indicates that this is rule number 10. The rule number is used just like line numbers in basic. It gives each rule a unique reference (from 1 to 65535) and defines the order in which rules will appear in listings of the rule base. However, unlike line numbers in a basic program, it does not determine the order of evaluation of rules which is dependent entirely on the logical structure of the rule base and not on the order in which rules appear.

The example shown above might be taken from a rule base designed to help choose a kind of wine. This could be done by evaluating the relative merits of each type of wine that is available and then selecting the one which scores most highly. The rule given could therefore be for determining the score of Rieslings.

The lines "Want dry white", "Want inexpensive" etc. refer to goals whose value is used in this rule to determine the value of "Riesling". These (sub) goals will themselves be defined by similar rules. "OR" is a keyword, not a goal name, and indicates a logical OR operation. The other goal values are implicitly AND'ed in this kind of rule which is equivalent to the following "sentence":-

```
"Riesling is..
(Want dry white AND Want inexpensive) OR (Want medium white AND Inexpensive)"
```

Hence, if the first two goals are *true* or the second two goals are *true* then the value of "Riesling" will also be *true*, otherwise "Riesling" will be *false*. This assumes that goals are either

true or *false* and is known as *boolean* logic. However, in the majority of situations, goals are unlikely to be so clearly defined and are more likely to have degrees of certainty associated with them. For example, you may be pretty sure that you would like an inexpensive wine, but as it's for a special occasion may not want to entirely rule out more expensive possibilities. Hence it would be more appropriate to assign some form of numeric "rating" to the goal "Want inexpensive" rather than to make it absolutely *true* or *false*. One way of achieving this is by using *fuzzy logic* as opposed to *boolean logic*. *QL Expert* supports both kinds of logic by always allowing goals to be given values from 0 to 1. A value of 0 is equivalent to *false* and 1 is equivalent to *true*. Intermediate values represent the *fuzzy* states which can be thought of as probability values.

Assuming that all goals evaluate to a numeric value and that none are undefined, the result of an (implicit) AND operation will be taken as the smallest of the values being AND'ed, and the result of an OR operation will be the largest of the values being OR'ed. This is fully compatible with the boolean treatment of *true* and *false* values (represented as 1 and 0 respectively). See the description of the *ordinary define rule* in section 6.6.1 for details of what happens when *undefined* values are encountered. (Note however that other kinds of rule can be used to define your own relationships between (sub) goal and goal values and so you are not restricted to this particular implementation of fuzzy logic.)

In addition, a goal name in the above kind of rule can be prefixed by the NOT keyword as in

NOT Want dry white

which has the effect of negating the value of the given goal. Hence in place of the value of "Want dry white" a value of "1 minus Want dry white" will be used. Therefore, "NOT *true*" would become *false* and a value of "NOT 0.35" would effectively become $1 - 0.35 = 0.65$.

Another feature of this kind of rule is the ability to apply weights to goal values using a FACTOR statement as in the following example:-

```
RULE# 15
  DEFINE Want dry white
  FACTOR 0.75
  ... etc.
```

The FACTOR statement must appear before the main body of the rule, but after a LABEL statement (see later) if present. When the value of the goal "Dry white wine" has been evaluated in the normal way, it will be multiplied by the factor of 0.75 before being assigned to the goal. Hence the above rule can never produce a value of "Dry white wine" greater than 0.75.

Another keyword can be used to provide similar effects, but also has added flexibility. This is EVAL and enables an expression to be used wherever a goal name might appear. For example:-

RULE# 25

```
DEFINE Riesling
EVAL 0.75*P(Want_dry_white)
EVAL 0.63*P(Want_inexpensive)
OR
Want medium white
Want inexpensive
```

P() is a function that returns the numeric value of the goal matching the string enclosed. Although not very meaningful, the above example shows how weightings can be applied using EVAL. Since EVAL is intended to evaluate a probabilistic type expression, which should always have a value from 0 to 1, an EVAL statement will generate a run time error if its expression produces a result outside this range. In practice, very much more complicated expressions can be used as in the following:-

```
EVAL .5*(P(Dice_roll)+P(Num_dice))/P(Required_roll)
```

Note that *QL Expert's* special functions such as BINOM(), EXP() etc. cannot be used in an EVAL statement. If you wish to use them to generate more complex relationships, you should use the *functional* form of the *define* rule which is described shortly.

One more feature of the *ordinary* define rule is the ability to test the state of a goal which has a string typed value. For example, you may know the colour of a wine and be trying to discover which region it comes from:-

RULE# 120

```
DEFINE Is Claret
Colour=Red
Full bodied
NOT Sparkling
... etc.
```

RULE# 130

```
INPUT Colour
PROMPT What is the colour of the wine
OPTIONS Red,White,Rose
```

The first line of rule 120 will evaluate to *true* only if rule 130 resulted in "Colour" being given the string value "Red". (Rule 130 is an example of an *input* rule which is described in a later section. As shown, it allows the user to assign a value of "Red", "White" or "Rose" to "Colour" or to leave "Colour" undefined.)

Note: In the case where the string valued goal being tested has multiple values (see next section), the above test will still succeed providing that the string matches one of the multiple values held by the multi-valued goal.

4.3.3. The Selective Define Rule

The next example is of a *selective define* rule:-

RULE# 20

```
DEFINE Wine selection
MAX OF Riesling,Burgundy,Chianti
```

This example can be thought of as being at a higher level in the logical structure of the rule base than the previous one because it makes use of the goal "Riesling". The other goals, "Burgundy" and "Chianti", could be defined by similar rules to that for "Riesling".

This is a very different kind of rule and in fact results in a text string rather than a numeric value, being assigned to the goal "Wine selection". Providing that the rule is successfully evaluated, the string assigned will be the name of the (sub) goal in the MAX OF statement which evaluates to the highest value. Hence if "Burgundy" achieves the highest score, the value of "Wine selection" will become "Burgundy".

Such a rule is very useful for causing a list of possible outcomes or diagnoses to be evaluated and provides for alternative selection criteria in addition to the MAXimum OF a list of goals. In place of MAX OF, you can also use MIN OF, FIRST OF or ALL OF.

MIN OF is similar to MAX OF except that name of the goal with the smallest value is selected.

FIRST OF selects the name of the first goal which is evaluated successfully, i.e. is not *undefined* or *false* (zero).

ALL OF is a very special case and assigns multiple values to the goal concerned. The result will include the names of all goals which have values greater than zero. Hence a goal defined using ALL OF can have several values (simultaneously). Later you will see how this can be used (with an *output*) rule to present a list of goal names and their relative merits instead of a single result.

In addition, a CONDITION statement can be used within a *selective define* rule to impose an alternative selection criterion. Normally, the criterion used to select a goal is that it has a non-zero value. The following condition statements show how this can be modified:-

```
CONDITION >0      (is equivalent to the default criterion)

CONDITION >0.5     (selects goals with values greater than 0.5)

CONDITION <=.25    (selects goals with values less than or equal to 0.25)
```

Only one CONDITION statement may be present in a rule and it must be the statement immediately before the MAX/MIN/FIRST/ALL OF statement.

4.3.4. The Functional Define Rule

The third and final kind of *define* rule is the *functional define* rule shown below:-

```
RULE# 30
DEFINE Want inexpensive
RETURN .4*Is_five_pounds_too_much+.6*Is_ten_pounds_too_much
```

This kind of rule is similar to a function definition in conventional programming language such as Pascal or Superbasic. A simple mathematical expression is used to determine the result and,

as in the above example, the expression can include goal values. Note that spaces in the names of goals included in such an expression must be replaced by underline characters.

This rule type enables you to define alternative methods of determining the value of a goal instead of the way used in the *ordinary define* rule described earlier.

The expression can use the following operators in the same way that Superbasic would allow them to be used:-

$+$, $-$, $*$, $/$, $^$
($^$ means to the power of)

To help with constructing complex relationships, the following functions (described fully in chapter 7) may also be included in the expression:-

ABS(), BINOM(), EXP(), FACT(), INT(), LN(), LOG(), MAX(), MIN(), MOD(), PO,
ROUND(), SQRT() and TEST()

Conditional operators such as

$<$, $>$, $<=$, $>=$, $=$, $==$
($==$ means approximately equal to, as in Superbasic)

can also be used as in the following example which returns *true* (i.e. 1) if the value of "year" is one of 1971, 1975, 1978 or 1982:-

```
RULE# 106
  DEFINE Good year Chianti Classico
  RETURN (year=1971)|(year=1975)|(year=1978)|(year=1982)
```

Note, 'I' is an additional operator where

xly is equivalent to $x + y - x*y$.

Note that where the operands are *boolean*, its effect is identical to a logical OR operation.

In addition, this kind of rule permits the use of IF statements. This enables the returned value to be made conditional on the result of a comparative expression. For example:-

```
RULE# 40
  DEFINE Want inexpensive
  IF How_much_do_you_want_to_spend<5
  RETURN 1
  IF How_much_do_you_want_to_spend<10
  RETURN .5
  RETURN 0
```

The above rule can provide one of three values to be assigned to "Want inexpensive" depending on the value of the goal "How much do you want to spend". If the latter is less than five (pounds) the return will be 1 (*true*), if it is greater than five but less than ten the return will be .5 and if the value is greater than ten the return will be 0 (*false*).

(Note that IF statements cannot be nested - i.e. consecutive IF statements are not allowed.)

IF statements can also be used to increase the sophistication of the rule base as in the following example:-

RULE# 50

```
DEFINE White wine is appropriate
IF TEST(Do_you_want_a_white_wine)
RETURN Do_you_want_a_white_wine
RETURN Is_white_wine_appropriate
```

RULE# 60

```
DEFINE Is white wine appropriate
Is the drink an aperitif
OR
To be eaten with fish
OR
NOT To be eaten with red meat
```

When *QL Expert* comes to evaluate the goal "White wine is appropriate" using rule 50, it will first evaluate "Do you want a white wine" which in this case would be the result of asking the user this question. An *input* rule (see later) would be used to restrict his replies to "yes" (i.e. *true*), "no" (*false*) or *undefined*. If the user did not know the answer, and hence replied with "undefined", the first RETURN statement would be ignored and the second one used to determine the result, causing evaluation of the goal "Is white wine appropriate" which requires more detailed analysis by *QL Expert*. Hence the rule base can be constructed so that it takes short cuts when the user can provide a value for a goal, but can also use alternatives when the user is unable to help.

The TEST() function returns *true* unless the goal specified is *undefined*.

4.3.5. Constructing Menus

QL Expert makes it possible to construct a menu of goals from which a user can select a goal to be evaluated. To generate a menu entry for a goal, you need to include a LABEL statement in a rule used to define that goal. The LABEL statement must always appear immediately after the line containing the word DEFINE as in the following examples:-

RULE# 35

```
DEFINE Select wine
LABEL Select a suitable wine for a meal
... etc.
```

RULE# 300

```
DEFINE Car purchase
LABEL Advice on whether or not to buy a car
... etc.
```

RULE# 555

```
DEFINE Car type
LABEL Advice on which kind of car to buy
... etc
```

When the user presses function key F2, *QL Expert* searches the rule base for LABEL statements and uses the text after the LABEL keyword to produce a menu of options. A rule base containing the above LABEL statements would produce the following menu:-

- A - Select a suitable wine for a meal
- B - Advice on whether or not to buy a car
- C - Advice on which kind of car to buy

The user can then select which goal is to be evaluated by pressing either A, B or C. Pressing A would cause evaluation of "Select wine" and so on.

4.3.6. The Input Rule

At some point in the evaluation of a goal, it will be necessary to obtain input from the user. *Input* rules are used to provide prompts for such input and to ensure that only valid replies are accepted. During development of the rule base, *QL Expert* will use a default form of *input* rule to obtain the values of any goals which do not have rules. This makes it very easy to test partially complete rule bases, but makes it possible to cause run-time errors by providing out-of-range values. Hence in the finished rule base, all input obtained from the user should be controlled by appropriate *input* rules.

An *input* rule consists of a PROMPT statement and an OPTIONS statement, although the latter can be omitted. For example:-

```
RULE# 610
  INPUT Wine colour
  PROMPT What colour wine would you prefer
  OPTIONS red,white,rose
```

Which will prompt with "What colour wine would you prefer?" and requires the user to answer 'red', 'white' or 'rose'. If an invalid response is given, a list of acceptable replies will be printed and the question repeated.

The OPTIONS statement can include text strings (as above), lists of numbers and conditional statements such as >5 or ≤ 0.3 . Examples:-

```
OPTIONS red,white,rose
```

Allows only the strings "red", "white" or "rose" to be accepted.

```
OPTIONS  $\geq 0, \leq 1$ 
```

Allows any number in the range 0 to 1 to be input. Alternatively, this could have been written as follows:

```
OPTIONS %
```

The '%' is a shorthand way of specifying any number from 0 to 1. Similarly, '#' means any valid number, '\$' means any valid string and '&' means any boolean value (i.e. yes, no, true or false). Note that "any valid string" means any string excluding reserved strings such as "yes", "no", "trace", "why" etc. which have other meanings when given as input. You can however include these strings in an OPTIONS statement in which case they will be accepted as straightforward inputs, overriding their reserved functions. The following statement will allow absolutely any input without restriction:

OPTIONS \$, #, &

The final example accepts any number in the range 0 to 1, or greater than 50, or the string "Quantum Leap" as input:

OPTIONS >=0,<=1,>50,"Quantum Leap"

4.3.7. The Output Rule

The fifth and final kind of rule supported by *QL Expert* is the *output* rule. When *QL Expert* succeeds in evaluating the top-level goal requested by the user, it presents the result which may be a string or numeric value, or may be undefined if the evaluation was inconclusive. If an *output* rule is not provided for a particular goal, a simple statement will be produced by default which shows the name of the goal evaluated and its value. However, this is not very informative and *output* rules provide a way of presenting the results in a more sophisticated way. A simple example is shown below:-

RULE# 355

OUTPUT Select wine

PRINT The most appropriate wine is a @

PRINT with a rating of %%

After evaluating the goal "Select wine" the above rule will be found and used to produce output similar to:

The most appropriate wine is a Burgundy
with a rating of 78%

Note that the '@' has been replaced by the value of the goal "Select wine", which in this case was a string ("Burgundy") but in other situations could have been a number between 0 and 1. Where the goal has a string type value, '##' will be replaced by the numeric value associated with the goal whose name matches that string. The '%%' used above has a similar effect but shows this as a percentage instead of the actual value which would be a number between 0 and 1.

Note that *QL Expert* will normally only use an *output* rule for the goal that the user selected for evaluation. Hence if "Select wine" were evaluated as part of evaluating a higher level goal, its *output* rule would be ignored and no output produced when "Select wine" is evaluated. One exception to this is when the "trace" facility is enabled (see section 4.3.9). In this case, an *output* rule will be used for any goal whose value is being printed if such a rule exists.

Another more subtle exception is when the top level goal is defined by a *selective define* rule. For example:-

RULE# 20

DEFINE Wine selection

MAX OF Riesling,Burgundy,Chianti

If *output* rules have been defined for the goals "Riesling", "Burgundy" and "Chianti", then they will be executed when all goals in the "OF" statement have been evaluated. In practice, the value of the top level goal will be presented first, followed by that for any sub-goals for which *output* rules have been defined. Note however that it is possible to suppress the output of information about the top level goal altogether, by providing it with a dummy *output* rule

which does not contain any PRINT, REPORT or FPRINT statements.

The above is a powerful mechanism for causing a list of goals to be evaluated in one session, followed by presentation of the results specified by a different *output* rule for each goal. Think about how this can be used with the "FIRST OF" construct which terminates evaluation of the goal list as soon as one is successfully evaluated (i.e. not undefined). It makes it possible to go through a list of possible "answers" to a problem and to produce output which will be entirely dependent on the "answer" required.

A very useful feature when evaluating multi-valued goals (i.e. goals using a *selective* rule with an ALL OF construct) is that any lines in an *output* rule containing substitution characters (@, ## or %%) will be repeated for each of the goals values. For example, the following combination of rules

```
RULE# 100
    DEFINE Select wine
    ALL OF Burgundy,Claret,Riesling,Chianti

RULE# 340
    OUTPUT Select wine
    PRINT The wines considered are rated as follows
    PRINT  @ score %%
```

would produce output similar to

```
The wines considered are rated as follows
    Burgundy score 35%
    Claret score 45%
    Riesling score 65%
    Chianti 84%
```

A backslash character can be used to precede a word that is only to be printed if the goal evaluated is *false*, as in

```
RULE# 123
    OUTPUT Riesling
    PRINT A Riesling is \t suitable
```

IF statements can be included in *output* rules to make the output conditional on the state of certain goals in the rule base.

A REPORT statement can be used to improve this facility by enabling output to be truncated at a given point. REPORT acts in exactly the same way as PRINT except that it forces termination of the *output* rule. For example:

RULE# 245

```
OUTPUT Select wine
IF P(Riesling)=1
REPORT You should definitely select a Riesling
IF P(Chianti)=1
REPORT You should definitely select a Chianti
PRINT The results were not absolutely definitive
PRINT and you should choose from the following
PRINT @ which scored %%
```

If either "Riesling" or "Chianti" equal 1, only the corresponding REPORT statement will be executed. Otherwise, the REPORT statements will be skipped and only the PRINT statements will be used. If PRINT statements were used in place of REPORT, the last three PRINT statements would always be executed.

Note that if a goal used in an IF statement has not yet been evaluated the statement following will always be evaluated.

Another statement type, FPRINT, is also allowed within an *output* rule. This is similar to PRINT, but prints text which has been stored in a file. You can simply print the file, or specify a position in the file from which to start printing. For example

```
FPRINT mdv1_wine_results
```

Would print the text held in file *mdv1_wine_results* until an ESC (decimal 27 or CHR\$(27)) character is found or the end of the file is reached. Note that substitution characters (##, %% and @) have no effect when included in a file. You can freely mix PRINT and FPRINT within an *output* rule.

To print a file starting from a given character position you could use

```
FPRINT flp1_wine_results,355
```

which would start displaying text at the 356th character. Alternatively, a negative number can be used to display starting at a particular *page* within a file, as in

```
FPRINT wine_results,-4
```

which would read through the file until the third ESC character and then start displaying text. Page -1 is the start of the file, page -2 means after the first ESC etc. Note that the device name need not be specified as part of the file name, in which case the default device will be searched.

FPRINT displays text 20 lines at a time, prompting the user to press the space bar to continue. In all cases, once FPRINT has started printing, it stops displaying text when it encounters either an ESC character or the end of the file. (Note that lines of text should not exceed 76 characters in length.)

4.3.8. Goal Evaluation

This section explains how *QL Expert* evaluates goals.

You can ask for a particular goal to be evaluated by selecting an option from a menu or simply by typing its name. The latter method is very useful for debugging because it allows any goal in the rule base to be evaluated at will. Note also that pressing F4 causes the last evaluated goal to be evaluated again.

The first thing that *QL Expert* does when it is asked to evaluate a goal as above, is to clear its current knowledge about goal values that may be left over from previous evaluations. This prevents previous sessions from affecting the outcome of the current session. (Note that when debugging it may be useful to suppress this clearing of information and this can be achieved by typing a space or comma immediately before the name of the goal that you wish to evaluate.)

Next, *QL Expert* searches for a rule which defines a method of evaluating the goal specified. When one is found, it suspends the search and starts to evaluate the goal according to the rule. If the evaluation results in the goal not being *true*, the search will be resumed for another rule defining this goal and the process is repeated until a rule produces a result of *true* or no more rules are available which define this goal. The results of these evaluations are then combined to produce the value of this goal. They are OR'ed together in exactly the same way that sub-goal values are combined in an *ordinary define* rule (see section 6.6.1 for full details).

The above procedure is not just applied to the top level goal selected by the user, but is applied to the evaluation of all goals.

When the value of the top level goal has been determined, *QL Expert* searches for an *output* rule for this goal, which if found is used to present the results. If not, a simple default form of *output* rule is used instead.

During the evaluation of a rule, *QL Expert* will usually encounter further goals which must also be evaluated. Each of these is evaluated in exactly the same way as described above, by searching the rule base for rules concerning the current goal and evaluating them. Hence this is a recursive process, and continues until the rule found is an *input* rule, or no rule can be found for the goal concerned.

If an *input* rule is found, the value of the goal will be obtained from the user's input according to the rule. If no rule is found, then *QL Expert* uses a default form of *input* rule and automatically prompts the user (with the goal's name) for a value. (The valid inputs for this default *input* rule can be set using the *DESIGN* command described in section 5.1.2.)

As each goal is assigned a value, it can be used in the evaluation of higher level goals until finally, the value of the goal requested for evaluation has been determined.

QL Expert uses several techniques to improve the efficiency of goal evaluation. Wherever the value of a goal can be determined without the need to evaluate all statements in a rule, it does so. An example of this is where an *ordinary define* rule contains several implicit AND operations and one of the operands (goals) evaluates to *false*. Once this happens, the result of the AND's must be *false* regardless of the values of subsequent operands and evaluation can stop. Similarly, once a goal has been evaluated during evaluation of one rule, its value will automatically be used without re-evaluation if it is subsequently encountered in another rule. This simple pre-scanning can eliminate a lot of redundant questioning of the user and makes any rule base far more effective.

QL Expert also "pre-compiles" expressions as would be done by a conventional compiler increasing the speed of evaluation at run-time.

4.3.9. Providing Multiple Paths

An important feature of any well designed rule base is the provision of multiple paths for the evaluation of particular goals. Thus, if one path produces an undefined result, an alternative can be tried. They are important because they make it possible to prevent goals with undefined

values propagating through the rule base forcing an inconclusive result. In *QL Expert* the *selective define* rule provides a simple mechanism for providing multiple evaluation paths as described below.

The following is a very simple example of a rule base without a multiple path for the goal called "Want inexpensive"

```
RULE# 100
    DEFINE Riesling
    Want inexpensive
    Want white wine

RULE# 110
    DEFINE Want inexpensive
    NOT Meal is special occasion
    OR
    Finance is restricted
    OR
    Want to make good impression
```

The above provides only one path for evaluating the goal "Want inexpensive" which can therefore easily become undefined and will probably cause the higher level goal "Riesling" to become undefined and so on.

The *selective define* rule can be used to alter this rule base to provide several paths as follows:-

```
RULE# 100
    DEFINE Riesling
    Want inexpensive
    Want white wine

RULE# 110
    DEFINE Want inexpensive
    RETURN P(Want inexpensive mpath)

RULE# 120
    DEFINE Want inexpensive mpath
    FIRST OF Want inex path1, Want inex path2
    Want inex path3, Want inex path4

RULE# 130
    DEFINE Want inex path1
    ... etc.

RULE# 140
    DEFINE Want inex path2
    ... etc.

    ... etc.
```


Rule 110 is a dummy rule which removes the need to replace all occurrences of "Want inexpensive" in other rules with "P(Want inexpensive mpath)". Rule 120 is the *selective* rule that evaluates each of the sub-goals listed in turn, until one produces a result. Only if all these sub-goals are undefined will "Want inexpensive mpath" and hence "Want inexpensive" become undefined.

The rules 130, 140 etc. provide the redundant paths. You must be careful in the design of these redundant paths to make them as independent as possible. If for example, they were all dependent on a common sub-goal which was itself undefined, the redundant paths will be useless since they will all be undefined.

4.3.10. Special Input Options

During an interrogation, a number of special options are available to the user at any prompt for input. These will be listed on just pressing <ENTER> and include the following:-

Special Input Options	
Option	Description
value ?	(I.e. A valid answer followed by a question mark.) This is a way of asking a WHAT IF type question. The interrogation will proceed as if you had just answered with the value given until you reply to an input with "exit". This causes the evaluation to return to the point where "value ?" was given and enables you to try several different replies to a given prompt. Up to six WHAT IFs may be active at any one time, and the most recently invoked WHAT IF will be shown on the top line of the display.
EXIT	Exit a WHAT IF path and return to the original input prompt. (See above.)
VALUES	Causes <i>QL Expert</i> to print out the values of all goals that have been evaluated so far. If the secondary output device is active (see PRINTER below), this will also be listed on it.
HOW goal_name	Causes all rules related to the evaluation of the given goal to be listed to the screen (and secondary output device if active).
PRINTER	Activates or deactivates the secondary output device. When active, all information printed on the screen will also be sent to the secondary output device which can be a printer or file. See the PRINTER command from the DESIGN menu in section 5.1.2 for details of how to select the device to be used.
QUIT	Quits the current interrogation and returns to the default mode of <i>QL Expert</i> .
TRACE	Causes the values of goals to be printed as they are evaluated.
UNDEFINED	Assigns the value <i>undefined</i> to the goal being prompted for.
WHY	When used the first time will print out the rule which contains the statement causing this particular request for input. At each subsequent response of WHY, the next highest rule in the current evaluation path will be printed (i.e. the rule that caused the rule previously printed to be evaluated).

Some of the above are of general use during any interrogation, but all are of particular value to the programmer when developing and testing a rule base. They make *QL Expert* very powerful by simplifying the task of creating and testing highly complex rule bases.

4.4. Designing An Expert System

The same principles of top-down design that apply to conventional software development apply equally well to the design of an expert system rule base. In conventional top-down software design the idea is that you start by defining the main functions of a system in a fairly abstract way and gradually decompose each function into a small number (typically less than seven) of equally easily expressed functions. Then you take these functions and apply the same process until the entire program can be described as a tree like structure containing a large number of relatively simple functions. This technique reduces the degree of complexity that needs to be dealt with at any stage and so makes it relatively easy to implement the small functional blocks using procedure and function calls.

In a rule base, goals take the place of functions and so you should start by identifying the top level goals that you wish to evaluate and proceed to identify the sub-goals that will need to be evaluated in order to determine the values of the top-level goals. Again the process can be continued recursively until a tree-like goal structure has been defined. Until this has been done, you should not have worried at all about how sub-goals will be used to evaluate goals (i.e. about the implementation of particular rules) and should concentrate entirely on the logical structure of the rule base - not its implementation.

One feature of good rule base design not required when designing conventional software, is the provision of multiple paths for the evaluation of goals. (See section 4.3.9 for an explanation of how to implement multiple paths.) Without this, a single undefined goal can propagate back through the rule base forcing an inconclusive result, as explained earlier. The best point at which to start considering the provision of multiple paths is after having established the logical structure of goals and sub-goals that are required. When this has been done, it should not be difficult to identify particular goals for which one or more alternative paths can most easily be provided. Note that for this to be most effective, you should distribute multiple paths evenly throughout all levels of the rule base, reducing the maximum depth (in rules) over which no multiple path is provided. Having identified the goals for which multiple paths can be most easily provided, you can then look for weak points in the rule base where excessively deep "trees" exist without alternative paths.

Note that there is little point in providing alternative paths for evaluating a particular goal if all such paths depend on the same input data or on common goals. Hence you should try to avoid re-using input data in alternative paths - this is contrary to the analogy with conventional software design which encourages re-use of low level functions.

Only once the tree structure has been finalised and suitable sets of duplicate (or even triplicate etc.) sub-trees have been specified, should you start to define the actual implementation of particular rules.

At this stage, you may find that the most appropriate logical structure leads to some goals requiring particularly complex or large rule definitions. Where this happens, dummy rules should be introduced which evaluate dummy goals and reduce rule size and complexity.

4.5. Programming Hints

The following sections list a few useful tips to help get the most out of *QL Expert* and to make programming and testing simpler.

4.5.1. Building The Rule Base

Complex rule bases should be designed using the techniques described in the previous section, and should be written down on paper before being entered into *QL Expert*. When you come to construct the rule base, you should enter and test one rule at a time. Start by entering the top-level rule and causing it to be evaluated. You can then verify its performance by manually providing values for the goals required to evaluate it. When tested, you can enter a rule defining how to evaluate one of the goals used by the top-level rule, and test this in the same way. When all the goals contributing to the top-level goal definition have been entered and tested you can proceed to add rules to define lower and lower level goals until the entire rule base has been constructed - and of course tested. Once this has been done you can add *input* rules to make the prompts more explicit and to trap input errors. Similarly, *output* rules can be added to make the output more informative.

Note that you will probably need to alter the default input options (to "\$,%,&") to allow input of *string* values, in addition to *boolean* and *uncertainty* values.

4.5.2. Conserving Memory

On an unexpanded QL, *QL Expert* has about 10K bytes of memory available for storing rules which is enough for about 100 medium sized rules. Each rule requires five bytes of storage, plus one byte per keyword, plus one byte per character in goal names and text strings, plus a few characters to store expressions (which are stored in compiled form).

Hence it is possible to significantly reduce the space required to store a rule by keeping goal names and text strings concise. Unless memory is desperately short however, you should not shorten names and descriptions at the expense of clarity. Remember that goal names serve a dual function, and are used as a form of comment as well as to express the logical structure of the rule base.

Note that you need to customise *QL Expert* using the INSTALL program if you want it to make use of expanded memory (see chapter 3).

4.5.3. Maximising Performance

QL Expert performs a linear search, starting with the lowest numbered rule, when trying to find a rule for a given goal. You should therefore try to position the rules which are most likely to be evaluated near to the start of the rule base. You should also put all *define* rules before the *input* and *output* rules.

4.5.4. Debugging

Ensure that the default form of *input* rule will accept any input as valid by setting the default options to \$, #, & (see OPTIONS command in the DESIGN menu). This will enable you to provide values for goals whose rules have yet to be defined. When debugging is complete, the default option should be restricted to minimise the risk of run-time errors, although ideally all input in the final system should be controlled using *input* rules.

When repeatedly asking for a particular goal to be evaluated, remember that you can press F4 instead of having to repeatedly type in a goal name.

Note that you can cause *QL Expert* to retain goal values from previous evaluations by preceding a goal name with a comma or space when asking for a goal to be evaluated.

Don't forget to make use of the special input options (TRACE, WHY, HOW, WHAT IF, VALUES etc.) described earlier.

4.5.5. Preparing For Use

Before releasing your rule base for use by end users, you can do several things to make it easy to use.

First, minimise the likelihood of run-time errors by adjusting the default form of *input* rule as already explained.

Use the INSTALL program to customise features like the default drive, the default secondary output device and in particular to cause your rule base to be loaded automatically when *QL Expert* first starts.

Make use of the LABEL statement to construct a menu of options. Even if your rule base only has one top-level goal, a menu is useful because it reduces the amount of typing required and is intuitively easier for a naive user to understand.

5. Creating And Interrogating A Rule Base

This chapter describes the interactive commands available within *QL Expert*, as opposed to the kinds of rule available for constructing a rule base. The rule base editor is also described, and an example session takes you through the steps of creating, editing and interrogating a simple rule base.

When *QL Expert* is first started, you are presented with a window in the middle of the screen similar to the one shown below:

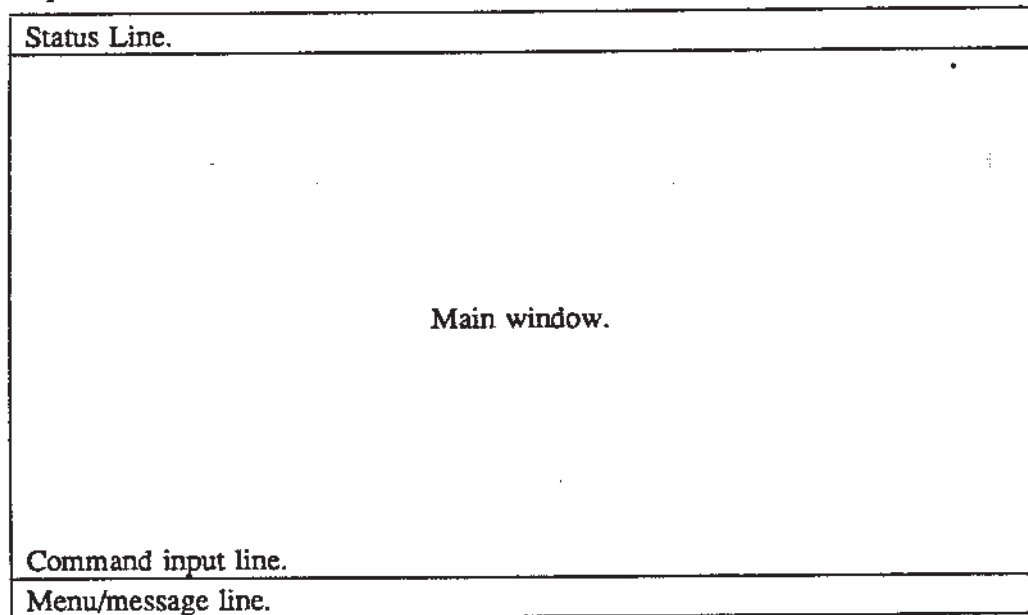
QL Expert	Copyright (C) 1987 Francesco Balena
Welcome to QL Expert	
An expert system shell for the Sinclair QL computer. Copyright (c) 1987 Francesco Balena.	
RULE BASE 'startup_exp' LOADED. 8192 bytes free. Press F2 for a menu of options.	

Note that the two lines immediately below the copyright message will not normally appear because as supplied, *QL Expert* is not configured to automatically load a particular rule base. It can be made to do so using the INSTALL program as described in section 3.3.

In this initial state, *QL Expert* responds to the function keys F1 to F4 as follows:-

- F1 Calls up an on-line context sensitive help facility.
- F2 Activates a menu defined in the currently loaded rule base.
- F3 Activates the top level *QL Expert* command menu.
- F4 Causes the last evaluated goal to be re-evaluated.

Different parts of the window have different functions as follows:



The top line displays a simple heading which varies depending on the mode of operation. For example, when the editor is invoked, the top line changes to indicate this and displays the amount of free memory.

The main window is used both for rule base interrogations and for editing the rule base.

The command input line is normally blank, but is used by built in *QL Expert* commands to issue prompts for user input such as file names, rule numbers etc.

The menu line displays a summary of options available in the current mode. These options vary depending on the menu selected.

The following sections describe *QL Expert's* commands in general, describe the rule editor, and include an example session during which a simple rule base is created and interrogated.

5.1. *QL Expert* Commands And Menus

When the *QL Expert* is first started there are two main options available. You can press F1 to interrogate the extensive on-line help, or press F3 to enter a command from the first *QL Expert* command menu. In theory, you can press F2 to get access to a menu relating to the current rule base, but as you don't have a rule base to interrogate yet, this has no effect.

5.1.1. On-Line Help

First of all, try using the on-line help facility. Press F1 and you will see a summary of *QL Expert* facilities and a list of items about which more information is available. To access this information, you need only type the first letter of the corresponding word, and another page of information will be displayed. This itself may have information that can be accessed in a similar way. Pressing <ENTER> causes the previous page to be displayed. At any point, or if no further information is available, you can press ESC to exit the help mode and return to what you were doing previously. Browse through a couple of pages of information now, and then press ESC to return to the normal *QL Expert* screen.

The help facility is context sensitive, so that if for instance you are using the rule editor when you press F1, you will be presented with help information relevant to the editor. When you exit, using ESC, you will be put back in the editor and can continue with what you were doing.

5.1.2. QL Expert Menus

F3 is used to call up a menu of commands on the menu/message line of the *QL Expert* display. Immediately above this on the command/input line, a prompt will be issued awaiting selection of a command. A command is selected from the menu by pressing the first letter in its name.

An alternative way of selecting commands from this top level menu is to hold the CTRL key down and press the first letter of the command you require.

Some commands produce an immediate action, others prompt for further information on the command/input line, and others call up a further menu of commands. At any point, ESC can be pressed to abandon the current menu or command and return to the normal *QL Expert* screen.

The menus, sub-menus and commands are summarised in the following tables.

Main Command Menu	
Command	Description
DESIGN	Invokes the DESIGN menu which allows various defaults to be changed temporarily (i.e. until <i>QL Expert</i> is re-loaded).
EDIT	Invokes the screen editor. You will be prompted for a rule number, which may or may not refer to an existing rule.
HOME	Clears the screen and moves the cursor to the top left corner of the window.
LOAD	<p>Loads a rule base. A file name will be asked for which can include a device name. If no device name is given, the default device will be searched. If you want to list the directory of a device before giving the file name, you have the option to press '?' and to press <ENTER> for a directory of the default device, or to specify the name of the device to be listed.</p> <p>(Rule bases are stored in files with a suffix of '_exp' which need not be specified by the user. All such files are highlighted in a directory listing using inverse type.)</p>
MERGE	Merges a rule base with the one currently in memory. This command is identical to the LOAD command except that the existing rule base is left in memory, and the new one merged into it. If the new rule base contains rule numbers which clash with existing ones, the new rules will overwrite those already in memory.
QUIT	Quits <i>QL Expert</i> and returns to Superbasic. This command always asks for confirmation.
RULES	Invokes the RULES menu described in another table. This menu is used for manipulating the rule base, including rule renumbering, auto-editing with incrementing rule numbers, searching for strings, listing rules and deleting rules.
SAVE	Similar to the LOAD command, but saves the current rule base in a file.
VALUES	Generates a list of the goals evaluated in the most recent interrogation of the rule base with their values.

Rules Menu	
Command	Description
AUTO	Invokes the rule editor on a series of rules. It prompts for a rule number and increment (defaults 100 and 10), and after completing the edit/creation of a rule commences editing of the next rule after incrementing the rule number. (Similar to the Superbasic AUTO command.)
DELETE	Deletes a rule or group of rules. This command prompts for the numbers of the first and last rules to be deleted (with defaults of the first and last rule in the rule base) and deletes the specified group of rules. If you specify the whole rule base, confirmation is required, at which point you can confirm, abort the command, or save the rule base.
LIST	Lists a rule or group of rules on the screen. Prompts for a range of rule numbers with defaults of the first and last rules in the rule base, and then lists these rules in the main window.
PLIST	Similar to the LIST command, except that output is directed to the secondary output device (set using the DESIGN menu) which will normally be a printer or file.
RENUMBER	Renumbers all, or a specified group of rules. It first prompts for the numbers of the first and last rules to be renumbered, and then for the start number and increment to be used in renumbering. Defaults are offered for all parameters.
SEARCH	Searches the rule base for a string starting from a given point. When an occurrence of the string is found, the user is shown the rule in which it was found, and prompted with a menu of options. These allow the search to be ABANDON'ed, CONTINUE'd, or for the given rule to be EDIT'ed or for the given rule to be LIST'ed on the secondary output device.

The following table describes the DESIGN menu which is used to alter default parameters for the current session of running *QL Expert*. The next time that *QL Expert* is loaded and run, the original defaults will be resumed. If you wish to permanently alter any of these parameters, you can do so using the installation procedure described in section 3.3.

Design Menu	
Command	Description
DRIVE	Changes the default drive used for file access by <i>QL Expert</i> commands (e.g. LOAD, SAVE etc.). You will be prompted for a new default drive (e.g. "flp1_"). If you specify some characters after the device name, e.g. as in "flp1_abc_", then only files starting with "abc_" will be shown. This allows files to be managed as if they were in separate directories.
OPTIONS	Sets the options list for the default form of <i>input</i> rule. See the definition of <i>opt_list</i> in section 6.5 for details of alternative options.
PRINTER	Sets the device used for the secondary output device (usually a printer) which can be any legal QDOS device name (e.g. ser1, par1, mdv2_logfile etc.)
SOUND	For enabling or disabling the use of sound.
WORKSPACE	Enables the amount of workspace that the <i>QL Expert</i> uses to be altered. If you request too large a value, <i>QL Expert</i> will use the maximum amount that can be allocated from the QL's memory. The number number given will be assumed to refer to a number of bytes unless it is followed by a 'k', as in 20k, in which case the amount will be assumed to be in kilobytes.

5.2. Summary Of Editor Facilities

A screen based editor is provided for creating and editing the rule base. The editor is invoked on one rule at a time, although the AUTO command from the RULES menu can be used to automatically edit a series of rules.

Before invoking the editor you must first make sure that you are in the default (start-up) mode by pressing the ESC key. The editor can then be started by either pressing 'F3' followed by 'E', or typing CTRL-E (i.e. hold the 'CTRL' key down and press 'E').

You can type any text on a line of a rule, but when you try to move to another line, *QL Expert* checks it for local syntax errors, and if any are found, will prompt you to fix them before accepting the line. When a line is accepted by the editor, any key words found in it will be re-written in upper case.

The editor lets you move around the screen using the cursor keys, editing any line of the rule at will. A number of additional editing facilities are described in the following table. When you have finished editing the rule, you can store it by pressing F3 followed by S or typing CTRL-S (holding the CTRL key down and pressing the CTRL key) or abandon the changes by pressing the ESC key. Before storing the rule, *QL Expert* checks it for syntax errors and will not accept it until these have been corrected.

Screen Editor Features	
Key	Function
Left arrow	Move cursor left one character.
Right arrow	Move cursor right one character.
Up arrow	Move cursor up one line.
Down arrow	Move cursor down one line.
ENTER	Move to the start of the next line.
CTRL-Left arrow	Delete the character to the left of the cursor.
CTRL-Right arrow	Delete the character to the right of the cursor.
CTRL-Down arrow	Insert a blank line immediately after the current line.
SHIFT-Left arrow	Move the cursor to the previous word.
SHIFT-Right arrow	Move the cursor to the next word.
ALT-Left arrow	Move the cursor to the start of the line.
ALT-Right arrow	Move the cursor to the end of the line.
CTRL-ALT-Left arrow	Delete the current line.
CTRL-ALT-Right arrow	Delete all characters to the right of the cursor.
ESC	Quit the editing session without storing the current rule. You should use F3,S or CTRL-S to store the rule before pressing ESC - see table of extended editor commands.

In addition to the cursor controlled editing facilities, the editor has its own set of extended commands. As with the other menus, these are accessed by either pressing the F3 key followed by the first letter of the command name, or by holding the CTRL key down and pressing the first letter of the command name. Pressing the F3 key will cause the menu of extended commands to appear on the menu/message line.

The following table explains the extended editor commands .

Editor Extended Command Menu	
Command	Description
DELETE	Deletes all lines on the screen after the line containing the cursor.
EXIT	Finishes the editing session leaving the rule that was being edited unaltered. This is equivalent to ending the editing session by pressing the ESC key.
PRINT	Copies the current screen contents to the secondary output device (usually a printer).
RESTORE	Restores the rule being edited to the state when editing was first started. This undoes all changes made to the rule being edited.
STORE	Stores the current rule. Before finishing the edit, <i>QL Expert</i> checks the syntax of the rule being stored and if it finds an error, indicates this and refuses to store it.
UNDO	Undoes all changes made to the current line.

5.3. Examples Of Using The Editor

This section takes you step by step through the creation and interrogation of a very simple rule base. No attention is paid to the techniques of designing a rule base or to the operation of *QL Expert* rule types, both of which are dealt with extensively elsewhere. The purpose is only to illustrate the operation of the editor and how to invoke a rule base once created.

For interest, the rule base we will be entering is intended to evaluate the relative likelihood of the "being" interrogating it being humanoid or alien. Although the rule types used in it are not discussed, they do illustrate several features of *QL Expert*.

First, get *QL Expert* up and running by following the procedure described in chapter 3. During this section, all commands will be specified in long form, i.e. using the function key F3 followed by a command letter. Where you are asked to press F3,E this means first press the F3 key, and then the E key. Remember though that as an alternative you can hold the control key down while pressing the command letter, avoiding the need to press F3 first.

To start with you should be faced with the screen shown at the start of this chapter. If you ever get in a tangle, you can always return to this screen by pressing ESC (the escape key).

There are two ways of invoking the editor. First we'll start by invoking it to enter a single rule. Press F3 and notice that a menu appears on the bottom line of the screen and that the cursor is sitting on the line immediately above this, after a prompt of "Command, ".

We want to create a rule, so press E to select the edit option. The prompt will extend to read "Command, EDIT RULE ", and is waiting for you to input a rule number, so type 100<ENTER> (i.e. a one, two zeros and the <ENTER> key). Notice that the main window has been cleared and "RULE# 100" has appeared on the top line. The status line immediately above, indicates that the screen editor is now active, and shows the amount of memory that is free.

You can move the cursor up and down the screen using the up and down arrow keys and could even alter the rule number if required. Start by replacing the cursor on the first blank line of the main window and typing "define origin". If you make a typing error, use the left and right arrow keys together with CTRL-left arrow (delete left) and CTRL-right arrow (delete right) to fix your error. Now press either <ENTER> or the down arrow to move onto the line below. If you hear a bleep, and the cursor has not moved off the line, it means that *QL Expert* has detected a syntax error in the line. In this case, the menu/message line at the base of the screen will have turned red, and will indicate the nature of the problem. At this stage, a syntax error is almost certainly caused by either an illegal character on the line, or the presence of more than one space character between the two words "define" and "origin". If you get in a real mess, you can clear the whole line and start to enter it from scratch by pressing CTRL-ALT-right arrow.

When you press <ENTER>, or use the down arrow key to move onto the next line successfully, notice that *QL Expert* has changed the line to read "DEFINE origin". The first word is a keyword, which *QL Expert* likes to keep in upper case for clarity, and the second word is the name of the goal being defined by this rule.

Proceed and enter the next line as follows:

```
all of alien,human
```

Make sure that you have typed only one space character between words separated by a space, and that only a comma separates the sub-goals "alien" and "human". Press <ENTER>, and the rule should read as follows:

```
RULE# 100
  DEFINE origin
  ALL OF alien,human
```

If the ALL OF keyword has not been recognised (i.e. changed to upper case), or you are prompted to fix the line because of a syntax error, check that your input exactly matches that shown above. Remember that you can always start entering a line again from scratch by pressing CTRL-ALT-right arrow to clear it.

Suppose that at this point, you realise that you should have included a line between the two entered so far. Move the cursor onto the line containing "DEFINE" and press CTRL-down arrow to insert a line. You should see the last line of the rule move down to leave a blank line in-between. On this line, type "label Human/alien species determination" and press <ENTER> to check its syntax. This rule is now complete and should read:

```
RULE# 100
  DEFINE origin
  LABEL Human/alien species determination
  ALL OF alien,human
```

Assuming that this is so, we now want to store this rule in the rule base. To do so, press F3 and then press S to select *store* and notice that after a very short delay, the screen will go completely blank. During the delay, *QL Expert* is checking the syntax of the rule before storing it. If it contains any errors, it will prompt for them to be corrected and won't accept the rule until all the errors have been removed.

The blank screen indicates that you have left the editor and are now back in the start-up mode. You now have a rule base, albeit of only one rule. Before extending the rule base slightly, and showing how to create or modify several rules in sequence, we will try interrogating this very simple rule base. There are two ways to start the interrogation. The first, and normal way for a user to do so is to press F2 which causes a search for all rules containing a LABEL statement, and uses them to construct a menu. Press F2 now, and notice a menu appear on the screen - with only one option - and a prompt to "CHOOSE YOUR GOAL" at the bottom of the screen. The menu consists of a key letter followed by the text included in the LABEL statement you have just entered. To select this, the only goal, press A. The name of the goal being evaluated will be shown on the status line and you should see the following display:

```
***** memory cleared *****

--> alien ?
```

The first line indicates that any knowledge stored from previous interrogations of the rule base has been discarded. The prompt is asking for the value that should be assigned to the sub-goal called 'alien'. We have not yet provided a rule defining how this sub-goal should be

evaluated, and so *QL Expert* uses a default form of *input* rule instead. This is very useful when developing a rule base as it makes it very easy to try out each rule before adding further rules to evaluate its sub-goals. However, the default *input* rule will accept invalid as well as valid input from the user, making it easy to cause run-time errors. In the final rule base, *input* should be used to trap invalid input and prevent run-time errors.

Lets test out this rule. In answer to the prompt, type 0.05<ENTER>. At the next prompt, which should be "--> human ?", type 0.95<ENTER>. *QL Expert* should respond by printing

```
'origin' is alien
'origin' is human
```

which indicates that both sub-goals alien and human evaluated successfully. That is, neither sub-goal is undefined. In fact 'alien' has evaluated to a value of 0.05 and 'human' to a value of 0.95 by virtue of your input. The goal 'origin' has two values, and is referred to as a multi-valued goal. The above output is pretty meaningless since you aren't told anything about the relative likelihoods of the two sub-goals and we will improve on this by adding an *output* rule shortly. First though, we'll try the second of the two ways of starting to interrogate the rule base.

The first way of starting an interrogation was to press F2 and select a goal from the menu. The second is to simply type the name of a goal that you wish to evaluate. This can be the name of any goal, it need not be a top-level goal, and need not even be defined in the rule base. This feature is very useful when debugging as you can test as large or as small a part of the rule base as you wish. So, type

```
origin<ENTER>
```

to ask for the goal 'origin' to be evaluated. The effect will have been identical to pressing F2 and selecting this goal from the menu. You can enter your response to the two prompts as before or type "quit<ENTER>" to abandon the session. (There is in fact one more way of starting the interrogation, and that is to press F4, which causes the goal last selected for evaluation to be evaluated again.)

Now, to give you some practice with the rule editor, you will probably find it useful to add some more rules, this time using the AUTO command that invokes the editor on several rules in sequence. If you don't want to type the rules in, skip to the end of this section and load the file "HUMAN" using the procedure explained. This file contains the rules as shown.

To enter the rules and familiarise yourself with the editor, follow the instructions below, and experiment with some of the editor facilities described in the two tables in section 5.2. Press F3 to select the top-level menu and press R to select the *rules* menu, and then A to select the AUTO command. After pressing F3,R the prompt will change from "Command, " to "Rule command, " and when you press A will change to "Rule command, AUTO EDIT from 100" with the cursor positioned at the '1'. You could alter the 100 to the number of the first rule that you wish to edit, but we require the default, so just press <ENTER>. The prompt will be extended by ",increment 10" so press <ENTER> to again accept the default. This means that *auto editing* will start with rule 100 and proceed to rules 110, 120, 130 and so on as each rule is stored. On pressing <ENTER> you should be put back into the editor editing rule 100. You could edit this rule, but on this occasion we want leave it unchanged, so press F3,S to store it and move on to edit rule 110. Now type in the following rules, pressing F3,S to store and move on as each is completed. Don't try and type in more than one rule on the same

screen - the rule editor will only edit one rule at a time.

RULE# 110

OUTPUT origin
PRINT Results:
PRINT The likelihood that the subject is @ is %%.

RULE# 120

DEFINE alien
FACTOR 0.75
green skin
bog eyes
OR
over 150 years old
alive

RULE# 130

DEFINE alien
FACTOR 0.9
xray vision
OR
dislike kryptonite

RULE# 140

DEFINE human
FACTOR 0.95
drink abbot ale
own ql

RULE# 150

INPUT green skin
PROMPT How green is subject's skin - 0 to 1 (1=very green)
OPTIONS <=1,>=0

RULE# 160

INPUT bog eyes
PROMPT Does the subject have 'bog eyes' (yes or no)
OPTIONS &

When you have entered the last rule, store it as usual before pressing ESC to finish the auto-editing session. If you don't store first, the last rule will not be saved in the rule base.

You can now experiment a little more with the very primitive rule base that you have created, and can amend and extend it to improve its ability to correctly evaluate a subject.

So far, only a few of *QL Expert's* many facilities have been used, but now that you have become familiar with the basic concepts of rules and the rule editor in particular, you should find it much easier to experiment with more complex features.

One more useful command to try before you move on though is the SAVE command. You can store your rule base in a file by returning to the start-up state of *QL Expert* and pressing F3,S. You will be asked for a file name, so insert a blank formatted cartridge in *mdv2_* and type *mdv2_alien*. (Note that you can't save onto the master cartridge because it is write protected.)

To load the rule base, press F3,L followed by the name of the file (i.e. alien). The file is actually stored as "alien_exp", but you need not specify the "_exp" extension. If you are ever unsure of the name of a file, you can type a '?' at any prompt where a file name is expected. This will ask which drive you want to list the files from, and when you reply produces a listing in which "_exp" files are highlighted in inverse video. You can then enter the file name required.

6. Rule Types And Programming Syntax

This chapter contains full details of the programming constructs provided by *QL Expert*. The full syntax of each rule type or construct is specified with a description and usually an example of use. The examples given build into a very primitive rule base which can be found on your *QL Expert* cartridge in the file CAR_EXP. This rule base is primarily aimed at showing the use of each rule type rather than how to construct a usable rule base. However, you can load it, press F2 to get a menu and use it to see how your input is interpreted by the different rule types illustrated. You may also find it a useful starting point for experiments by trying to improve its operation - it certainly has plenty of room for improvement!

6.1. Kinds Of Rule

QL Expert provides three basic kinds of rule: *Define* rules, *input* rules and *output* rules. *Define* rules describe how to evaluate goals, whereas *input* and *output* rules are used to improve interaction with the user.

Input rules can define the prompt that should be given to the user and have a powerful error trapping facility to ensure the validity of input, and if necessary, to explain to the user why input has been rejected.

Output rules specify how the results of an evaluation should be presented. *Output* rules enable goal values (whether boolean, numeric or strings), and or goal uncertainties to be embedded in a given string. The content of the output can also be made to vary dependent on the values of given goals.

For reference purposes, each rule is given a number in the range 1 to 65535. When defining rules automatic number generation and rule renumbering is available, similar to that in Superbasic (see chapter 5).

6.2. Goal Types

QL Expert allows goals to have the following kinds of value:-

Boolean	(0 for false or 1 for true).
Uncertainty	(a real number in the range 0 to 1).
Real	(a real number without range restrictions).
String	(a word or sentence not starting with any of 0123456789.+ or -).

Mixing of types is permitted wherever it will be meaningful. For example, *boolean* and *uncertainty* values may be mixed freely without exception, but a run-time error would be generated if a *string* was returned where a *boolean*, *uncertainty* or *real* value was expected. Type checking is performed at run-time, and so the rule base must be carefully designed to avoid situations where a returned value is either of the wrong type, or outside the valid range of the

expected type. A list of restrictions is given below:

Expected Type	Types that can be returned legally		
	Boolean	Uncertainty	Real
<i>Boolean</i>	Always	Always	$0 \leq \text{real} \leq 1$
<i>Uncertainty</i>	Always	Always	$0 \leq \text{real} \leq 1$
<i>Real</i>	Always	Always	Always
<i>String</i>	Never	Never	Never

Hence, it is valid to return a *real* value where a *boolean* or *uncertainty* is required providing that its value is guaranteed to be greater than or equal to zero and less than or equal to one.

A *string* typed goal can in some circumstances have multiple values. This is made possible by the ALL OF statement used in *selective define* rules discussed in section 6.6.2, and is useful for presenting the relative likelihoods of a number of possible outcomes, rather than having to select a single definitive answer.

6.3. Goal Names

A goal name can be any string of alphabetic and numeric characters, but must start with an alphabetic character. Spaces may also be included, but beware that a difference in the number of spaces between two words in a pair of otherwise identical goal names will cause them to be treated as different goals. The case of alphabetic characters is not significant.

Examples of valid goal names:-

colour black
 Colour black
 COLOUR BLACK
 Measurement 1
 Measurement 1
 Weight
 Likelihood of height under 156cm

Note that the first three names refer to the same goal, whereas the rest all refer to different goals.

Examples of illegal goal names:

1st bore promising (starts with a numeral)
 Height under 1.56m (contains illegal '.' character)

6.4. Notation Used In Syntax Definitions

This section defines the notation used in the definitions of rule syntax contained in later sections. Characters with special meanings are as follows:-

- | Is an OR operator indicating that one of a number of elements is permitted to appear at a given point.
- [] are used to enclose an optional element that may appear either once or not at all.

{ } are used to enclose an element that may appear once, repeatedly, or may be omitted altogether.

Here are some examples:

any_expression = *arithmetic_expression* | *conditional_expression*

Means that wherever *any_expression* appears, it can be replaced by either *arithmetic_expression* or *conditional_expression* which will have been defined elsewhere.

goal_list = *goal_name*{,*goal_name*}

Means that a *goal_list* consists of one or more occurrences of *goal_name* separated by commas. Spaces are allowed within goal names and so cannot be used between them.

signed_number = [+ | -]*number*

Means that a *signed_number* is a *number* optionally prefixed by either a '+' or a '-'.

6.5. Preliminary Definitions

The following elements are referred to in later definitions:

goal_name

is the name of a goal and may contain any combination of alphabetic and numeric characters, plus space characters so long as the first character is alphabetic. The case of alphabetic characters is not significant.

goal_list

is a list of goal names separated by commas (spaces are only permitted as part of a goal name - not as separators). An alternative definition would be:

goal_list = *goal_name*{,*goal_name*}

opt_list

is a list of options or validity tests for user input, and is used as part of an *input* rule. The formal definition is as follows:

opt_list = \$ | # | % | *string* | *real* | *val_test*{,*string* | *real* | *val_test*}

val_test = <*real* | =<*real* | <*real* | >*real* | >=*real*

\$ means any valid *string*

means any valid *real*

% means any valid *uncertainty*, i.e. a number between 0 and 1.

& means any *boolean* value, i.e. *yes*, *no*, *true* or *false*.

An *opt_list* is used to restrict the responses that will be accepted for a given goal evaluation under the control of an *input* rule. Input by the user will be accepted provided it matches one of the strings or numbers in the list, or it passes all the specified validity tests. Note that "\$,#" means that any *string* or *real* will be accepted by the *input* rule.

real is a real number with an absolute value in the range 1E-615 to 9.999999E+615.

uncertainty

represents a value which must be a real number in the range 0 to 1.

expr is an arithmetic expression conforming to the syntax used by Superbasic, but with the following exceptions: A *goal_name* may be used in an expression wherever a number is permitted as long as all spaces present in the *goal_name* are replaced by underscores. Note also that *QL Expert* provides a number of function calls that may be included in expressions (see chapter 7). Valid expression operators are: +, -, *, /, ^ and !. ^ means "to the power of" and ! is a probabilistic OR operator such that

$x!y$ is equivalent to $x + y - x*y$.

Note that where the operands are *boolean*, its effect is identical to a logical OR operation.

comp

is a comparison defined as follows:

comp = *expr* [comparison_operator *expr*]

Valid comparison operators are:

=	(equal to)
≠	(not equal to)
≈	(approximately equal to - as Superbasic)
<	(less than)
<=	(less than or equal to)
>	(greater than)
>=	(greater than or equal to)

string

is a sequence of characters that must begin with an alphabetic character. Unless stated otherwise, it may contain any printing ASCII character. That is, any ASCII character except for control characters such as a tab or carriage return.

file is a valid QDOS file name. If the drive name is omitted, the default drive will be used.

pntr is an integer value to be used as a pointer to a byte in a file.

6.6. Rule Types And Syntax

This section formally describes the syntax of rule types provided by *QL Expert*. Brief explanations are given of how *QL Expert* evaluates each kind of rule, but you should refer to chapter 4 for detailed explanations of rule evaluation. Note that more than one rule can refer to any given goal, providing a mechanism for multiple evaluation paths.

6.6.1. Ordinary Define Rule

```
RULE# n
    DEFINE goal_name
    [LABEL string]
    [FACTOR real]
    [NOT] goal_name | statement
    {[OR]
    [NOT] goal_name | statement}
```

Returns: *boolean* | *uncertainty* | *undefined*

A LABEL statement is used to include the goal defined by this rule in a menu, but has no effect on how the rule will be evaluated. If a FACTOR statement is present, the rule will be evaluated and the result will be multiplied by the *real* value given. This allows rules to be given weights.

The remainder of the rule definition is a mixture of goal names, and statements. The body of the rule must contain at least one line, and there must always be at least one line either side of an OR operator. Each of these lines will return a value when it is evaluated (as part of the process of evaluating this goal). Which lines are actually evaluated to determine the value of the goal will depend on the structure of the rule and the values returned by other lines. All lines must evaluate to either true (i.e. 1), false (i.e. 0), an uncertainty (i.e. a real value between 0 and 1) or undefined.

There is an implicit AND operation binding lines which are not separated by an OR operator. Such lines are evaluated in turn until a false (i.e. 0) value is returned, or an OR operator is encountered, marking the end of a series of AND operations. When one of these events occurs, the group of AND'ed rules can be evaluated. If a false value is encountered, the value of a group of AND'ed sub-goals must be zero (even if some sub-goals are undefined) and so the remainder can be skipped.

Each group of AND'ed sub-goals is reduced to a single value by taking the lowest value of the group. If any member of the group is undefined, then the group will be undefined, unless there was a zero value, in which case the group can be guaranteed to evaluate to zero. This works equally for values between 0 and 1, as well as boolean values.

Successive groups of AND'ed goals are reduced to single values in this way until either they have all been evaluated, or a group becomes undefined. The values determined for groups of AND'ed goals can then in turn be OR'ed to obtain the end result of evaluating this rule. If all AND'ed groups were successfully evaluated (i.e. were not undefined), the result of the OR'ing the group values will be taken as the maximum value of any of the AND'ed groups. However, if any group is undefined, the whole goal becomes undefined, because it is impossible to say what the actual goal should evaluate to.

Examples:

RULE# 100

DEFINE Time to buy a car
LABEL Advice on whether you should purchase a car
Do you have a car
Does your car need replacing
OR
NOT Do you have a car
Do you need a car

RULE# 110

DEFINE Does your car need replacing
FACTOR 0.75
Will it be more economical to replace than to repair

RULE#120

DEFINE Does your car need replacing
EVAL Prob_of_fail_MOT*Likely_MOT_cost/New_car_cost

6.6.2. Selective Define Rule

RULE# *n*

DEFINE
[LABEL *string*]
[CONDITION *cond*]
FIRST OF *goal_list* | MAX OF *goal_list* | MIN OF *goal_list* | ALL OF *goal_list*
{ *goal_list* }

Returns: *string*{*string*} | *undefined*

This kind of *define* rule is very powerful. It enables a list of goals to be specified for evaluation, and by offering different types of list construct, provides different mechanisms for selecting the return. The *selective define* rule can cause a list of sub-goals to be evaluated, and will return the names of one or more of the sub-goals depending on the outcome of the evaluation.

The effect of the LABEL statement is the same as in an *ordinary define* rule.

A *selective define* rule will often be the top level rule in the rule base, but if it is used to define a sub-goal within another rule, you must remember that it can only ever return a *string* (or *undefined*) value. Note that the P() function can be used to obtain the numeric value of a string which is the name of a goal. Using this technique, the *selective define* rule enables multiple goal paths to be implemented.

Four kinds of selection construct are provided, FIRST OF, MAX OF, MIN OF and ALL OF. Each is followed by a list of goals, but only one selection construct is permitted per rule. each construct causes part or all of a list of sub-goals to be evaluated, one by one, and requires that each sub-goal return a numeric value. If a *string* is returned by a sub-goal, a run-time error will be generated. The effects of each construct are described below.

FIRST OF

Evaluates each goal in its list, until one returns a value greater than zero. The rule will

return either the name of the first successfully evaluated sub-goal, or *undefined* if none returns a value greater than zero.

MAX OF

Evaluates all the sub-goals and returns the name of the sub-goal returning the highest numeric value. Only if all evaluate to *undefined* will the rule evaluate to *undefined*.

MIN OF

Has a similar effect to MAX OF, except that it returns the name of the sub-goal returning the smallest value.

ALL OF

Forces all the sub-goals to be evaluated. A goal defined using ALL OF will normally become a multi-valued, goal. That is, it will have a set of values corresponding to the names of all the sub-goals which returned non-zero values. It is useful for evaluating a list of possible solutions to a problem, and then with the help of a suitable *output* rule, to list all the solutions which were evaluated successfully, and to show their relative merits by also displaying their numeric values.

By default, each sub-goal must evaluate to a value greater than zero if it is to contribute to the value of a goal defined by the *selective define* rule. This default can be modified to any conditional test using a *CONDITION* statement. For example,

```
CONDITION >.5
ALL OF ...
```

would assign the names of only those sub-goals evaluating to a number greater than 0.5, to the multi-valued goal being evaluated. Interestingly,

```
CONDITION <.5
MAX OF ...
```

causes the sub-goal with the highest value less than 0.5 would be returned. Goals with values greater than or equal to 0.5 will be ignored.

Examples:

RULE# 130

```
DEFINE What kind of car should you buy
LABEL Advice on kind of car to buy
MAX OF Estate car,Saloon car,Hatch back,Hearse
Sports car,Racing car
```

RULE# 140

```
DEFINE Features of importance
LABEL Advice on points of importance
CONDITION >0.75
ALL OF Want low running costs,Want high performance
Want reasonable luggage space,Want large luggage space
Want low purchase cost,Want high level of comfort
Want small overall size,Want to drive on public roads
```

6.6.3. Functional Define Rule

```
RULE# n
  DEFINE
  [LABEL string]
  {IF comp
  RETURN expr}
  RETURN expr
```

Returns: *boolean | uncertainty | real | undefined*

The effect of the LABEL statement is the same as in an *ordinary define rule*.

This kind of rule is similar to an ordinary function call as used in languages such as Pascal, or Superbasic. IF statements can be used to select which of a number of RETURN statements will provide the result. The value returned will be the number resulting from evaluating the expression given in the RETURN statement that is chosen.

Example:

```
RULE #150
  DEFINE Want low running costs
  IF TEST(Is_low_cost_very_important)
  RETURN Is_low_cost_very_important
  RETURN Running_cost_importance
```

```
RULE# 160
  DEFINE Running cost importance
  FACTOR 0.75
  Does running cost override performance
  Does running cost override status value
```

```
RULE# 170
  DEFINE Running cost importance
  Do you earn under 20000 per year
```

6.6.4. Input Rule

```
RULE# n
  INPUT goal_name
  PROMPT string
  [OPTIONS opt_list]
```

Returns: *boolean | uncertainty | real | string | undefined*

An *input rule* should be defined at every point in the rule base where the value of a goal is to be determined directly by input from the user. During development of the rule base, this is unnecessary, because *QL Expert* assumes a default form of *input rule* for every goal it encounters for which there is no rule definition. However, this makes it possible for the user to cause run-time errors by assigning invalid types or out of range values to a goal, and so it is important that the default *input rule* only be used during development.

The PROMPT statement specifies the prompt that will be put to the user when this rule is evaluated. The *string* will be printed out with an appended question mark. When the user ends his input by pressing the <ENTER> key, the *opt_list* (if present) will be used to test the validity of the input. If it is illegal, a list of valid inputs will be presented and the original prompt repeated. Only when a valid input has been given will it be assigned to the goal being evaluated.

See section 6.5 for a formal definition of the *opt_list*. Individual values and ranges of values as well as strings, can be specified as valid inputs. *QL Expert* makes sure that the input either matches one of the individual values or that it satisfies all of the specified conditions of the *opt_list* before accepting the input as valid.

Note that an *input* rule cannot ever generate a run-time error. If the user tries to provide an invalid input, he will be prompted with a list of valid inputs, and the question will be repeated. However, run-time errors will be generated if an *input* rule is defined that allows the user to assign a value to a goal that is used in a rule expecting another type. In this case, an error will be generated when the *define* rule tries to make use of the goal value.

Example:

```

RULE# 180
  INPUT Is low cost very important
  PROMPT Is low cost very important (answer yes or no)
  OPTIONS yes,no

```

6.6.5. Output Rule

```

RULE# n
  OUTPUT goal_name
  [IF cond
  PRINT string | REPORT string | FPRINT file[,pntr|,-page]]
  PRINT string | REPORT string | FPRINT file[,pntr|,-page]
  [[IF cond
  PRINT string | REPORT string | FPRINT file[,pntr|,-page]]
  PRINT string | REPORT string | FPRINT file[,pntr|,-page]]

```

An *output* rule is used to define how the results of an enquiry session are to be presented to the user. It is possible to define a line or block of text which will be printed when a top-level goal has been evaluated. A top-level goal is a goal that does not appear as a sub-goal within any other goal definition.

Both the name of the goal and its value can be included in the output text.

In the case of a multi-valued goal, output which contains either the goal name or its value will be repeated for each of the (multiple) values enabling a list of results to be compiled (see below).

IF statements can be used to make the format of the output dependent on the state of the rule base.

6.7. PRINT, FPRINT and REPORT Statements

PRINT, FPRINT and REPORT statements are used to format the results produced by *QL Expert* at the end of an interrogation. Although they can only occur within an *output* rule, they

are discussed in this separate section because they are relatively complex.

A mixture of PRINT, FPRINT and REPORT statements may be present in an *output* rule, in which case each will be obeyed in turn until there are no more statements in the rule, or a REPORT statement is encountered. After obeying the REPORT statement, evaluation of an *output* rule stops, even if it contains further statements. When used with IF statements, this allows the format of the output to be determined by the state of the rule base. Note that when an IF is used within an *output* rule, all goals required by the IF must have already been evaluated. Also note that where an IF encounters an undefined goal, the statement following it is always evaluated.

By including special characters within the string, the value of the given goal can be included in the output text.

PRINT and REPORT

Syntax:

PRINT *string*

or

REPORT *string*

The following characters can be embedded in the string in order to include certain information in the output:-

\ must be followed by a word of text (without a separating space). When the string is printed as part of an *output* rule, the word following the back-slash will only be printed if the goal corresponding to this rule has a value of zero. For example:

OUTPUT poisonous

PRINT This mushroom is \not poisonous

would produce an output of "This mushroom is poisonous" if *poisonous* was non-zero, and "This mushroom is not poisonous" if *poisonous* was zero (i.e. FALSE).

@ is replaced by the value of the goal associated with this *output* rule. This includes goals with string or numeric values. (See examples shortly.)

is used to insert the numeric uncertainty of a string valued goal into a string.

%% is used in the same way as "@" and "##", but causes the uncertainty value to be printed in percentage format. So, for example, an uncertainty printed as ".35" using "##" would be printed as "35%" if "%%" was used instead.

Note that in the case of a multi-valued goal (where an ALL OF statement has been used in the goal's rule definition), a PRINT or REPORT statement containing any of "@", "##" or "%%" will be re-executed for each of the goal's values.

For example if the following *output* rule were used for a goal which has multiple values

OUTPUT hardware fault

PRINT The fault is probably due to one or more of

PRINT the @ (probability %%)

the following output would be produced if the goal evaluated to have three values:-

The fault is probably due to one or more of

the disc drive (probability 5%)

the memory interface (probability 25%)

the power supply (probability 74%)

Further examples:

RULE# 400

OUTPUT What kind of car should you buy

IF P(What_kind_of_car_should_you_buy)>=.8

REPORT A @ is strongly recommended (scoring %%)

PRINT A @ is recommended (scoring %%)

RULE# 410

OUTPUT Features of importance

PRINT The most important features to consider are:

PRINT @ (score %%)

Rule 410 produces a list of the features meeting the condition statement in rule, together with the actual percentage score of each goal.

FPRINT

FPRINT is similar to PRINT, except that the output text is taken from a file, reducing the amount of RAM required to store the rule base. A whole file, a page within a file, or a section of a file can be specified. Note that the special characters used to insert goal values in PRINT statements have no effect when included in a file. You can of course mix PRINT and FPRINT statements, so this is not restrictive.

The syntax is as follows:

FPRINT *file*[*,pntr* | *-page*]

The *file* can include a device name, and may be followed by one other parameter. If no device name is specified as part of the file name, the default drive will be searched.

If only a file name is given (i.e. no *pntr* or *page*) the file will be displayed from the start, until an ESC character (decimal 27 or CHR\$(27)) is found in the file. When large numbers of lines are printed, they will be shown twenty at a time and the user will be prompted to press a key in order to see the next page. Lines should not be longer than 76 characters and should be terminated by a new-line character (decimal 10 or CHR\$(10)) which is automatically generated by Superbasic PRINT statements and most editors.

If a second parameter is specified and it is a positive number, it will be interpreted as a character position within the file. If the number is negative, it will be interpreted as a page number.

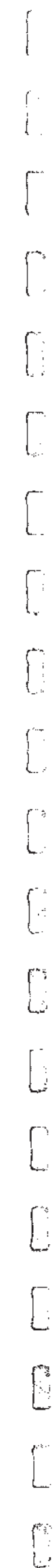
If a character *pntr* is given, it indicates the point in the file at which to start displaying lines.

If a *page* is given the display will start at this page in the file. Page one starts at the beginning of the file, page two after the first ESC character and so on.

7. Special Functions

The following table describes a number of special functions provided by *QL Expert*.

Function	Description
ABS(<i>x</i>)	Returns the magnitude or absolute value of <i>x</i> . (See also: MOD0)
BINOM(<i>n</i> , <i>k</i>)	Returns the binomial coefficient, i.e. $n!/[k!(n-k)!]$. Note that both <i>n</i> and <i>k</i> must be between 0 and 300. (see also: FACT0)
EXP(<i>x</i>)	Returns the exponent of <i>x</i> , i.e. e^x . (See also: LN0)
FACT(<i>x</i>)	Returns the factorial of a number, i.e. $x!$ or $x.(x-1).(x-2)...1$
INT(<i>x</i>)	Returns the integer part of <i>x</i> .
LN(<i>x</i>)	Natural logarithm: $\log_e(x)$ (See also: EXP0, LOG0)
LOG(<i>x</i>)	Base ten logarithm: $\log_{10}(x)$ (See also: LN0)
MAX(<i>x</i> , <i>y</i>)	Returns the value of the parameter with the highest value.
MIN(<i>x</i> , <i>y</i>)	Returns the value of the parameter with the smallest value.
MOD(<i>x</i> , <i>y</i>)	Returns <i>x mod y</i> similar to Superbasic's MOD function, except that this version allows floating point arguments.
P(<i>string</i>)	Returns the numeric value of the goal whose name matches the <i>string</i> . An error will be generated if the value is outside the range 0 to 1. Note that all spaces in the goal name must be substituted by underline characters. (See also: TEST0)
ROUND(<i>x</i> , <i>y</i>)	Returns <i>x</i> rounded to <i>y</i> decimal digits. The value of <i>y</i> must be in the range 0 to 8. (See also: INT0)
SQRT(<i>x</i>)	Returns the square root of <i>x</i> .
TEST(<i>goal_name</i>)	Returns zero if this goal's value is undefined or has not been evaluated yet. If the goal has been successfully evaluated, this function returns a value of one. (See also: P0)



An expression has been found to contain more or less operands than are required.

Options Related Errors

The following errors may be generated in connection with either an OPTIONS clause in an *input* rule, or the OPTIONS command from the DESIGN menu:-

- 15 SYNTAX ERROR : allowed comparison operators are < <= > >=
- 16 SYNTAX ERROR : a number is expected after the comparison operator
- 17 SYNTAX ERROR : check your commas and '='s

A line has been found to either start or end with a comma or '=', or contains two consecutive commas or '='s.

General Editor Errors

These errors may be detected when trying to save a rule from the editor:-

- 18 SYNTAX ERROR : more subgoals in the same line
- 19 SYNTAX ERROR : can't be the first keyword of a rule
- 20 SYNTAX ERROR : can't appear in this kind of rule
- 21 SYNTAX ERROR : can't appear in this position
- 22 SYNTAX ERROR : extra lines required

This rule is incomplete.

- 23 SYNTAX ERROR : consecutive IF's are not allowed
- 24 SYNTAX ERROR : consecutive OR's are not allowed
- 25 OUT OF MEMORY

The syntax of this rule is correct, but there is no room for it in the program workspace. If the rule is long, and would take time to re-type, try trimming lines from it and re-saving. Then, use the design menu to increase the program workspace. (This will only work if you have sufficient QL RAM spare for the program to make use of, if not, you should consider purchasing a RAM expansion).

8.3. I/O Errors

The following errors are connected with file or device input and output:-

- 26 In use
- 27 Bad or changed medium

The medium may have become damaged, or data may have been corrupted, or it may simply have been removed from the drive. Remove and replace the medium before trying again.

- 28 End of file

This may occur if an FPRINT statement is executed where the pointer parameter has too large a value. If happens at any other time, it may indicate that the file being accessed has been corrupted.

29 Drive full

This may occur during an attempt to save the contents of a rule base. If so, try again, but use a medium with more free space.

30 This file does not exist

31 This file already exists - Press 'Y' to overwrite

32 OUT OF MEMORY : ____ bytes required

The file being loaded is too large for the current program workspace. Try increasing the size of the workspace using the DESIGN command before trying to re-load the rule base.

33 Directory ____ does not exist

You have either specified an incorrect drive name or a drive without a formatted medium in it.

34 File ____ can't be shown - Press 'R' to retry

An FPRINT operation has been attempted on a file that cannot be found. You have the option of changing the medium and re-trying the operation or to abandon it, which is useful when debugging.

35 Insert system cartridge in ____ and retry

Help has been called for, but the program cartridge or disc is not in the default drive.

36 Read only device

An attempt has been made to save on a device which has been write protected. This will occur if you try and save onto you master cartridge which has had its write protect tag removed to prevent files being accidentally deleted or overwritten.

37 Default secondary output stream ____ can't be opened

For some reason, the device name given for the secondary output device cannot be opened. Try altering it using the DESIGN/PRINTER command.

8.4. User Input Errors

The following run-time errors may be generated during evaluation of *input* rules:-

38 Invalid string

Your input was not a valid number, and so is assumed to be a string, but has been rejected because it starts with a digit, decimal point, plus sign or minus sign.

39 Too many pending WHAT IF statements

A seventh level of WHAT IF has been attempted.

40 There are no pending WHAT IF statements

An exit command has been given when there are no pending WHAT IF statements.

41 VALID ANSWERS :

The reply given was either not in the currently active list of valid answers (defined either by an *input* rule, or by defaults set by the DESIGN/OPTIONS command or by the INSTALL program). A list of valid inputs will be given. This will also be printed if <ENTER> is pressed on its own.

8.5. Run-time Errors

42 ***** memory cleared *****

This message is printed whenever the "facts" gathered during an interrogation session are cleared. This normally happens whenever a goal name is typed for evaluation. However, you can retain the data gathered from a session, and start to evaluate a goal by preceding its name by a comma or space. This is a very useful feature when debugging.

43 ERROR : out of range [0,1] in rule# _____

A value has been encountered within a rule that is outside the range required for evaluating boolean or fuzzy logic. This will most often occur when goals are being evaluated without error trapping through the use of *input* rules. Note that it is quite legal for a goal to have any value whatsoever, only when the goal value comes to be used in the evaluation of another rule will it cause this error. Hence, such errors may not be detected until long after values have been input or assigned to a goal.

44 ERROR : out of memory while evaluating rule# _____

This will occur if there is insufficient room in program workspace to store the value of a goal that has just been evaluated. Use either the *INSTALL* program, or the *DESIGN/WORKSPACE* command to increase the program workspace. If the problem remains, you may need to purchase a RAM expansion.

45 ERROR : too many pending rules

As supplied, *QL Expert* can evaluate a rule base of up to fifteen rules deep. If this error occurs, you may want to increase this by altering the maximum level of recursion using the *INSTALL* program, but note that this will increase program memory requirements.

46 ERROR : overflow while evaluating rule# _____

This may be caused by any mathematical operation (for example a "divide by zero" error).

47 ERROR : illegal operand in rule# _____

An *LN()*, *LOG()* or *SQRT()* has been attempted on a negative number.

48 ERROR : number expected in rule# _____

A goal which has a *string* type value has been included in an expression that requires a numeric value. Note that the *P()* function can be used to find the numeric probability factor of a *string* valued goal.

49 ERROR : rule# _____ invokes itself

A goal cannot appear within a rule subsequently used directly or indirectly to define itself.

50 ERROR : syntax error in rule# _____

This error should never occur as rules are checked for syntax before being accepted into the rule base. It most probably indicates that the rule base or program memory has become corrupted.

8.6. Miscellaneous Errors

51 MEMORY CONTENTS WILL BE LOST : Press 'S' to save in file

This message gives you an opportunity to save a rule base that has been changed since the last save operation before continuing with a command that will cause all changes to be lost.

52 Search failed

No more occurrences of a string being searched for can be found.

53 Unable to load '_____'

The INSTALL program allows the name of a rule base file to be specified for automatic loading whenever *QL Expert* is started up. The above message indicates that this file could not be found.

54 Return to Superbasic

This message is printed after using the QUIT command.

55 MERGE not complete

A rule base merge operation failed, probably due to a shortage of program workspace or file input error.

56 MEMORY CLEARED. _____ free bytes

The rule base has been cleared using the RULES/DELETE command.

57 Memory is empty

A VALUES command has been given before any goals have been evaluated.

8.7. Informative Messages

58 Press '?' to list files

All file operations have the option to list the directory of any device by typing a '?' instead of a file name.

59 Press <SPACE> to stop printing

This option is given whenever listing operations are directed to the secondary output device (normally a printer).

60 Press any key to continue

61 Choose your goal

9. Distributing QL Expert Rule Bases

If you wish to distribute a program that uses *QL Expert* whether for sale or just for use by others on a friendly basis, then you must obtain a license from Compware before doing so.

We offer attractive licensing terms to individuals or software houses wishing to distribute *QL Expert* with their software, but will of course take the necessary steps should *QL Expert* be distributed without permission or outside the terms of a given license.

If you wish to find out more about licensing, please write giving brief details of your intention. Please indicate if you would be interested in a cut down version of the *QL Expert* to reduce the amount of memory required.

10. What To Do If Things Go Wrong

BAD OR CHANGED MEDIUM

If your *QL Expert* cartridge has become corrupted and you do not have a backup copy, (shame on you!) all is not lost. Return your original cartridge to Compware within 30 days of purchase, enclosing proof of date of purchase (or quoting your invoice number) and we will replace it free of charge. If the 30 day period has expired, the charge will be the same as a new version upgrade which includes the latest version of the software and documentation. Contact us for pricing.

IF ALL ELSE FAILS

If after thoroughly reading the manual and studying the examples provided, you are sure that you have found a problem in the operation of *QL Expert*, we would be very grateful if you would describe the problem so that we can correct it. Before doing this though, you can help us considerably if you first perform a simple investigation. If you have any add-on units (such as ROM toolkits, disc drives, mice etc.), remove them and see if the problem is still present. If not, identify which of the add-ons is linked with the problem by gradually re-introducing them, and let us know the results.

Please use the *problem report form* at the back of this manual (or a copy of it) to give us your details and describe the problem. Try and fill in all sections of the form and to give step by step details of the problem you are having.

11. Bibliography

Below is a list of titles for users interested in reading up a little more on the theory of expert system design. There is such a range of books available on this subject that it is impossible to select definitive texts. Those listed vary in both price and technical depth. If you have a good scientific book store available, it would certainly be worthwhile browsing through a few titles yourself, but if not, one of the texts listed should prove useful.

The following references offer a good general introduction to the design and operation of expert systems using tools such as *QL Expert*. The second also contains some discussion of the methods of implementing expert system shells such as *QL Expert*.

"Expert Systems - A Practical Introduction" P.S. Sell
Macmillan, ISBN 0-333-37264-6

"Expert Systems for Personal Computers" M. Chadwick
Sigma Press, ISBN 1-85058-044-8

The next two titles go into significantly more detail and are more theoretically oriented. They are also significantly weightier and more expensive.

"A Practical Guide to Designing Expert Systems" S.M. Weiss.
Chapman & Hall, ISBN 0-412-26450-1

"Artificial Intelligence" P.H. Winston
Addison-Wesley, ISBN 0-201-08259-4

QL Expert - v1.1 NEW

QL Expert is an expert system shell for the Sinclair QL computer. It is powerful enough for use by the professional knowledge engineer and at the same time has been designed to introduce the novice to expert system design. All the facilities that you will need to produce practical expert systems are included in this package.

QL Expert uses artificial intelligence techniques for the construction and evaluation of a rule base. Seven rule types are provided together with extensive facilities for defining custom probability relationships, trapping input errors, formatting output, debugging, rule base management etc.

A run-time option is available to allow commercial distribution of expert systems produced using *QL Expert*.

Main Features

- Forward chaining
- Boolean and fuzzy logic
- Seven rule types
- Multiple goal paths
- Many debugging facilities
- Extensive on-line help
- Syntax sensitive rule editor
- Powerful reporting and explain facilities
- User definable probability relationships
- Intelligent rule base searching and pre-scanning
- Special constructs: MAX OF/MIN OF/FIRST OF/ALL OF
- Decision analysis: HOW/WHY/FACTS/WHAT IF/TRACE
- Fast evaluation using pre-compiled expressions
- Ability to divert output to file/printer

Mathematical functions: ABS(), BINOM(), EXP(), FACT(), INT(), LN(), LOG(), MAX(), MIN(), MOD(), PO(), ROUND(), SQRT(), TEST().

With *QL Expert* you will learn how to construct a rule base capable of making apparently intelligent assessments of problems. The "intelligence" or knowledge of the rule base is constructed from three main kinds of rule: The *ordinary*, *selective* and *procedural* kinds of *define* rule. These support a wide range of constructs for boolean and fuzzy logic methods of goal evaluation, and also support user definable probability relationships. Useful probability functions such as the binomial and factorial expansions are built into *QL Expert*.

Once you have created a rule base, it will be capable of asking questions of a user and adapting its search for further information dependent on the answers given. During the interactive interrogation, the user is able to request an explanation of why he is being asked a particular question. He can also record details of the questions and responses in a file or on a printer for later analysis. Once the expert system has collected all information that it "regards" as useful, it presents its conclusions in a format defined by the programmer. The presentation of conclusions can be made to vary depending on the outcome of sub-goal evaluations and if appropriate to include text taken from files. Reporting is very powerful and includes controlled substitution of goal names and values within the output in various formats. Other features include simple windowing, histograms and user input via menus.

Many features have been provided to simplify the development and debugging of a rule base. These include the ability to request evaluation of any goal, and to manually provide values of goals for which rules have yet to be defined. A trace facility can be used to print the values of sub-goals as they are evaluated.

More Than Enough Power For The Professional

Designed For Use By Novice And Expert

QL EXPERT v1.10
Supplementary Manual

Copyright © 1988 Francesco Balena & Compware

All rights reserved

CONTENTS

1. Introduction
2. Summary Of Enhancements
3. Revised Loading Procedure
 - 3.1 Making A Backup
 - 3.2 Making A Working Copy
4. Clarification: Pre-scanning Feature
5. Reporting
 - 5.1 Reporting During Pre-scanning
 - 5.2 Conditional Reporting
 - 5.3 Undefined Goals
 - 5.4 FPRINT Reports
 - 5.5 Substitution Characters In Report Strings
6. New Keywords
 - 6.1 FREEZE
 - 6.2 HOME
 - 6.3 WARNING
 - 6.4 ERROR
 - 6.5 EXPLAIN
 - 6.6 REMEMBER
 - 6.7 FORGET
 - 6.8 SET
7. Reserved Goals
 - 7.1 TRACE
 - 7.2 ECHO
 - 7.3 RULENUMBER
 - 7.4 KEYCODE
 - 7.5 DEDUCE
8. Multiple OPTION Lines
9. Inputting Values From The Editor
10. Query Session Commands
11. The Bottom Line
12. The VALUE Command
 - 12.1 New DESIGN And RULE Options
13. New Screen Editor Commands
14. The TEST Function
15. The INSTALL Program
 - 15.1 Error Messages
16. QL Expert Manual Errata
 - 16.1 Customising Default OPTIONS
 - 16.2 Boolean Input Options

1. Introduction

This supplement documents the features introduced in *QL Expert* version 1.10. First a brief summary is given, followed by detailed descriptions of all the new and modified features. Finally some errata to the original *QL Expert* manual are noted.

Run-time Expert

You may also be interested to hear of a *run-time* option for *QL Expert* which allows the creation and distribution of *QL Expert* rule bases. It includes stand-alone versions of both the rule editor and rule evaluation packages which are smaller than the integrated version making it possible to create and interrogate larger rule bases. You may also purchase a license to distribute the rule evaluation package with a rule base created by yourself. You must already have purchased the *QL Expert* package before the *run-time* option will be of use to you. Contact COMWARE for details if you are interested:-

Compware, 57 Repton Drive, Haslington, Crewe CW1 1SA.

2. Summary of Enhancements

QL Expert version 1.10 represents a significant upgrade to the original package. Apart from the fixing of some minor bugs, several areas have been radically improved and some totally new features have been added. The main changes are described below.

Forward Chaining

QL Expert now supports *forward chaining* which significantly improves its rule evaluation strategy by increasing the degree of deduction used to evaluate goals. It also reduces the number of questions put to the user.

Reporting

The reporting features of *QL Expert* have been extended and may also be used in ordinary rules (i.e. they are no longer restricted to *output* rules). New reporting commands include FREEZE, HOME, WARNING and ERROR. The range of substitution characters used in report strings has also been extended.

The EXPLAIN command has been added to allow the user to access an explanation of the significance of the questions he is being asked - i.e. the reasoning behind the evaluation of the current goal.

Query Session Enhancements

Commands given during a query session (e.g. HOW, WHY, TRACE etc.) are now given using the same method as those for the editor. (i.e. either as F3<command_letter> or by CTRL<command_letter>.) The command menu is also permanently displayed, rather than being called up only when F3 is pressed. These measures simplify the method of giving commands and eliminate the possibility of clashes between input to the query session and query session commands.

The following new commands are now available during a query session: EXPLAIN, REMEMBER, FORGET and PLIST. Each is explained in detail later.

Editor Enhancements

Some small changes have been made to the editor with the addition of a NEXT command and the ability to set goal values.

Reserved Goals

The following *reserved* goals can now be used within rules: TRACE, ECHO RULENUMBER, KEYCODE and DEDUCE. These are used for determining current user settings, the number of the rule being evaluated, and for waiting on a key-press. The DEDUCE command is used to invoke forward chaining reasoning mentioned earlier.

General Improvements

SET is a new keyword which is used to assign a value to a goal, or to clear its value.

TEST has been improved to provide information about the type of value assigned to a goal.

3. Revised Loading Procedure

Before describing the new features of the *QL Expert* program here are some notes about the files contained on the cartridge and the procedure for loading.

QL Expert has become such an extensive program that it has only just been possible to squeeze it onto a single microdrive cartridge. For this reason, the boot files required in order to run the program have not been supplied, but this is not a problem as they are generated automatically by the INSTALL program.

In order to use *QL Expert*, you should follow the procedure described below for backing up the cartridge and making a working copy.

3.1. Making A Backup

First and most importantly, make a backup copy of the program supplied by following the procedure described on page 7 of the main manual (section 3.1). Note though that you will need a cartridge with at least 218 good sectors in order to do this.

I.e. when you format the cartridge onto which you wish to copy *QL Expert*, and the format program prints a pair of numbers separated by a slash character as in "221/223", the first number must be at least 218. If not, the backup program will not have enough space to copy all the files onto the backup cartridge. If you can't format a cartridge with enough good sectors, you should split the files between two cartridges, and copy them manually using the COPY command described in your QL User Guide in the section about QL Superbasic. If you need to do this, you should split the files as follows:-

Cartridge 1: EXPERT, EXPERT_BIN, EXPERT_HOB, EXPERT_DAT, INSTALL.

Cartridge 2: BACKUP, CAR_EXP, HUMAN_EXP, QLCON_EXP.

This arrangement will allow you to make a working (installed) copy of *QL Expert* with the first cartridge alone.

3.2. Making A Working Copy

Having backed up all the files from the master cartridge, you can now make a working copy of the program (including the currently missing boot files: BOOT and JMBOOT) on either a floppy disc or a microdrive.

To do this, you need a formatted microdrive or disc with at least 210 good sectors. Place the blank medium in the drive from which you will normally want to load *QL Expert* (usually *mdv1_* or *flp1_*) and place your backup microdrive (cartridge 1 if split between two cartridges) in *mdv2_*.

Then type *lrn mdv2_install* and press the <ENTER> key. You should then follow the prompts from the program which are explained in some detail in section 3 of the main *QL Expert* manual. See also the notes below.

When the installation process is complete (which takes several minutes from microdrive to microdrive), you should put your master and backup copies in (preferably separate) safe places and can use the installed version to load *QL Expert* as described in the main manual. Note though that the example rule bases referred to in the manual are not copied onto your installed cartridge, and that you will need to load them from the backup cartridge when required.

NOTES:

The INSTALL program now includes two new options: TRACE and ECHO which affect the default state of the TRACE and ECHO (to printer) facilities.

Some of the initial default settings (e.g. for program workspace and number of recursion levels) are now different to those described in the main manual. This is to correct some problems when using *QL Expert* on a QL without expanded memory. The symptoms were either apparent locking up, or issue of a "RETRY" message when trying to access the on-line help file, or during access of a rule file. The symptom seen was dependent on the version of QL ROM. The cause of the problem is that on standard QLs, QDOS will allow a program to reserve too much memory, so that when the program asks QDOS to open a file, QDOS itself has not got enough memory available to complete the operation. On JM and earlier QLs, the result was a perpetually spinning microdrive, while on JS QLs, an error message is produced. The only solution is not to reserve too much memory which means that on a standard QL, you should not alter the default workspace setting. For expanded QLs however, QDOS does not seem to make this *mistake*.

4. Clarification: Pre-scanning Feature

This is not a new feature, but supplementary information about how *QL Expert* evaluates rules.

When invoked, a *define* rule is evaluated twice; the first time trying to evaluate the goal without asking for input from the user, and then, if it has been unable to give a value to the goal, a second time in the usual manner. For example, if we have the following rule:

```
DEFINE goal
one
two
OR
three
```

and 'three' has already been evaluated as TRUE (or it can be proved to be TRUE by pre-scanning its related *define* rule), no questions are asked and 'goal' becomes TRUE. The same thing happens if both 'one' and 'two' are TRUE, or if either of them is FALSE and 'three' is FALSE too (in which case 'goal' is obviously FALSE if there are no other rules defining it).

5. Reporting

This version of *QL Expert* contains very much more powerful reporting features:

All printing instructions, i.e. PRINT, REPORT, FPRINT and the new FREEZE, HOME, WARNING and ERROR (see below), are accepted in any kind of rule, not just *output* rules. They can be inserted in any position, with the following exceptions:

- a) not in any line after the PROMPT clause in an *input* rule;
- b) not in any line after the CONDITION or FIRST OF, ALL OF, MAX OF or MIN OF clauses in a selective *define* rule;
- c) of course not in the first line of the rule.

5.1. Reporting During Pre-scanning

Note that printing instructions given in *define* rules will not be executed if the rule is being pre-scanned. For instance, the rule

```
DEFINE goal
PRINT I am evaluating the main goal
one
two
PRINT The session is over
```

will print nothing if both 'one' and 'two' are known, or either of them is false, since in these cases 'goal' can be evaluated without asking any questions of the user. The exception is the REPORT instruction, which is executed ONLY when pre-scanning. For instance

```
DEFINE goal
REPORT I am trying to evaluate 'goal'...
PRINT I need help. Please answer the following questions:
one
two
```

will inform on what it is about to do, but ask for input only if necessary. Note that when not used in an *output* rule, the REPORT instruction does not cause the rule to be exited.

5.2. Conditional Reporting

Note also that the IF clause may be used in any kind of rule, but is intended solely as an aid to reporting. When used in *input* rules or ordinary or selective *define* rules, it can be followed only by a printing command. The editor will signal an error in any other case.

5.3. Undefined Goals

One of the most important improvements is that messages printed by PRINT, REPORT and the new WARNING instruction may contain any goal's value (see next paragraph). It is important to keep in mind that such a value need not to be known by the system at the time the printing instruction is executed provided that it may be evaluated by pre-scanning its related *define* rule, i.e. without asking anything else of the user.

QL Expert also invokes *output* rules correctly for undefined goals. This means that the programmer can ensure that *output* rules work as intended by performing relevant tests. For example, it is recommended that each *output* rule start with a conditional statement:

```
OUTPUT goal
IF TEST(goal)=0
REPORT Data is insufficient for this task
PRINT .....
```


5.4. FPRINT Reports

FPRINT files accept both ESCAPE (ASCII 27) and "@" (ASCII 64) characters as page delimiters. This allows you to create and edit such files with any screen editor which can produce ASCII files (e.g. Quill by printing a document to a *_lis* file or with an ordinary editor such as those supplied with most programming languages). Such characters should be at the beginning of a line, and should be followed immediately by a carriage return.

5.5. Substitution Characters In Report Strings

The following is the complete list of special characters that may appear in messages printed by PRINT, REPORT and the new WARNING instructions (see below). It includes both new and existing features.

- ;
If the message ends with a semicolon a carriage return will not be issued after printing the string (the semicolon is not printed). Note that WARNING instructions treat this character differently.
- If the message ends with one or more tilde characters, each one of them causes an additional carriage return to be printed. WARNING instructions treat this character in a different way.
- \
As in previous versions of *QL Expert* the word which follows a backslash character is printed only if the value of the current goal (i.e. the goal to which the current rule refers) is false. If the word contains blanks they should be converted to underscores (as in expressions used in EVAL, RETURN or IF clauses). Spaces or other non-alphanumeric characters mark the end of the word implicitly.
- @
When on its own, is replaced by the value of the current goal, be it a string or a number. Note that values of 0 and 1 will be shown as FALSE and TRUE respectively. A value of "?" will be printed as UNDEFINED. Since reporting instructions can be included in *define* and *input* rules, a goal value may be printed before it has been evaluated, and in this case, the "@" will be replaced by "UNKNOWN".

An "@" character may be followed immediately by a goal name, in which case that goal's value is used instead. Goal names follow the same restrictions as for backslash characters.
- @@
is a special case of the previous one. Since characters are replaced from right to left, the rightmost one is first replaced by the referred goal's value, which is then used as an argument for the leftmost "@". The effect is therefore similar to "###" (i.e. the value of the goal which is the value of a string-typed goal), but values are expanded as explained before.
- ##
works as in previous versions of *QL Expert* but may now be followed (optionally) by a goal name, and the returned numeric value is related to that goal. Note that *undefined* or *unknown* values are not shown at all, so if in doubt you should TEST that goal first, as in

```
OUTPUT goal
IF TEST(goal)=0
REPORT The goal cannot be evaluated.
PRINT The result is ##
```

%% works like "##", except that values are shown as percentages.

&& is replaced by the number of values of the current goal. If the goal is numeric it obviously has only one value, but it may have more values if it is the result of an ALL OF clause. Undefined values return zero. For example

```
OUTPUT goal
PRINT Your main goal has && values :
PRINT @, scoring %%
```

If "&&" is followed by a goal name, it will be replaced by the number of values of that goal.

\$\$ is replaced by an "s" (lower-case) character if the numeric value of the current goal is not one. This makes it possible to express plurals correctly, for example

```
OUTPUT cars
PRINT Your firm should buy ## car$$.
```

When "\$\$" is followed by the name of another goal, the test is made against that goal's value. Since "&&" characters are replaced before any "\$\$", the sequence "\$\$&&" will insert an "s" if the number of values of current goal (or any other goal) is not one. Therefore a previous example could be better re-written as:

```
OUTPUT goal
PRINT Your main goal has && value$$&& :
PRINT @, scoring %%
```

** is replaced by a number of asterisks equal to the (integer) value of the current goal (or any other goal, if followed by a goal's name), up to 76 characters. Values less than one show no asterisk. This can be used to build a histogram related to evaluated goals, as in the following example:

```
OUTPUT car sold
PRINT Number of cars sold in past four years :
PRINT 1984 **year84
PRINT 1985 **year85
PRINT 1986 **year86
PRINT 1987 **year87
```

where *year84*, *year85* etc. have already been calculated or entered by the user. Remember that the line width is 76 characters, so take care to adjust your values accordingly.

6. New Keywords

The programming language supported by *QL Expert* has been expanded with the addition of eight new keywords.

6.1. FREEZE

This keyword allows a limited windowing capability. Its syntax is:

FREEZE <n>

where <n> is an integer in the range [0,16]. Its effect is to reduce the active window by 'freezing' the upper <n> lines. This enables useful information to be held on the screen, preventing it from scrolling out of view during a query session. In *define* rules this instruction is executed only when not pre-scanning. The new active window is automatically cleared. Use 'FREEZE 0' to use the entire screen again.

6.2. HOME

Clears the currently active window, without changing its dimensions. In *define* rules it is executed only when not pre-scanning. No arguments are allowed.

6.3. WARNING

Prints a message in the bottom line of the window (the one where *QL Expert* displays its menus). The syntax is:

WARNING <message>

where <message> may contain substitution characters as in PRINT and REPORT instructions. There are however a few special cases:

- a) if <message> ends with ~ (tilde), it is printed and the program stops until the user presses a key. After which the bottom line will be cleared and execution continues. If <message> contains no other characters, the default string "Press any key to continue" is used instead.
- b) if <message> contains only blanks the bottom window is cleared and elaboration continues.
- c) if <message> ends with one or more ";" (semicolon) characters, it is printed and the program pauses for that number of seconds.
- d) if <message> ends with one or more ";" followed by one "~" (tilde), the message is shown, *QL Expert* will pause for as many seconds as the number of semicolons, and then the bottom line will be cleared.

In all cases semicolon and tilde characters will not be printed unless they are not at the end of <message>. Note that the bottom line is always cleared when the program puts a question to the user. WARNING instructions in *define* rules are executed only when not pre-scanning.

6.4. ERROR

Prints a message and aborts the query session, exactly as if a QUIT command were issued by the user. Its syntax is

ERROR <message>

No character substitution is performed on <message>. If it is encountered when pre-scanning a *define* rule, it does not abort the session. This instruction may be useful in many ways.

Suppose that 'temperature' must be known for the goal to be evaluated. We might write this rule

```
EVAL temperature<0
IF TEST(temperature)=0
ERROR Evaluation can't continue
RETURN temperature<0
```

6.5. EXPLAIN

Is an alternate form of the OUTPUT keyword. Its syntax is

```
EXPLAIN <goalname>
```

and must appear in the first line of the rule. Such *explain* rules are in all aspects similar to *output* rules, but they are invoked upon a user's request during a query session (see below). They could be used to explain the real meaning of a goal, or the reason for which it is being evaluated (note that it is different from the WHY command, which simply shows which goal depends on the current one).

For instance we may have

```
INPUT age
PROMPT How old is the patient?
OPTIONS >0,<120

EXPLAIN age
PRINT The age of the patient is important in order to
PRINT evaluate the risk when exposed to a given class
PRINT of medicines.
```

Note that you are not restricted to EXPLAIN'ing goals at the lowest level in the goal tree (i.e. those being asked to the user), since the user may ask to explain any goal, for instance goals shown during a WHY request.

6.6. REMEMBER

This command has no arguments. Its effect is like putting a "mark" at the current state of the goal values area. When a new query session is started and a REMEMBER is active, the memory is not entirely cleared as it would usually be, but all goals evaluated before the REMEMBER was found are retained, whereas others are simply "forgotten". This keyword may appear in *define* rules only, wherever a PRINT instruction would be accepted. If it is executed during a "what if" session (as shown by the upper status line), only values evaluated before the first WHAT IF command will be "remembered".

6.7. FORGET

Has no arguments. It may appear only in *define* rules, wherever a PRINT instruction would be accepted. It clears the goal value memory and undoes the effect of a previous REMEMBER.

This provides a neat way to execute multiple sessions with some "default" values for a set of goals. For instance

```
DEFINE start query
FORGET
EVAL defaults=0
REMEMBER

DEFINE defaults
ALL OF one,two,three,four,five
```

The very first time that the rule base is questioned you should evaluate "start query", which in turn activates "default" and causes the values of 'one', 'two'... to be entered by the user. The REMEMBER will protect such values from being cleared when the session is started again. If you need to give different values to these goals, you should re-evaluate "start query": the FORGET instruction will undo the effect of the previous REMEMBER and 'one', 'two'... will be asked again. Note that "start query" is a dummy goal used just to activate "defaults", since selective *define* rules cannot contain REMEMBER or FORGET keywords after the ALL OF instruction. Both FORGET and REMEMBER may be invoked directly from editor mode (see VALUE command).

6.8. SET

Allows you to assign a value to a goal. It works of whether a rule is being pre-scanned or not. Two syntaxes are acceptable:

- (a) SET <goalname>
- (b) SET <goalname>=<value>

Form (a) deletes any previous value of the goal, whereas (b) assigns a value to the goal. <value> is taken literally, so you should use 0 for no/false, 1 for yes/true and ? for undefined. Note that if <goalname> already had a value, using set will cause it to be ignored: this means that memory is not de-allocated, and that if the previous value was "protected" against clearing by a subsequent REMEMBER instruction, the new value will not be remembered unless another REMEMBER is executed afterwards. The SET instruction allows you to write simple user-defined functions with very little effort. For example :

```

DEFINE age_in_days
RETURN current_giulian_date-birth_giulian_date

```

```

DEFINE current_giulian_date
SET day=current_day
SET month=current_month
SET year=current_year
RETURN giulian_date

```

```

DEFINE birth_giulian_date
SET day=birth_date
SET month=birth_month
SET year=birth_year
RETURN giulian_date

```

which allows you to call a *define* rule for "giulian_date" (not shown) with two different sets of values for its implicit arguments "day", "month" and "year". If it can be guaranteed that all dates are to be asked of the user, another simple method can be adopted:

```

DEFINE age_in_days
RETURN current_giulian_date-birth_giulian_date

```

```

DEFINE current_giulian_date
PRINT Please enter today's date.
RETURN giulian_date

```

```

DEFINE birth_giulian_date
PRINT Please enter birth's date.
RETURN giulian_date

```

```

DEFINE giulian_date
SET day
SET month
SET year
RETURN .....

```

where the SET instructions are used to delete previous values, so that they will be asked of the user when evaluated in the RETURN expression.

7. Reserved Goals

QL Expert now includes five reserved goals. These are goals which are always defined, and can be used to obtain various types of information, or to invoke a particular action:-

7.1. TRACE

Returns 1 if trace is active, 0 if not. It may be assigned to (using a SET instruction), allowing trace control from within a program.

7.2. ECHO

Returns 1 if echo to printer is active, 0 if not. This too can be assigned to (using a SET instruction), allowing echo control from within a program.

7.3. RULENUMBER

Returns the rule number of the *define* rule in execution. This may be useful within *input* or *explain* rules for printing different messages depending on the *define* rule causing the *input* or *explain* rule to be invoked. Assignments to this goal are ignored.

7.4. KEYCODE

When this goal is assigned a string value, it stops the program and waits for the user to press any one of the keys whose corresponding letter appears in the string. For example:

```
SET keycode=ABCD
```

will wait for the user to press A, B, C or D key, and ignore all others. The bottom line of the *QL Expert* display will also be cleared. From then on, whenever the KEYCODE goal is invoked from within a rule, it returns the ASCII code of the key pressed (converted to upper-case). This feature, together with PRINT and WARNING instructions, makes it possible to build menus for input from the user:-

```
DEFINE smoking habits
PRINT How many cigarettes do you smoke each day ?
PRINT (0) I don't smoke
PRINT (1) Less than 5 cigarettes
PRINT (2) From 6 to 15 cigarettes
PRINT (3) More than 15 cigarettes
WARNING Press a key in the range 0-3
SET keycode=0123
RETURN keycode
```

7.5. DEDUCE

This keyword enables forward chaining reasoning, one of the most powerful new features of *QL Expert*. When invoked, either from within a rule or directly from editor mode, it causes *QL Expert* to make all the deductions it can from current goal values without asking for further input from the user. If DEDUCE is invoked from the editor (by typing it or by pressing F5), all results are shown as they are reached, whereas if it is invoked from within a rule, deductions are shown only if trace is active. The value of the goal "DEDUCE" is TRUE if it succeeded in making at least one deduction, FALSE otherwise. If invoked from the keyboard, memory is never cleared since this would make little sense, and its final value (i.e. TRUE or FALSE) is never shown. This feature greatly improves the *intelligence* of *QL Expert*.

As an example, suppose we have *define* rules for many diseases. The user can enter the symptoms he has detected, and let the program make the diagnosis by repeated deduction until DEDUCE is false.

8. Multiple OPTION Lines

QL Expert now allows any number of OPTION lines after a PROMPT clause in an INPUT rule. However, the total length of option values should not exceed 256 characters, or some of them will be lost.

9. Inputting Values From The Editor

Goal values can now be entered from within the editor using the syntax shown below:

`<goalname>=<value>`

`<value>` is taken literally, so boolean values should be entered as 0 or 1, and undefined values as "?" (see SET instruction).

10. Query Session Commands

During query sessions, user commands must now be entered in the same way as editor commands, i.e. by pressing F3 and then choosing from the displayed menu, or directly by pressing the CTRL key together with the first letter of the command itself.

A new command has been added, EXPLAIN, which causes the program to search and execute the EXPLAIN rule related to the desired goal, or print a default message if no such rule exists. The command prompts for a goal name in the bottom window, taking the current goal as a default.

The HOW command prompts for a goal name in the bottom window, all other commands execute immediately.

The change in the way commands are invoked reduces clashes with expected values. Now the only special answers are YES, NO, TRUE, FALSE and UNDEFINED. All of which may be entered in abbreviated form (e.g. Y for YES etc.), provided that the current option setting does not contain the "\$" character (allowing any string as input), in which case they must be typed in full.

11. The Bottom Line

The bottom line of the *QL Expert* window is now always active, showing the meaning of all five function keys. Previously, it was only displayed after pressing F3 to enter a command.

12. The VALUE Command

The VALUE command now has a sub-menu which facilitates:

CLEAR'ing

of goal value memory. However, if a REMEMBER is active, it clears only the goals evaluated after the REMEMBER was encountered, otherwise it prints a "memory cleared" message.

FORGET'ing

of all goal values, and de-activation of any previous REMEMBER command. This always prints a "memory cleared" message.

LIST'ing
of all goal values to the screen.

PLIST
to list all all values to screen and printer.

REMEMBER'ing
of the current state of goal value memory. When a new query session is started all such values will be "remembered" and not asked for or re-evaluated again. A **REMEMBER** command remains effective until a **FORGET** command is executed (from keyboard or program), or the rule base is edited.

12.1. New DESIGN And RULE Options

Two new options are available in the **DESIGN** menu:

TRACE
switches trace on and off.

PRINTER
switches the printer on and off line.

In previous versions of *QL Expert* it was only possible to alter these features during a query session.

The **SEARCH RULE** command now has the option to delete the rule shown on screen.

Note that rule searching is much faster when looking for **DEFINE**, **OUTPUT**, **INPUT** and **EXPLAIN** keywords followed by a goal name, but in these cases it searches for an exact match, and not for any line that contains the given string. For example, if you need to locate **DEFINE** rules for all goals whose name starts with the "doc" sequence you should specify the string "**EFINE doc**": This disables the "exact match" feature and causes the search to include sub-strings, although the search will take longer.

13. New Screen Editor Commands

EXIT now has the same effect as the **ESCape** key, i.e. the rule is exited without storing and auto mode (if active) is exited too.

The (new) **NEXT** command stores current rule and goes to the next one in the rule base.

14. The TEST Function

The **TEST** function now returns information on the type of the value of a goal:

- 1 is returned if the goal's value is numeric
- 0 is returned if the goal is undefined (zero values)
- 1 is returned if the goal's value is a string
- 2 is returned if the goal is a multiple valued (two values) and so on.

15. The INSTALL Program

The INSTALL program now has two new items allowing the default starting states of TRACE and ECHO to be set.

15.1. Error Messages

Note that the following error messages have changed:

- 5 **SYNTAX ERROR : this keyword cannot stand alone**
appears when a keyword other than OR, HOME, REMEMBER and FORGET is not followed by any other character.
- 6 **SYNTAX ERROR : this keyword must be followed by one blank**
may appear with any keyword but PRINT, REPORT, WARNING, ERROR and those covered by the next error message.
- 9 **SYNTAX ERROR : this keyword cannot be followed by anything**
when an OR, HOME, REMEMBER or FORGET instruction is followed by any other character.

The following new error messages have been added:

- 62 **No help is available**
During a query session the user asked to EXPLAIN a goal which has no corresponding EXPLAIN rule.
- 63 **No (more) deductions are possible**
end of DEDUCE evaluation.
- 64 **SYNTAX ERROR : not allowed after IF clauses**
only RETURN and printing instructions may appear after IF statements.

Note that Error 54 (Return to Superbasic) has been removed.

16. QL Expert Manual Errata (13th December 1987)

The following modifications apply to issue 1 of the *QL Expert* manual dated December 7th 1987 (see page 1 of your manual to find what issue it is).

16.1. Customising Default OPTIONS

At several points in the manual it is stated that you should use the INSTALL program to alter the default options used for checking the validity of input in the absence of an *input* rule. The INSTALL program does not perform this function.

Instead, you should configure the options required using the OPTIONS command of the DESIGN menu (page 33). When you save a rule base, the current options setting is saved with it, and will be resumed whenever the rule base is loaded.

When *QL Expert* is first started, the default options are always `&,%` (i.e. an uncertainty or boolean value: 0 to 1, yes, no, true or false), but whenever a rule base is loaded the default input options will be altered to the options current when the rule base was saved.

16.2. Boolean Input Options (Rule example, page 48)

The example (rule 180) shows an options statement as follows:-

```
OPTIONS yes,no
```

This should now read

```
OPTIONS &
```

The `'&'` indicates that boolean input is required and will allow any of the following inputs to be accepted:

```
yes,no,true,false
```

Where no ambiguity exists in a list of explicitly stated options, the above may be shortened to

```
y,n,t,f
```

If the options statement specifies these strings explicitly, as shown in the manual, they will be treated as string values (without boolean meaning) and so will generate an error if a rule subsequently uses them as booleans.