Instruction Manual

for the

QL-488 IEEE Interface.

TABLE OF CONTENTS.

Page number

1. INTRODUCTION.....1-1

2. THE IEEE-488 STANDARD INSTRUMENT BUS.

2.1 Introduction.....2-1

2.1.1 Benefits of a Standard Interface.....2-1
2.1.2 IEEE-488 Instruments Available.....2-1
2.1.3 Other Interfaces Closely Related to the IEEE-488.....2-1

2.2 The IEEE-488 Standard.....2-2

2.2.1 Subsets of the Full IEEE-488 Standard.....2-2
2.2.2 The Signal lines of the IEEE-488 Interface.....2-2
2.2.3 Bus Commands and Controllers.....2-3
2.2.4 Addressing Commands.....2-4
2.2.5 Other Bus Commands - Polling and Triggering.....2-5
2.2.6 Other Commands - Transfer of Control.....2-6
2.2.7 Control Lines and Handshaking.....2-7
2.2.8 Logic Levels and Open Collector Drivers.....2-7
2.2.9 Handshaking in Detail.....2-8
2.2.10 Bad Bus States.....2-9

3. INSTALLATION.

3.1 Installing the QL-488 Board.....3-1

3.1.1 Accessing the QL Expansion Port.....3-1
3.1.2 Using a QL Expansion Unit.....3-1
3.1.3 Powering Up.....3-1
3.1.4 In Case of Fault.....3-2
3.1.5 A Simple Test Example.....3-2
3.1.6 Added SuperBASIC Keywords.....3-3

3.2 Connecting the Interface to Peripheral Devices.....3-3

3.2.1 Connecting Cables.....3-3
3.2.2 Setting Device Addresses.....3-4
3.2.3 Connecting Devices.....3-4

3.3 Testing the Bus.....3-4

4. USING THE Q-488 INTERFACE.

4.1 Introduction.....4-1

4.1.1 The Scope of this Section.....4-1
4.1.2 Setting up the System.....4-1

4.2 Principles of Operation.....4-1

4.2.1 SuperBASIC Extensions.....4-1
4.2.2 The IEEE Device.....4-2
4.2.3 Opening and Closing Channels.....4-2
4.2.4 Choice of Channel Number.....4-3
4.2.5 Addressing IEEE Devices.....4-4

4.3 Transferring Data over the IEEE Bus.....4-4

4.3.1 Sending Data to IEEE Devices.....4-4
4.3.2 Receiving Data from IEEE Devices.....4-6
4.3.3 SuperBASIC Extensions Requiring Channels.....4-7

4.4 Other Commands.....4-7

5. ADVANCED USE.

5.1 Advanced Use from SuperBASIC

5.2 Use with Other Languages

5.3 Use in "Device" Mode

6. THE BUS ANALYSER.

6.1 Purpose of the Q-488 Analyser.....6-1

6.1.1 Introduction.....6-1
6.1.2 Outline of Operation.....6-1
6.1.3 A Quick Demonstration.....6-2

6.2 The Q-488 Bus Analyser.....6-2

6.2.1 Activating the Analyser.....6-2
6.2.2 The Display Window.....6-3
6.2.3 Activating the Input Cursor.....6-4
6.2.4 Moving the Display Window.....6-4
6.2.5 The Function Keys.....6-4

7. REFERENCE.

7.1 Introduction.....7-1

7.2 The Standard System.....7-1

7.2.1 Syntax of Q-488 Channel Names.....7-1
7.2.2 SuperBASIC Commands used by the Standard System.....7-4

7.3 Low Level Commands.....7-4

7.3.1 I/O Trap for Low Level Commands.....7-4
7.3.2 Summary of Low Level Commands.....7-5
7.3.3 Low Level Command Definitions.....7-6

7.4 Bus Analysis Functions.....7-23

Acknowledgements

We would like to thank the following people for their help in this project:

Mark Horton, Guy Jennings, Martin Brown, Simon Mentha and Richard Barber of Procyon Research Ltd. for the design and continuing development of the firmware, and the writing of the manual.

David Oliver and Martin Baines of Cambridge Systems Technology for the hardware design, software consultation, management and marketing of the product.

Vic Oliver of C.V.O. Electronics for aid with production and procurement.

Plus others too numerous to mention all of whom have made essential contributions to the success of this project.

Disclaimer

Important

Please note that neither Sinclair Research nor CST can undertake any commitment to advise or assist users beyond replacing faulty goods.

Users with specialised requirements are invited to contact Cambridge Systems Technology or Procyon Research Limited directly for information on customised systems.

KNOWN BUGS AND PROBLEMS

This section lists the known bugs which have come to light in released versions of the Q-488 system. Please do not hesitate to contact us with further problems you may come across. Wherever possible, we have indicated methods of avoiding the bugs in such a way that programs will still work with upgraded versions of the Q-488 system.

Version 1.00 Bugs

The BASIC function IB_SPOLL does not unaddress the polled device after it has been polled. Be careful always to explicitly unaddress the device with a call to IB_UNADDR after performing a serial poll. This will still be legitimate when the function has been corrected, the second unaddress call will then have no further effect.

2. The BASIC command IB_STERM does not accept arguments as indicated in the manual. It only takes a single argument which is passed directly to the CM.STERM trap call and which should contain four terminator characters packed into a long word. It will not be possible to maintain compatibility with an upgraded system for this command.

   Note that the remaining output terminators are still saved internally if you change the number of terminators, and by default are set up to CR and LF you will be able to choose most combinations merely by changing the number of output terminators.

3. Parallel polling is not implemented in Version 1.00.

4. The IO.SSTRG (Send String) trap call is incorrect in that it takes a Long Word length in D2.L (rather than a 16-bit word, D2.W). Therefore if the upper half of D2 is non-zero, the wrong number of bytes will be output. Assembly language programmers should take care to clear D2 before loading the length, or to sign-extend it to a long word before calling the trap.

   This bug does not seem to cause any trouble with SuperBasic, but other language systems may give problems. Please notify us if you have any problems in this area.

SECTION 1 - INTRODUCTION

This manual describes how to use the Q-488 IEEE interface for the Sinclair QL. Examples are given throughout the text to help the user become familiar with this versatile piece of equipment. Newcomers to the IEEE bus are especially recommended to read the following section outlining the concepts involved. Because of the particularly full implementation of the IEEE-488 1978 standard, the more experienced programmer should be able to make good use of the advanced features provided; and yet even with only a limited knowledge, it is easy for the novice to make a start.

A note about examples: items in **bold** type indicate the user's input to the QL. It is assumed that lines are terminated by **<ENTER>**. Hyphenation implies simultaneous operation, e.g. **<CTRL>-<SPACE>**.

Users with particular requirements or enquiries are referred to the comment at the foot of the Acknowledgements.

SECTION 2 - THE IEEE-488 STANDARD INSTRUMENT BUS

2.1 INTRODUCTION

2.1.1 Benefits of a Standard Interface.

The IEEE-488 interface provides a standardised input/output interface between a computer and up to 14 other instruments and peripherals. It allows digital data to be transferred between the instruments and the computer in any combination. In addition, it is possible for computers with IEEE-488 interfaces to be connected to each other and the interface then allows very rapid transfers of data between the computers. Data transfer rates of 1 million bytes per second are feasible.

2.1.2 IEEE-488 Instruments Available.

The range of instruments available is very large, and includes such diverse items as microwave signal generators, modems, digital voltmeters, disc drives, printers and plotters. Any computer with an IEEE-488 interface can connect to any of these pieces of equipment without any problems of incompatibility.

2.1.3 Other Interfaces Closely Related to the IEEE-488.

The IEEE-488 interface may also be known as the "General Purpose Interface Bus", GPIB, or as the "Hewlett-Packard Instrument Bus", HPIB. Instruments with these interfaces are fully compatible with the IEEE-488 Interface.

Instruments following the IEC-625 standard use a different connector to the IEEE and ANSI standards, but are otherwise the same. They can be used on an IEEE-488 system provided that suitable adaptors are used. These are available from more specialised dealers.

2.2 THE IEEE-488 STANDARD

2.2.1 Subsets of the Full IEEE-488 Standard.

The standard allows considerable freedom to the device designer to implement subsets of the full standard. Thus there is no guarantee that a particular device will implement all the interface functions described in this chapter. This is sensible in measuring devices where many of the features of the standard would be nonsensical, but for controllers it is less so. Most controllers implement a large proportion of the standard and are thus very versatile in a large number of applications. The main areas where subsets of the full standard occur in controllers are in the multiple controller facilities. Many IEEE-488 controllers insist on being the only controller in a system. The PET and Apple interfaces, for example, must be the only controller in the system. The Q-488 IEEE interface is not restricted in this way and provides full support for multiple controller systems.

2.2.2 The Signal Lines of the IEEE-488 Interface.

The IEEE-488 interface bus uses eight data lines, eight control lines and eight ground lines. The ground lines form twisted pairs with each of the control lines to minimise radiated noise and cross-talk between the asynchronous lines. The bus connections are made in parallel to every device on the system. The standard specifies connectors which are stackable to allow many cables to be connected at a given instrument. Standard IEEE-488 cables with the correct connectors are available from many suppliers. We recommend that you use only these cables.

When connecting the devices in a system together with the standard cables it is important to avoid making loops with the cables as this can cause interference problems. The recommended cable configurations are:

Linear: Connect all the devices in a chain, one after the other

Star: Connect all the devices directly to the controller, one by one.

Apart from this there are few constraints on the cable layout. You must however stick to a maximum total cable length of two metres times the number of devices you have, or twenty metres overall, whichever is shortest. So if you have 5 devices you are limited to 10 metres overall and if you have 13 devices you are limited to 20 metres. This will ensure that the bus can run properly at full speed. There is no limit to the length of individual cables that you can use so long as the total length is not exceeded. In practice pre-made cables are only readily available up to 2 metres in length. Using longer cables than this is to be avoided. Special devices are available to extend IEEE-488 cables if this distance is insufficient.

2.2.3 Bus Commands and Controllers.

When a data byte is being sent on the bus the byte travels from a device described as the source to a device (or devices) described as the acceptors. A particular device need not always be a source or an acceptor. Handshaking is used on all data bytes sent to ensure that all acceptors receive the data properly. The details of handshaking are described in a subsequent section.

At any particular time there is a special device, designated the active controller, which is allowed to issue commands to the other devices on the bus. These commands can tell other devices to become sources or acceptors ("talkers" or "listeners"), or to ignore their front panel controls; or may have a variety of other effects.

The commands consist of single bytes which are sent with handshaking in the same way as data bytes. They are distinguished from data bytes by the control line "Attention" (ATN) which is set by the controller to indicate that commands are being sent. Whenever ATN is true all other devices on the bus must listen to the current controller which will send commands to them. All devices in a system except the active controller accept these commands.

2.2.4 Addressing Commands.

The most important commands are those known as "Addressing Commands". These are used by the controller to indicate which devices are to take part in future data transfers. Each device is assigned a unique address in the range 0 to 30 which (usually) remains constant throughout operation of the system. With most instruments, the assignment is made by switches on their rear panel. Some devices may have fixed addresses when they are switched on, but may be changed by software intervention.

The addressing commands fall into three groups: talker, listener and secondary address. Each command is followed by the number of the device which is being addressed. This must lie between 0 and 30 to be a valid address.

A talker address is a command to tell a device to become a talker when the controller decides to release ATN. Only one device may be a talker at any one time: it is said to be "addressed to talk". If a talker address command is issued for another device then the first device must be told to stop talking: it is unaddressed.

The standard mnemonic for this command is "My Talk Address" (MTA). So the shorthand for the command to tell device six to talk is:

MTA 6

MTA 31 represents a special case. No device is allowed to have this address and if it received it will cause all talkers (there should only be one!) to be unaddressed. It is given a special name: Untalk (UNT).

A listener address is a command to tell devices to listen to data that will be sent when the controller releases ATN. More than one device may be a listener at any one time so if a device receives a listen address other than its own it does not unaddress. The mnemonic is "My Listen Address" (MLA). Again a listen address of 31 is a special case and causes all addressed listeners to unaddress. This is called "Unlisten" (UNL). Using UNL is the simplest way to cause a listener to be unaddressed. If a device which has been addressed as a listener is subsequently addressed as a talker then the listener will (usually) unaddress and the device will be addressed as a talker only. This prevents a device from being asked to talk and listen simultaneously!

A secondary address may be sent immediately after a talker or a listener address. They are used where a device has several possible internal functions which may be selected via their secondary addresses. There is no difference between secondary addresses for talkers and for listeners. They have values in the range 0 to 30, as before. Secondary address 31 has no special function, but should not be used. A typical example of the use of secondary addresses is shown by CBM disc drives where they are used to refer to individual files on the disc when several are open simultaneously.

The general format of a data transfer on the bus consists of:

a group of addressing commands, which select which device is to talk, and which are to listen.

a group of data bytes sent by the source to the acceptors, with the bus control line "End Or Identify" (EOI) set true on the last byte.

a group of commands to unaddress talkers and listeners.

2.2.5 Other Bus Commands - Polling and Triggering.

As well as the addressing commands there are a number of other commands available on an IEEE Interface system. The most widely used are the commands relating to the serial poll.

Serial polling is a method whereby the controller on a bus system can ask a device its current status. This is accomplished by a command called "Serial Poll Enable" (SPE). This is received by all the devices and puts the bus into what is known as serial poll mode. In this state, individual devices may be addressed as talkers and they will send a single byte status word to the controller. When all the desired devices have been polled the controller issues the "Serial Poll Disable" (SPD) command to put the bus back into the normal operating mode.

Not all devices are capable of responding to a serial poll so you should check in the device manual before attempting to use serial polls.

There is another form of polling used by a some IEEE devices called the parallel poll. This is much faster than serial polling and works by allowing devices to put one bit of status information onto a single data bus line; different devices are told to use different data lines. Then the controller can read back status information from up to eight devices simultaneously. Very few devices allow parallel polling however, so its use is not very common.

In a system with several measuring instruments it is often desirable to have them all take a reading simultaneously. The IEEE bus command "Group Execute Trigger" (GET) can

be used to achieve this. The command causes all devices which have been addressed as listeners to be triggered simultaneously. Devices which are not addressed as listeners will not be triggered.

It is often desirable that a device which has finished doing some task should be able to interrupt the normal operation of the controller. This is accomplished by the "Service Request" (SRQ) function of the IEEE bus.

The bus line called "Service Request" (SRQ) can be driven by any device on the bus to indicate a request for some attention from the controller. When it sees that the SRQ line has been activated, the controller executes a serial poll of devices which may have caused the request. The source of the request can be recognised as it will be the only device polled to have bit 6 set in its status byte. The IEEE standard defines bit 6 of the status word to have this special significance. Reading the status normally causes the device to clear the request.

"Service Request" is often implemented as an interrupt to the controller so there can be a speedy response. However, it is perfectly admissible for the controller to ignore the request altogether.

2.2.6 Other Commands - Transfer of Control.

In a bus system, it is useful to be able to specify which devices may be controlled by their front panel controls and which may not. This guards against accidentally disturbing the settings of controls while measurements are in progress.

IEEE devices have two states of operation as far as their front panel controls are concerned: local and remote state. A device which is in "local control state" may be controlled by its front panel controls but not by the IEEE bus, and one which is in "remote control state" will ignore changes to its front panel controls and only respond to the IEEE bus. There may also be controls to force a device to go into local control.

To put a device into remote control state it is simply addressed as a listener. It will remain there even after it is unaddressed. It will only go to local state when the signal line "Remote Enable" (REN) is de-activated. An individual device may be put into local state by addressing it to listen and sending the "Go To Local" (GTL) command to it. Devices often have an indicator on their front panels which lights up when they are in the remote state.

There may be more than one device in a system capable of being a controller but only one device may be the controller at any particular time. There is a command "Transfer Control" (TCT) which allows the current controller to nominate another device as controller and allow it to control the bus thereafter. The system controller may regain control of the bus at any time by activating the "Interface Clear" (IFC) line. Only one device in a system may be the system controller. It is usually the first interface that powers up and asserts the "Remote Enable" (REN) line. More modern controllers can sense this line when waking up, and avoid a clash if there already happens to be a (system) controller on the bus. Be wary of older IEEE interfaces: they may only be capable of being the system controller.

2.2.7 Control Lines and Handshaking.

Out of the eight control lines in the interface, three are used exclusively for handshaking. These are called "Ready for Data" (RFD), "Data Valid" (DAV) and "Data Accepted" (DAC). (RFD and DAC are properly called NRFD and NDAC to indicate that they are in negative logic but we will continue to call them RFD and DAC). The handshaking ensures that data transfers take place at the speed of the slowest device involved in them, so that no data bytes are lost.

The other lines are used for a variety of functions. The line Interface Clear (IFC) is a general reset line which, when activated, restores equipment on the interface bus to a standard quiescent state. Only the system controller is allowed to use IFC.

2.2.8 Logic Levels and Open Collector drivers.

The signal lines on the IEEE bus are defined to be at the logic value TRUE when the voltage on them is less than about 0.5 Volts and FALSE when greater than about 2 Volts: this is negative logic. The circuits driving the bus lines are also open-collector drivers. This allows several "output" lines to be connected together in a wired-OR configuration. Thus if any device on the bus drives a signal line TRUE (to a LOW voltage) then the signal line will immediately assume that value whereas it requires all devices collected to the bus to drive the line high before the line will go to a high voltage (or FALSE logic value). This fact is crucial to the operation of the IEEE bus. Some more modern drivers use tri-state outputs which results in greater speed at the expense of increased complexity. It is necessary to switch the tri-state drivers to open-collector operation during certain bus operations. The Q-488 IEEE interface uses tri-state buffers. The switching is handled automatically.

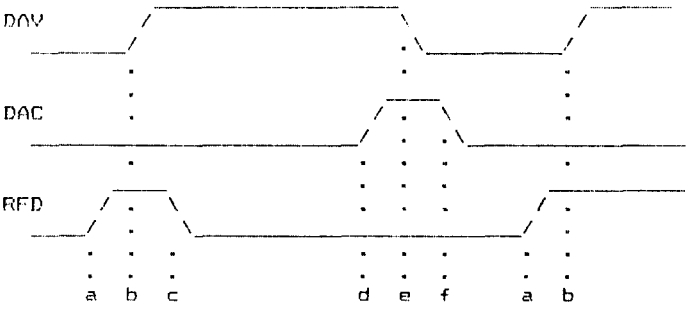
Several of the signal lines on the IEEE bus have an additional level of inversion present. These are the signals where proper bus operation requires that a wire-AND operation be applied. In these cases the signal level is inverted and wire-ORed. These lines are identified by

names starting with the letter N, e.g. "Not Ready For Data" (NRFD). In this case the name RFD would be frequently used to refer to the signal "Ready For Data", although this signal never appears as a physical signal on the bus lines.

2.2.9 Handshaking in Detail.

The three-wire handshaking on the IEEE-488 bus is applied to all data transfers and most command transfers. The only exception is in the parallel poll sequence which does not use handshaking. It is not vital to understand handshaking in detail but the following section describes how it works.

Note that the diagram shows the logic values of the handshaking signals and not the actual voltage levels on the connector pins. The diagram for DAV should be inverted to give the voltage levels present on the three handshaking lines.



a. When a byte is to be sent the source looks at the RFD line. If this is true this means that the acceptor is ready to accept some data. The logic of the line is such that if there is more than one acceptor at any time then the RFD signal will only become true when ALL acceptors are ready.

b. When the source has found that RFD is true it can put a byte of data onto the data lines and then it will make the DAV signal true to indicate that the data on the data lines is a valid byte.

c. The acceptors respond to this by making RFD false. The source must now keep DAV and the data byte on the bus while the acceptors process the byte.

d. When the acceptors have all accepted the byte then they make the signal DAC true. Again this signal will become true when ALL acceptors have accepted the byte.

e. As soon as this happens the source may deassert DAV and then change the data on the data lines.

The acceptors respond by deasserting DAC, completing the handshake sequence.

2.2.10 Bad Bus States.

Note that RFD and DAC are never simultaneously true during the handshake. The controller can use this fact to test for the presence of listeners on the bus. If a device is addressed as a listener then it will make at least one of these signals false. If no device is listening at any time then both of the signals will be true. If RFD and DAC are both true when ATN is also true this means that no IEEE-488 devices are connected to the controller at all. The Q-488 IEEE interface makes frequent checks on the bus state and generates an error if it is found to be invalid.

SECTION 3 - INSTALLATION

3.1 INSTALLING THE Q-488 BOARD

3.1.1 Accessing the QL Expansion Port.

Remove the power supply from the QL: the orange LED at the bottom left corner of the keyboard should be unlit. The Q-488 IEEE hardware is designed to plug into the expansion port under the lower left of the machine. The port is normally blanked off with a piece of plastic. Unclip this to gain access to the edge of the main QL circuit board. Insert the Q-488 board into the QL, chips uppermost. Carefully apply firm pressure until the connecting pins are felt to engage. There should be no need to use excessive force. If in doubt, re-check the alignment of the connectors.

3.1.2 Using a QL Expansion Unit.

If you have an QL Motherboard that allows you to plug in several expansion boards at once, please refer to the manufacturer's instructions for information on inserting new cards. The Q-488 IEEE interface has been designed to work in any slot number. In addition, it is possible to have more than one Q-488 interface attached to the QL at a time! In this instance, the user can distinguish between the interfaces by referring to the appropriate slot number when opening QL IEEE channels.

3.1.3 Powering Up.

The interface takes its 5V power supply from the unit into which it is plugged.

IMPORTANT: Ensure that the board is correctly fitted before switching the power on as otherwise it may be damaged.

Reconnect the power, remembering to remove any microdrive cartridges from their slots and to open disc drive doors. Amongst the "start-up" messages at the top of the screen should appear something like:

CST Q-488 Module PRL V m.nn (c) 1985

where m.nn is the version number. This indicates the QL has been able to read the ROM firmware on the Q-488 board.

3.1.4 In Case of Fault.

If no such message appears, or worse still the QL does not "boot-up" properly to display its initialisation screen, remove the power rapidly. The most likely fault is that the board is not plugged in properly. Remove it from the expansion port and peer into the QL to verify that the pins in the edge connector are undamaged. Try re-installing the board and switching on. If the problem persists, check the QL still functions on its own and with any other expansion boards that are to hand. Beyond this, please return the board to your dealer with a written description of the symptoms.

3.1.5 A Simple Test Example.

With no IEEE devices connected, boot the QL and type:

```
ANALYSE
```

A rectangular window will appear in the top right hand corner of the screen: this is the built-in Q-488 Bus Analyser, described fully in Section 6. Its purpose is to display the status of the IEEE bus lines and it is very useful in tracking down bus faults and problems. Note that the REN (Remote Enable) line is asserted: in the absence of any external devices, the interface has taken on the role of system controller.

Type in the following commands, watching their effect on the bus lines:

```
OPEN #5,ieee_6
PRINT #5,"Anything"
```

Nothing happens directly after the **OPEN #5**, but the **PRINT #5, "Anything"** causes items to appear in the bottom line.

Do not worry if you don't understand the details, but this is what happens:

OPEN #5,ieee_6 opens a QL input/output channel labelled 5 to an IEEE device at address 6. (The numbers are of no particular significance.) This is remembered by the QL until the channel is closed with a **CLOSE #5**. Nothing happens to the bus lines at this stage.

PRINT #5,"Anything" attempts to send the string **"Anything** through the QL channel 5, and hence to an IEEE device at bus address 6. In the previous chapter, the protocol for such interchanges was described: first, the IEEE device must be addressed to listen (MLA ...); second, the message is sent; third, the device is told to unlisten (UNL). In this particular example, there is no device 6 connected and the transaction is held up at the first stage.

Looking at the analyser window, the bottom line shows that the byte **$26** (hexadecimal for "&") is currently on the bus

data lines, along with the control line ATN (Attention). These taken together are interpreted as **LAG 6** (Listener Address Group 6) - the IEEE command to make device 6 listen.

Proceedings stop at this stage because there are no devices physically connected. The **LAG 6** command has to be acknowledged by any devices by a procedure known as handshaking (see Chapter 2) before the next byte can be output.

To restore the bus to its quiescent state, type:

IB_RESET #5

The analyser reflects **$00 NUL** present on the data lines.

In case of difficulty, refer to Section 3.1.4.

3.1.6 Added SuperBASIC Keywords.

The Q-488 adds many SuperBASIC keywords to the QL. If the system has been booted with the QL Toolkit, typing

EXTRAS

will list what is available. Most of the Q-488 keywords begin **IB_.....** In addition, there are words to define the logic states **TRUE** and **FALSE.**

The new words and their actions are fully described in the Reference Section.

3.2 CONNECTING THE INTERFACE TO PERIPHERAL DEVICES

3.2.1 Connecting Cables.

The bus lines are brought out of the Q-488 expansion unit on a 34-way IDC plug. A screened ribbon cable adaptor is supplied to convert this to the 24-way IEEE standard plug. If an IEC-625 type connection is required, use a special adaptor or add this type of plug to the cable provided.

Note that IEEE bus connecting leads are not supplied as these are normally supplied with the equipment intended to be used with the interface. They may be ordered separately from dealers.

3.2.2 Setting Device Addresses.

You must choose a unique primary address in the range 0 to 30 for each and every device that is connected to the IEEE bus. Most pieces of equipment have a dual-in-line (DIL) switch on their back panel to set the address: refer to the appropriate instruction manual on how to alter it. A few devices may have their addresses set under software control.

Confusion will arise if more than one device accidentally share the same primary address, particularly if both become addressed to talk! It is a good idea - once the address of each device has been decided upon - to write the number on a label and stick this on the front panel where it is easily seen.

3.2.3 Connecting Devices.

When you have set up the addresses of all the devices, you can start to connect them together with IEEE cables. The connectors on these cables may be stacked upon each other and bolted together for security.

The manuals for devices give hints on choosing cable routes; basically you must avoid producing loops with the cables, which would be very prone to interference. To recap what was said in Section 2, the recommended methods are:

Linear: Connect all the devices one after another.

Star: Connect all the devices to the controller separately.

The second method suffers from the disadvantage that if many IEEE connectors are stacked together, too much strain may be applied to the connector mountings causing them to bend. You should avoid stacking more than three connectors at any point.

3.3 TESTING THE BUS

You may connect up to 15 devices at once to the IEEE bus. (The limit is imposed for electrical reasons.) To check that the hardware is working, try the following examples with one device at first then more later.

You can use the interface itself to test that all the addresses and connections are correct. If you try to address a device and then send data, testing the IEEE bus lines will tell you whether or not they were accepted.

The simple way of doing this manually is to use the **Q-488 Bus Analyser** which will display the bus transactions on the screen. Call up an analyser with:

```
ANALYSE
```

For this example, we shall assume that we are testing the connection to a device at address 5. First, open a Q-488 channel to it:

```
OPEN #9,ieee_5
```

(The channel number 9 is of no particular significance.) Nothing will happen to the bus lines. All IEEE devices are capable of accepting commands, so we shall test device 5 by trying to address it to listen. Watch the **analyser** window carefully:

```
PRINT #9,"A"
```

All is well if the bus lines return to **$00 NUL:** the command and data byte were accepted. Alternatively, if the device does not know how to listen (this is unusual), the addressing commands will go through and leave the byte **$41 "A"** on the data lines.

Something is wrong if the display halts at the addressing stage: **$25 "A" LAG 5.** Refer to the end of this section.

If anything remains on the bus other than **$00 NUL**, use:

```
IB_RESET #9
```

before retrying. Close the channel when you have finished:

```
CLOSE #9
```

The **analyser** can be removed by activating its input cursor (type **<CTRL>-C**) then pressing **<CTRL>-<ALT>-F5.**

The second approach is to run the following SuperBASIC program that makes use of some of the low-level Q-488 commands. It should tell you which devices are connected. Do not worry about the details: at this stage, it is merely provided as a quick test of the bus connections. It is quite instructive to run an **analyser** at the same time.

```
100 REM +++ IEEE Bus Testing Program +++
110 OPEN #5,ieee_
120 IB_RESET #5
130 FOR dev=0 TO 30
140   IB_TCS #5
150   IF no_listener_present(#5) THEN
160     PRINT "No devices are present."
170     EXIT dev
180   END IF
190   IB_UNL #5
200   IB_LAG #5,dev
210   IB_GTS #5
220   IF no_listener_present(#5) THEN
230     PRINT "Device"!dev!"is not present."
240   ELSE
250     PRINT "Device"!dev!"is present."
260   END IF
270 END FOR dev
280 CLOSE #5
290 :
300 DEFine FuNction no_listener_present(chan)
310   LOCAL ndac
320   ndac=4096
330   PAUSE 50
340   RETURN (IB_LINES(#chan) && ndac)) = 0
350 END DEFine
```

After testing the bus for the presence of any devices at all, the program will output a list of valid primary addresses in the range 0 to 30 with a comment indicating whether the device was found or not. Check that the list agrees with what you expected.

If any problems arose in the test you must check the device in question to make sure its cable is connected properly, also that it is switched on, and that its address is set correctly. If the test still fails for <u>all</u> IEEE devices that you attach, then it is likely that the Q-488 interface is at fault: contact your supplier for advice.

SECTION 4 - USING THE Q-488 INTERFACE

4.1 INTRODUCTION

4.1.1 The Scope of this Section.

This section of the manual is intended as an introductory guide to the use of the Q-488 IEEE interface from SuperBASIC. It contains explanations of how to perform the most likely tasks that you will need, with some sample programs to illustrate.

There are far too many IEEE devices commercially available to give an extensive set of examples. In this section, they have been written mainly for a Hewlett-Packard HP7470A or HP7475A pen plotter. This can perform most of the functions of which the bus is capable. It should be easy to change the examples for other devices.

4.1.2 Setting Up the System.

All the examples assume that the IEEE system has been connected up in the required configuration, and the connections have been tested following the procedure given in Section 3 earlier.

Please note that the you should alter the examples where necessary when device addresses given are different from those set on your own equipment. The HP plotter is normally supplied with its address set to 5, so this is the value assumed in the programs.

4.2 PRINCIPLES OF OPERATION

4.2.1 SuperBASIC Extensions.

The firmware supplied with the Q-488 interface has the task of decoding all the IEEE input/output calls (e.g. **OPEN #n,ieee...; INPUT #n,a$; LBYTES ieee_...** etc.) passed to it through the QL's operating system, QDOS.

In addition, a number of special commands have been added to the list of keywords that SuperBASIC normally knows about. If the QL Toolkit is loaded or the CST disc interface attached, these extensions can be listed with the **EXTRAS** command. If not, you will have to refer to the list given in the Reference Section 7 of this manual.

4.2.2 The IEEE Device.

To be of any practical use, a computer must be capable of processing input to produce output. On the QL, this is done through devices: these can be thought of as the physical connection through which the computer communicates with the outside world. Some typical QDOS input and/or output devices are:

con	-	the console (keyboard)
scr	-	the VDU screen
ser2	-	the second serial port
mdv1	-	the left-hand microdrive
flp1	-	the top floppy disc drive

When a Q-488 interface is added to your QL, QDOS will recognise a new device:

ieee	-	the IEEE bus

4.2.3 Opening and Closing Channels.

Many of the SuperBASIC keywords require you to supply a channel number as an argument. A few familiar examples are:

OPEN #	**CLOSE #**
INPUT #	**PRINT #**
CLS #	**PAN #**

QDOS uses these channel numbers to identify where to send output and where to get input.

Before any SuperBASIC command can send (or receive) messages to (from) a device, an association must be made between a channel and a device. This is the purpose of the command **OPEN #**. The full syntax is:

OPEN #numeric_expression, device

The **numeric_expression** evaluates to an integer which becomes the channel. When SuperBASIC makes subsequent references to this channel, the I/O will automatically be associated with the physical device stated in the **OPEN** statement.

QDOS will remember the channel until a **CLOSE #** is executed. The syntax is:

CLOSE #numeric_expression

In the case of the Q-488 interface, before attempting I/O your first action must be to **OPEN** a channel to the IEEE bus. This is slightly more complicated than for other QDOS devices as the full syntax for the device name is more involved:

IEEE[slot]*[_primary[Ssecondary]][_option]***

(As usual, the [...] signify an optional item, and the *...* mean that the item is repeated as necessary. The alphabetic characters can be in upper or lower case.)

For the purposes of the examples, it is assumed that there is only one Q-488 card attached to the QL, so the **slot** part is irrelevant. The **options** part is not important here, either. A full description is given in the Reference Section 7.

The simplified syntax is:

IEEE*[_primary[Ssecondary]]*

Some examples of valid device names are:

ieee_5	to IEEE device 5
IEEE_2_20	to IEEE devices 2 and 20
ieee_1s5	to IEEE device 1, secondary address 5

4.2.4 Choice of Channel Number.

An important point to note is that the channel numbers 0,1 and 2 have a special significance to SuperBASIC. Under normal circumstances, do not **OPEN** these channels to the **ieee** device. (Try it if you like!)

You are unlikely to run out of channels: QDOS can cope with a large number, the precise amount being determined by the memory available.

If you reopen an existing channel (i.e. one that has been opened, but not closed), QDOS will replace the old definition with the new. There is no simple way of closing channels whose identification has been forgotten.

A suggestion you may find helpful is to make the channel number the same as the the primary address of the IEEE device. For example:

```
          OPEN #5, ieee_5
or        OPEN #36, ieee_3_6
(but not  OPEN #0, ieee_0)
```

4.2.5 Addressing IEEE Devices.

Before any data transfer can occur on the IEEE bus, the devices involved must be addressed. (If you are unfamiliar with the principles behind this, please go back and read Section 2.) From the point of view of a SuperBASIC program, you need to be able to specify to which IEEE device(s) output is to be sent, or from which IEEE device to obtain some input. On its own, the QDOS device type **ieee** is not enough: you must supply the additional information of which IEEE devices to use. This is simply done by listing the required IEEE device addresses after the **ieee** when the channel is **OPEN**ed (syntax as in the previous subsection).

Handling the IEEE addressing protocol is quite complicated. Fortunately, the Q-488 interface firmware does all of this automatically for you. Whenever a SuperBASIC command sends output to a particular **ieee** channel, all the IEEE devices listed in the channel definition made upon **OPEN**ing will be addressed to listen, the data is sent, and then they are told to unlisten. When input is requested, you must remember that only one talker is allowed on the bus at a time. The right-most IEEE device listed in the channel definition (there is no confusion if there is only one device mentioned) will be addressed to talk, send its data, then told to untalk.

Each channel may be used for both input and output: no distinction is made. The use is determined by whatever is appropriate for the calling command.

Remember that valid primary and secondary addresses lie in the range 0 to 30.

4.3 TRANSFERRING DATA OVER THE IEEE BUS

4.3.1 Sending Data to IEEE Devices.

Once an **ieee** channel has been **OPEN**ed, sending data is achieved by using SuperBASIC output commands. The most useful is **PRINT #;** for example:

```
PRINT #5, "Hello"
```
or
```
PRINT #36, a$, PI
```

In general, the syntax is:

```
PRINT #[ #channel, *[expression]* ]*
```

To illustrate, in the case of a HP7470A plotter at address 5, we can make it select pen number 1 by typing:

```
OPEN #5, ieee_5
PRINT #5, "SP1;"
```

and we can draw a small square by typing

```
PRINT #5, "PU0,0,PD100,0,100,100,0,100,0,0,PU;"
```

The Q-488 interface can send both strings and numerics using **PRINT #**. Strings are sent character by character, starting from the left; numerical values (e.g. PI) are first converted by QDOS to a string ("3.14159...") from the QL's internal representation before being transmitted.

As a further example, here is a complete program to plot data points on the HP7470A, marking each with a small square:

```
100 OPEN #5,ieee_5
110 REPeat plot_square
120   INPUT "Input point coordinates (x,y): ",x,y
130   PRINT #5, "PA,PU";x;",";y;";"
140   PRINT #5, "PR-60,-60,PD 120,0, 0,120, -120,0, 0,-120,PU;"
150 END REPeat plot_square
```

Line 100 opens channel 5 to IEEE device 5.
Line 120 asks you to input the x,y coordinates of a data point which is read into the variables x and y.
Line 130 turns the x,y coordinate that you type into a string like **"PA,PU1024,1456;"** and this is then sent to the plotter, moving the pen to the correct position.
Line 140 draws a small square, 120 units (3mm) on a side, at the pen position.
Line 150 repeats the process until <CTRL>-<SPACE> is typed. Remember that this will leave channel 5 open.

Whenever characters are sent from the QL, they are placed in an internal output queue which can hold about 500 bytes. The transmission on the IEEE bus occurs as and when the acceptors are ready. The buffer is managed asynchronously by QDOS, so the calling program can continue uninterrupted (unless the queue has become full).

Most other SuperBASIC output commands are not relevant to the **ieee** device. Examples are:

```
PAN #                    INK #
POINT #                  PAPER #
```

CLS # has been implemented to send the ASCII code 12 ("Form Feed"). The SuperBASIC extension **BPUT #** should work, if loaded.

4.3.2 Receiving Data from IEEE Devices.

Taking input from the bus is as easy as sending output. An **OPEN** channel is used in conjunction with a SuperBASIC input command. Usually, this will be the **INPUT #** keyword. For example:

```
INPUT #5, bus_value
or  INPUT #36, string_1$, string_2$, value
```

In general, the syntax is:

```
INPUT #channel, variable *[, variable]*
```

Input is handled slightly differently: the calling program halts execution until the input termination condition (refer to Section 7 for details of setting terminators) is met. As a consequence, the program will hang if the addressed IEEE device is not ready (or unable) to talk. Typing **<CTRL>-<SPACE>** at this point will escape.

The **variable** may be either a string or a numeric. A string will retain the ASCII representation of the data received (e.g. "Hello from device 5."), so should be used whenever text is expected. If you use a numeric variable, then SuperBASIC will attempt to convert the ASCII data (e.g. "3.4E12") to a numerical value: this can be convenient if you are expecting purely numerical data.

If you give a list of variables, then data will be read into the first until a terminator is found, then into the second, and so on until the list is exhausted.

IEEE devices usually need to be commanded beforehand to output something. For example, the HP7470A must be told what string to return: this is *done* by using **PRINT #** to send one of the HP7470A Output Commands:

```
PRINT #5, "OI;"
INPUT #5, identification$
```

The **OI** (Output Identification) command tells the HP7470A to send a string containing its model number. The **INPUT #** statement then reads this string into a string variable.

```
PRINT identification$
```

will cause the computer to print

```
7470A
```

on the screen.

You may find it useful to set a timeout on input: this means that you can write your program so that it can deal with absent talkers. The SuperBASIC keyword **INKEY$** is a function which will return a single character input from a specified channel with an optional waiting period. The syntax is:

INKEY$ #channel [, time]

The **time** is a number from 1 to 32768 measured in units of 1/50th of a second. If **time=0**, then **INKEY$** will return immediately; if **time=-1**, it will wait indefinitely.

SuperBASIC extensions such as **BGET #** should also work.

4.3.3 SuperBASIC Extensions Requiring Channels.

Several of the keywords added by the Q-488 interface must be supplied with an **ieee** channel. Perhaps the most useful example is the command that flushes the output queue and resets the Q-488 hardware:

IB_RESET #channel

where **channel** can be any one that has been **OPEN**ed to an **ieee** device. In this instance, the **channel** is a dummy argument which is used to pass the command to the Q-488 firmware.

Refer to Section 7 for a comprehensive list.

4.4 OTHER COMMANDS

Several SuperBASIC commands act on QDOS devices. Some examples are:

COPY	**MERGE**
LBYTES	**SBYTES**
LOAD	**SAVE**

These may be very useful when used with the **ieee** device. **copy** is especially powerful. To illustrate:

COPY con TO ieee_5

will allow you to send input typed on the QL's keyboard directly to IEEE device 5;

COPY ieee_7 TO scr

will list the output from device 7 to the QL's screen.

SECTION 5 ADVANCED USE

5.1 ADVANCED USE FROM SUPERBASIC.

This section deals with the more advanced things you can do with the Q-488 interface. Most of these will not be needed in everyday use and this section may be omitted at a first reading. Though most of the examples are specific to Superbasic programs the principles discussed apply equally to programs using the Q-488 interface from other languages. A small number of examples are given using other QL language systems.

5.1.1 Options in Q-488 Filenames.

A number of options may be set for each open channel in the Q-488 system. The principal options control the use of character translation on input and output. This allows you to match the characteristics of the QL language system you are using to the IEEE devices that you are controlling with the Q-488 interface.

The options may be set individually for each channel that is opened, or a default value may be set which will be used where no explicit value is given. The option settings on a particular channel may also be changed at any stage once the channel has been opened.

The following methods are used :-

Setting Options:

The 'options' field of the IEEE device name given to the OPEN statement can be used to give initial values for the options associated with that channel. These options settings will not (usually.. see section 7 for the exceptions to this) affect subsequently opened channels. The range of options which can be chosen in this way has been (deliberately) limited to the most widely used alternatives.

Example:

```
OPEN £4,ieee_4_e0
```

opens a channel to device 4 with EOI ignored on input and output.

Changing Options already set:

A special QDOS trap call to the IEEE system is implemented which allows options on an existing channel to be changed. This trap may be easily accessed from Superbasic by means of a number of Q-488 extension keywords which are provided in the Q-488 system. This also allows you to set up combinations of options which are not covered by the previous method.

Example:

```
IB_RAW £4,TRUE
```

switches off all character translation on the previously opened channel.

Specifying default options:

A special type of device name is provided which accesses the Q-488 interface itself, rather than individual channels. Any options set up in this way will be treated as defaults for the whole system and will apply to any subsequently opened channels.

Example:

```
OPEN  £99,ieee_
IB_RAW £99,TRUE
CLOSE  £99
```

after these commands have been executed then all subsequent channels opened will have character translation switched off by default.

5.1.2 Options available with Q488 channels.

The full list of options associated with a Q-488 channel will be detailed in section 7. In this section we will explain in more detail the more generally useful options.

The options control three main areas of operation, the use of the EOI line, output terminator characters and input terminator characters. The qualifiers which may be added to a Q488 device name select from the most commonly used settings for the option whilst less commonly used combinations of settings must be selected by modifying the options set on an already open channel as detailed in section 7.

The following qualifiers are provided (as well as some others which are detailed in section 7):

R This qualifier indicates that the channel should be a 'raw' channel. This type of channel performs no character translation either on input or output and uses EOI only to terminate input. You should use raw channels if you want to transfer binary data.

Example:

LBYTES ieee_4_r,buffer

loads binary data from device 4 into the QL memory. (Assuming that 'buffer' has been set up to point to an appropriately sized area of free memory).

E This qualifier controls the use of the EOI line in data transfers. It should be followed by a single digit (which must be either 0 or 1). A value of zero indicates that the channel should totally ignore the EOI line, both on input and on output. Note that if you try to use such a channel for input you must be sure that you have set up an input terminator character otherwise the string read operation will only terminate when the input buffer fills up. This can be used to perform a transfer of a fixed number of bytes of data into the QL.

Example:

SBYTES ieee_5_e0,buffer,80

sends 80 characters from the QL memory to device 5, without setting EOI on the last character.

T This option controls the use of terminator characters, both on input and output. It should be followed by a single digit which must take one of the values 0,1 or 2. The options have the following interpretation:

0 No additional characters are added to output strings, no characters are taken as input terminators. If you are using this option on an input channel make sure that EOI is set on the last byte by the device sending data otherwise the transfer will not be completed. (If EOI is being ignored on the channel then the transfer will only be completed when the input buffer fills up).

This option is the one best suited for use between two QLs connected to the same IEEE interface.

1 No additional characters are added to output strings, but the line feed character is used as an additional input terminator. In other words, an input string will be terminated if either EOI is received with a character, or the Linefeed character is received.

2 Whenever the line feed character is to be output, it will be converted into a carriage return followed by a linefeed. On input line feed is still taken as a terminator, but any control characters sent to the interface will be removed from the string returned on the QL.

This option is the most suitable for general use with ordinary IEEE devices. Strings sent by such devices are normally terminated by a carriage return/ line feed pair, with EOI set on the linefeed. This option will accept both the CR and the LF, but will remove both from the string returned to the calling program.

Examples:

```
OPEN £4,ieee_4_el_t2
PRINT £4,"R3F3T"
INPUT £4,result
```

opens a channel to a 'standard' IEEE device, sends the string **R3F3T** to it and then reads back a numeric result.

```
OPEN £4,ieee_12_r_e0
PRINT £4,"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
```

opens a 'raw' channel to device 12 and then sends the letters of the alphabet to it. EOI will not be sent with the last character.

5.1.3 More about OPEN : Multiple and secondary addresses.

The channels discussed so far have been opened to single devices with primary addressing only. The IEEE-488 standard allows for two further levels of sophistication when addressing devices.

Multiple Listener Addresses:

If you have are addressed a device as a listener, in other words, if you have openied an output channel from the QL, then it is possible to address additional devices as listeners at the same time. When you send the data itself then it will be received by all the addressed devices. The handshaking lines on the IEEE bus are designed to allow this mode of transfer. It is not possible to have more than one device simultaneously addressed as a talker however.

An analogy to conversations between people should make this point easy to remember, it is easy for several people to listen to a single speaker, but much harder for a single listener to understand what is being said by several people talking at the same time.

To specify multiple device addresses in a Q-488 device name you must put all the device addresses you want into the device name, separated from each other by underscore characters, immediately before any option qualifiers in the name. (Note that the options apply equally to all the devices in the transfer- you cannot arrange to send the same strings, but with different EOI options, to several devices at the same time).

Examples:

```
OPEN £4,ieee_1_2_3_4_5_6_7
OPEN £5,ieee_10_12_t2_e0
```

Channel £4 could be used to send data to devices 1,2,3,4,5,6 and 7 simultaneously and Channel £5 could be used to send data to devices 10 and 12, with conversion from the QL linefeed termination to Carriage Return/Linefeed termination on the IEEE bus, and ignoring the EOI line.

If you use such a channel to address an external talker, i.e. as an input channel, then only the last device in the last will actually be addressed to talk. This is because the IEEE standard specifies that any devices which have been addressed to talk should automatically unaddress themselves if they notice that another talker has been addressed. The Q-488 system does not disallow such use at present, though it is obviously bad practice to rely on this.

Secondary Addressing:

The IEEE standard provides thirty one possible device addresses, these are known as 'primary' addresses. In addition each device may implement what are known as 'secondary' addresses. Secondary addresses may have a value in the range zero to thirty. They are normally used where a device may have a number of distinct functions, and the secondary address can be used to select which function of the device is being addressed.

A good example of this would be a (hypothetical) disc drive which used secondary address zero as a control channel to specify filenames on the disc to open, and the other secondary addresses would be used to send data to/from files opened on the disc.

Secondary addresses may be included in a Q-488 filename after any primary address, separated by the character 'S' (or 's'). They can be combined with other primary and secondary addresses in multiple listener lists.

Examples:

OPEN £4,ieee_1s4

opens a channel to device 1 secondary address 4.

OPEN £5,ieee_1_2s3_3_4s30_e0

opens a multiple listener channel to device 1, device 2 secondary address 3, device 3 and device 4 secondary address 30.

5.1.4 Service Requests and Polling.

The IEEE standard provides two methods of polling devices in the system to determine their status. Serial polling provides a method of reading a byte of status information from a single device. If you want to poll more than one device then you must serial poll each device sequentially. As an alternative to serial polling, parallel polling allows up to 8 devices each to send a single bit of status information to the bus controller simultaneously.

Serial polling is widely used by IEEE interfaced measurement equipment in conjunction with the Service Request (SRQ) function. A device can signal that it has completed making a measurement by activating the SRQ signal. The controller may then be interrupted and can read the status byte of the device by means of a serial poll operation.

Parallel polling is less widely implemented but a few devices allow it to be done. It is much faster than a serial poll because no new devices need to be addressed: all devices on the bus respond simultaneously. It is usually necessary to configure the devices beforehand so that they each use a different bit of the IEEE data bus on which to reply.

The Q-488 system supports both serial and parallel polls by means of additional Superbasic commands. Programmers not using Basic may access the polling routines by means of the additional QDOS Trap calls, detailed in chapter 7. (Support for Parallel Polling is not implemented in early versions of the Q-488 system).

Three functions are provided to:

i) test the state of the SRQ line
ii) perform a serial poll
iii) perform a parallel poll

Testing the state of SRQ:

The additional function **IB_SRQ(£channel)** is used to test the state of the SRQ signal. It returns TRUE if any devices are currently requesting service, FALSE if there are *none*. The *channel* number should be an IEEE channel, though the actual device addresses on the channel are not important. It may be convenient to use a 'null' channel, to avoid confusion with addressed channels.

Example:

```
10   OPEN £4,'IEEE_'
20   OPEN £5,'IEEE_12'
30   PRINT £5,'Trigger'
40   REPEAT polling_loop
50      IF IB_SRQ(£4) THEN EXIT polling_loop
60      PRINT 'Waiting for measurement'
70   END REPEAT polling_loop
80   INPUT £5,result
90   PRINT 'The result was ';result
```

This program assumes that a device 12 is available on your system which can take a measurement when the string 'Trigger' is sent to it. It signals that it has completed its measurement with SRQ. We also assume that there are no other devices on the bus which could use the SRQ line. This is important since the program does not check to make sure that the SRQ signal is really coming from device 12 before attempting to read back the result of the measurement. We will see how to do this in the next example.

Performing a Serial Poll Operation:

The additional Superbasic function **IB_SPOLL(£channel)** returns the Serial Poll status byte from the device addressed by the channel. Note that the channel should only have a single address on it.

N.B. There is a known bug in Version 1.0 of the Q488 system. As a result of this it is necessary to explicitly unaddress the serial poll channel after using the **IB_SPOLL** function. You can do this with the **IB_UNADDR** command.

e.g.

```
h = IB_SPOLL(£14)
IB_UNADDR £14
```

The result is returned as an integer in the range 0 to 255. If the device which is being polled was also requesting service via the SRQ function then bit 6 (mask value 64) will be set, otherwise cleared.

Example:

This is a somewhat extended example to illustrate how to determine the source of a service request in a system where more than one device may issue one by means of serial polling.

We assume that there are two devices (device 1 and device 2) which are each capable of taking measurements independently of each other. They may both be activated by sending the string 'GO'.

When either device has made a reading it will signal that it has done so with the SRQ line. The controller can determine which of the devices was responsible by reading their serial poll response.

There is an IEEE interfaced printer on the system with address 7 which will be used to store a log of the times at which readings were made by each measuring device and a floppy disc drive which will be used to obtain a machine readable record of the experiment.

```
10   OPEN £4,'IEEE_'
20   OPEN £5,'IEEE_1'
30   OPEN £6,'IEEE_2'
40   OPEN £7,'IEEE_1_2'
50   OPEN £8,'IEEE_7'
60   OPEN £9,'FLP1_LOG_FILE_DAT'
```

opens channels to various combinations of the devices and to the printer and disc drives.

```
100  PRINT £8,"Run started : ";DATE$
110  PRINT £8
120  PRINT £9,"Run started : ";DATE$
130  PRINT £9
140  PRINT £7,"GO"
```

prints a header message on the printer and activates the devices.

```
200  REPEAT polling
210     IF IB_SRQ(£4) THEN poll_devices
220  END REPEAT polling
```

the polling loop

```
1000 DEFine PROCedure poll_devices
1010    IF it_is(£5) THEN
1020       read_device £5
1030    ELSE
1040       IF it_is(£6) THEN
1050         read_device £6
1060       ELSE
1070         PRINT £8,"Unknown SRQ source":STOP
1080       END IF
1090    END IF
1100 END DEFine
```

a procedure to test each device in turn to identify if it is the source of the service request.

```
2000 DEFine FuNction it_is(£dev)
2010    LOCAL response%
2020    response% = IB_SPOLL(£dev)
2030    IB_UNADDR £dev
2040    RETURN (64 && response%) = 64
2050 END DEFine
```

a function to test if a given device is requesting service

```
3000 DEFine PROCedure read_device(£dev)
3010    LOCAL time$,reading$
3020    INPUT £dev,reading$
3030    time$=DATE$
3040    time$=time$(13 TO)
3050    PRINT £8,"Channel ";dev;" at ";time$
3060    PRINT £8,"Reading ";reading$
3070    PRINT £8
3080    PRINT £9,"Channel ";dev;" at ";time$
3090    PRINT £9,"Reading ";reading$
3100    PRINT £9
3110 END DEFine
```

read back the reading from the device, and print out a log of the event on the printer and the floppy disc file.

Performing a parallel poll operation:

The additional Superbasic function **IB_PPOLL(£channel)** may be used to perform a parallel poll operation. Since no addressing is needed with a parallel poll it is unimportant what sort of a channel is used with this call. It returns an integer result in the range 0 to 255. The polling is carried out immediately, without waiting for the Q-488 output queue to empty.

Example:

```
1000 DEFine FuNction read_parallel_poll
1010    LOCAL result
1020    OPEN £99,'IEEE_'
1030    result = IB_PPOLL(£99)
1040    CLOSE £99
1050    RETURN result
1060 END DEFine
```

this defines a convenient function which returns the current parallel poll response on the bus. It opens a channel, (number 99 is used to avoid clashes with already opened channels), reads back the poll response, and closes the channel again. This means that a channel will be opened and closed each time the function is called. For efficiency it would be better to have a channel permanently open throughout your program's execution, and to poll this channel.

Note that the IB_PPOLL function is *not* implemented on early versions of the Q-488 system.

5.2 Use from languages other than SuperBasic.

One of the strengths of the QL operating system is the ease with which new I/O devices may be added to the system. This feature means that it is very easy to use the Q-488 system from any of the other programming language systems available for the QL. This section will take as examples the following language systems:

GST QC C language system
Metacomco LISP
Metacomco Pascal
68000 Assembly Language

We shall give examples of the use of the Q-488 system from each of these languages.

The majority of interface operations will be accomplished merely by opening 'channels' in exactly the same way as you would with the disc or microdrive systems or with the serial interface. The options available in Q-488 device names should allow most IEEE bus operations to be performed.

Such operations as serial and parallel polling, or driving the interface directly at a low level, cannot be performed in the manner described above. These operations are accessed from SuperBasic by means of additional keywords provided by the Q-488 system. In other language systems the corresponding functions will not in general be directly available.

Access to these functions is provided by 4 special QDOS trap calls which have been reserved for use by the Q-488 system. They are all calls to TRAP £3, the QDOS general I/O trap. These calls will be documented fully in section 7 of this manual, we will describe only the principles here.

5.2.1 GST QC C Language System

As an example of the use of the Q-488 interface from C here is a program which draws a square on an IEEE interfaced Hewlett Packard plotter.

```
£include <stdio.h>

main ()
{
     int fd;

     fd = fopen ( "IEEE_5", "w" );

     fputs ( "IN; SP1; PU 100,100; \n", fd);
     fputs("PD1000,100;  \n",           fd);
     fputs("PD1000,1000; \n",           fd);
     fputs("PD100,1000; \n",            fd);
     fputs("PD100,100; \n",             fd);
     fputs("PU0,0; SP; \n",             fd);

     fclose ( fd );
}
```

The operation of this program should be straightforward: a channel is opened to IEEE device 5 (assumed to be a Hewlett Packard plotter). The plotter is initialised, pen 1 is selected and a square is drawn on the paper. The pen is then lifted and returned to its stall.

5.2.2 Metacomco LISP

The example below performs exactly the same function as the C program given above.

```
(SETQ plotter (OPEN 'IEEE_5 ()))

(WRITE plotter 'IN;SP1;PU100,100;)
(WRITE plotter 'PD1000,100;)
(WRITE plotter 'PD1000,1000;)
(WRITE plotter 'PD100,1000;)
(WRITE plotter 'PD100,100;)
(WRITE plotter 'PU0,0;SP;)

(CLOSE plotter)
```

the operation is exactly the same as the previous program.

5.2.3 Metacomco Pascal

Yet again, the plotter draws a square!

```
PROGRAM squarebash (input, output);

     VAR plotter : TEXT;

BEGIN
     rewrite (plotter, 'IEEE_5');

     writeln (plotter, 'IN; SP1; PU100,100;');
     writeln (plotter, 'PD1000,100;');
     writeln (plotter, 'PD1000,1000;');
     writeln (plotter, 'PD100,1000;');
     writeln (plotter, 'PD100,100;');
     writeln (plotter, 'PU0,0; SP;');
END.
```

5.2.4 68000 Assembly Language

This time the plotter draws a right-angled triangle.

```
        ; Trap £1 calls...

        mt.frjob  EQU  5

        ; Trap £2 calls...

        io.open   EQU  1
        io.close  EQU  2

        ; Trap £3 calls...

        io.sstrg  EQU  7

        triangle:
             BSR       open        ; Open channel to plotter
                                   ; channel ID into A0.

             TST.L     D0          ; Test failure code
             BNE       failed

             LEA  tristring,A1     ; String to send
             MOVE.W    (A1)+,D2    ; get string length from
                                   ; first word

             EXT.L     D2          ; Bug Fix (see text)

             MOVEQ     £-1,D3      ; No timeout
             MOVEQ     £io.sstrg,D0

             TRAP      £3          ; Call IO trap
             TST.L     D0          ; Test failure code
             BNE       failed

             BSR       close

        failed:                    ; If the I/O call failed
                                   ; return the error code.
             MOVE.L    D0,D3           ;return error code
             MOVEQ     £mt.frjob,D0    ; Force remove job
             MOVEQ     £-1,D1          ; Myself!
             TRAP      £1
```

```
open:
     MOVEQ     £-1,D1     ; Owner Job ID (=me!)
     MOVEQ     £0,D3      ; Access code (R/W)
     LEA       fname,A0
     MOVEQ     £io.open,D0

     TRAP      £2

     RTS

fname:
     DC.W      fnameend-fname-2    ; Length of filename
     DC.B      'IEEE_5'
fnameend:
     DS.W      0          ; ensure word aligned

close:
     MOVEQ     £io.close,D0   ; Close file (ID in A0)
     TRAP      £2
     RTS

tristring:
     DC.W      strgend-tristring-2 ; length
     DC.B      'IN;SP1;PU100,100;',10
     DC.B      'PD1000,100;',10
     DC.B      'PD100,1000;',10
     DC.B      'PD100,100;',10
     DC.B      'PU0,0;SP;',10
strgend:

     END
```

N.B. Note the use of the EXT.L instruction to extend the word length for the IO.SSTRG call. This is to circumvent a bug in Version 1.0 of the Q-488 system, where the string length to send is taken to be a long word, rather than a word, quantity. (See also the separate section on known bugs and fixes).

5.3 Use of the Q-488 interface in "device" mode.

So far, all the examples of use of the Q-488 interface have used the QL as a controller of devices on the IEEE bus. This is likely to be the most commonly met situation in real life. It is also possible, however, to use the Q-488 interface as an addressable device, just like any other IEEE device. This can be useful in a number of circumstances, for example:

i. if you have an experimental setup with more than one controlling computer it can be convenient to keep each computer permanently connected to the same IEEE bus. By using one or other of the computers as a controller at any one time you can eliminate the need to swap the IEEE cables around. The other computers can be put into "device" mode so that they do not interfere with the current controller.

ii. you may wish (as an OEM) to use the QL as a dedicated intelligent controller for a large, complicated piece of equipment (such as an electron microscope) but not wish the user of the equipment to be able to access the QL directly. You could provide an IEEE interface using a Q-488 and a QL which behaves as an IEEE device which can be controlled by an external computer but which can perform complex control functions entirely within the equipment.

iii. the IEEE interface provides a quick and trouble-free method of communicating between two or more computers. The Q-488 interface is a convenient (and well standardised) method of down loading data and programs between two computers made by different manufacturers.

This section of the manual will give examples showing how to implement each of the above three possibilities. For further details of device mode operation see section 7.

5.3.1 Using more than one controller on the same IEEE bus

Most IEEE interfaces for computers have switches which are used to set options for the interface, such as whether the interface should be system controller, or what device address it should take. The Q-488 interface differs from this norm by having an automatic startup facility which determines when the interface is first initialised whether it should be the system controller. The same technique is used by the Procyon IEEE interface for the BBC Microcomputer.

The automatic startup causes the Q-488 interface to assume that it should be system controller unless there is already a controller on the bus. Therefore, if you have a number of QLs (and/or BBC machines with Procyon interfaces) connected together by an IEEE-488 interface then the machine which is first switched on will become the system controller, and all the others will become devices.

The system determines if it should be system controller by looking at the state of the REN bus line. If this is active then there must be another system controller present (only the system controller may drive REN).

To choose which computer should be system controller merely switch that one on first! If you later change your mind and want to make another computer into the system controller you can do this easily by deasserting REN on the system controller and then resetting the machine that you want to become system controller. This machine will see that REN is not asserted and will become system controller and assert REN. You can then reset the original system controller, which will now become a device.

Example:

Machine 1, originally system controller

```
OPEN £4,'IEEE_'
IB_REN £4,FALSE          deassert REN
```

Machine 2, originally a device

```
OPEN £5,'IEEE_'
IB_RESET £5              reset interface
```

Machine 1

```
IB_RESET £4
```

at the end of this procedure, machine 2 will be the system controller, and machine 1 will be a device.

5.3.2 Using a QL with Q-488 interface as an intelligent controller.

A QL plus Q-488 interface could provide an excellent low cost single board computer for use in measurement/control equipment. The multi-tasking operating system is ideally suited to a real-time control environment. There are a large number of high and low level language systems available whose object code, being re-entrant, position independent and otherwise "nice" is well suited to being programmed into EPROMs which can be permanently installed on the circuit board.

Such a piece of measurement equipment would almost certainly be required to provide a external computer interface, with the IEEE-488 interface being the obvious choice. Using the Q-488 interface in device mode would provide the obvious way of implementing the interface.

We will give a brief example of the firmware needed to drive such a system, using SuperBasic as the implementation language. In a real system, one would probably choose a compiled language, possible C or Pascal, so that the object code produced could be programmed directly into ROM. The example is only intended as a rough sketch of the required code and should not be taken literally.

The automatic startup feature of the Q-488 interface is probably undesirable in this application and so we would take care to avoid becoming system controller if no system controller was on the bus. The easiest way to do this is by deasserting REN on power up and by 'unilaterally' releasing control of the bus. We therefore have a startup sequence like:

```
10   REM Startup sequence
20   OPEN £ieee,'IEEE_'
30   IB_REN £ieee,FALSE
40   IB_RLC £ieee
50   IB_DEV £ieee,5
```

after this code had been executed we would be in device mode, regardless of what was on the bus when we were first switched on. We have assumed a fixed device address of 5, but we could obviously read an address switch or something at this point to determine the desired address to take.

We can then open a device channel to read commands from the IEEE bus which could be translated by some sort of lookup table into an appropriate action. We will assume for the sake of simplicity that only single letter commands are to be decoded:

```
100  REM Command decoding
110  OPEN £cmd,'IEEE_D'
120  IB_SPR £ieee,0

130  REPeat decode
140     c = CODE(INKEY$(£cmd,0))
150     SELect ON c
160        =CODE("X"):  handle_x_command
170        =CODE("V"):  handle_v_command
180        =CODE("?"):  handle_query
190        =REMAINDER:  idle_loop_routine
200     END SELect
210  END REPeat decode
```

Note that the serial poll response is set to zero in line 120. At subsequent points in the program we can set it to different values to indicate the instrument status. The polling will be carried out automatically.

Note also the use of **INKEY$** to read a single character from the IEEE interface, or to return if no character was available. This allows the program to perform additional functions in an 'idle loop' while waiting for the input to appear.

5.3.3 Using the IEEE interface to transfer data between computers.

Because the IEEE-488 interface is well standardised between manufacturers the interface can provide a very straightforward and reliable means of transferring data between two computers.

As an example of this we will consider the problem of transferring data from a QL with Q-488 interface to a BBC machine with a Procyon IEEE interface. We will assume that the data to be transferred is contained in a binary file on the BBC machine, with filename 'BINARY', and that we wish to transfer it to a QL floppy disc. We will further assume, to keep the example simple, that the file is sufficiently small to be read into the BBC machine in entirety. We will assume that its length is &1000 bytes (4096 bytes).

Switch the BBC machine on first so that it becomes the system controller, then the QL. Set the QL device address to 1 by the commands

```
OPEN £4,'IEEE_'
IB_DEV £4,1
```

then load the binary file into the BBC machine RAM

```
MODE 7
*DISC
*LOAD BINARY 2000
```

switch to the IEEE filing system on the BBC machine and send the data over the IEEE bus

```
*IEEE
*SAVE 1 2000 +1000
```

and receive the data on the QL and copy it onto a floppy disk

```
COPY ieee_d TO flpl_binary
```

the data will then be stored in the file on the QL floppy disc.

Text files present a few extra problems, one has to make sure that appropriate line terminators are provided and assumed by the different machines. The Q-488 terminator options can be extremely useful in getting this right. Note that the QL uses a single ASCII character 10 as a new line character, this is more usually interpreted as the Linefeed character and the Carriage Return/ Linefeed combination taken as an end of line terminator.

SECTION 6 - THE BUS ANALYSER

6.1 PURPOSE OF THE Q-488 ANALYSER

6.1.1 Introduction.

A typical bus transaction starts with the QDOS recognising the need to access the Q-488 IEEE hardware. The firmware supplied on the board interprets the commands and drives the specialised IEEE hardware. This results in the bus state being altered or read, as required.

The physical description of the IEEE bus has already been detailed in Section 2. Each of the bus lines can take the logical values of true or false, and these are represented by the presence of different voltages on the wires.

Sometimes it is useful to be able to monitor what is happening on the bus. For example, bus transactions may not be happening as expected or devices may mysteriously "hang". In principle, it is possible to monitor the bus lines directly with a voltmeter, oscilloscope or even a logic analyser. It is possible to buy (expensive) IEEE-488 bus analysers that take matters further by allowing the user to step through all the bus transactions slowly, decoding the commands as they appear.

6.1.2 Outline of Operation.

The Q-488 has such a device built into it. The bus control and data lines are monitored at a low level and are displayed on the QL's screen. This is independent of the internal output queue and so represents what is being seen by all the other IEEE devices on the bus at that moment. It operates in one of three ways:

(1) Free - The bus is read as frequently as is reasonable without severely degrading the performance of the QL.

(2) Step - Each bus transaction is halted until a key is pressed.

(3) Slow - Similar to "Step", but with an automatic timeout onto the next transaction after a short period.

The analyser has already been mentioned in Section 3 as a means of checking the operation of the Q-488 interface after installation. It is also a very good way of familiarising yourself with the operation of the IEEE bus as you can monitor the effects of commands as you type.

6.1.2 A Quick Demonstration.

Try typing:

```
ANALYSE
```

and you will see a new window appear at the top right of the screen. The precise layout of the box will depend on the screen mode, but in the bottom right-hand corner there will be a subsidiary box with the word "Free" in it. Type **<CTRL>-C** once, and a flashing red cursor should appear there. This indicates that until another **<CTRL>-C** is typed, any keyboard input will be sent to the analyser rather than SuperBASIC.

If your system has another device active on it, you could try sending it the string **"Hello"** whilst watching the analyser window. Assuming device address 5, type:

```
OPEN #5,ieee_5
PRINT #5, "Hello"
```

The string will be sent in the normal way, and in addition the analyser will show some of the bytes being transferred. Since the analyser display is asynchronous, the exact amount that it will "see" will depend on the speed with which the bytes are accepted: a very slow device might allow the analyser to show all the bytes, whilst a very fast one might accept the whole message before the analyser has time to update the screen.

If the transfer halts before completion, the byte that is not accepted will be shown in the bottom line of the display. Tidy up by resetting the bus to its quiescent state with:

```
IB_RESET #5
```

To see every stage of the transaction, you should switch the analyser from "Free" to "Step" or "Slow" operation (see later). Finally, tidy up with a

```
CLOSE #5
```

6.2 THE Q-488 BUS ANALYSER

6.2.1 Activating the Analyser.

An analyser is activated by the SuperBASIC command, **ANALYSE**. The full syntax is:

```
ANALYSE [#channel,]  [x0 [,y0 [,mode]]]
```

where:

channel - is an SuperBASIC channel number. This may be used to attach an analyser to a particular Q-488 interface if more than one are plugged into the

QL at a time. If there is only one interface, we recommend that you do not use the channel option since the analyser will be unable to read the bus lines properly.

x0,y0 - can be used to specify the initial coordinates of the top left-hand corner of the analyser display window on the screen.

mode - indicates the initial mode of operation of the analyser. Allowable values are:

0 - Free-running (the default)
1 - Single-step
-1 - Slow-motion

If you wish, you may have several analysers activated at the same time. Try the following, for example:

```
100 FOR x=0 TO 256 STEP 256
110 FOR y=0 TO 200 STEP 35
120 ANALYSE x,y
130 END FOR y
140 END FOR x
```

Each time an analyser is created, it spawns two jobs. This places a limit on the maximum number you can activate. If you have the SuperBASIC **EXTRAS** loaded, you can list them with the **JOBS** command, or alter their priority with **SPJOB.**

6.2.2 The Display Window.

The display window of the analyser is divided horizontally into three boxes, the lowest of which is subdivided into four compartments:

```
¦ Q-488 Bus Analyser  (c) PRL & CST ¦
¦ ATN DAV NDAC NRFD EOI SRQ IFC REN ¦
¦  $00   ¦   NUL   ¦          ¦Free¦
```

Mode 0 Display

(The low-resolution display (mode 8) has a similar layout.)

The top line contains the copyright message; the middle, a display of the current values of the IEEE control and data lines. The latter is maintained in real time as far as possible.

Note that the bus line states seen by the Q-488 interface will not always match those actually on the bus because of the open-collector drivers used. For example, if two devices are engaged in a transfer in which the Q-488 is not taking part, then the analyser will not see the state of NDAC and NRFD.

The bottom line decodes the bus lines into IEEE command mnemonics and the hexadecimal/ASCII representation of bytes on the data lines. The right-most box indicates the mode of operation selected and is the location of the analyser's input cursor.

6.2.3 Activating the Input Cursor.

QDOS indicates the channel from which it is currently taking keyboard input by placing a red flashing cursor there. Normally, this is the SuperBASIC command channel.

Analysers sometimes need keyboard input, too. Each analyser window has an input cursor in the bottom right-hand corner of the display. Before the analyser can respond to any of the commands listed in the next sections, this cursor must be activated.

When **ANALYSE** is executed, the display is put up on the screen and the input reverts to the calling SuperBASIC channel. To activate the cursor in the analyser window, type **<CTRL>-C**. The cursor will start to flash, indicating that the analyser is capable of accepting keyboard instructions. When you wish to return to SuperBASIC, simply type another **<CTRL>-C**.

If there are several analysers active, then each **<CTRL>-C** will cycle on to the next invocation, until eventually you are returned to SuperBASIC.

6.2.4 Moving the Display Window.

Once the analyser has been activated, you can move it round the screen using the four cursor keys. The input cursor must be activated first.

If you are listing SuperBASIC programs, it is a good idea to move the display out of the scrolling region to the bottom right of the screen.

6.2.5 The Function Keys.

The function keys **F1** to **F4** are set up to do the following:

F1 - Select single-step operation. Once this has been done, further presses of **F1** step onto the next transaction.

F2 Select free-running operation. This is the default option. The display monitors the bus lines asynchronously.

F3 - Select slow-motion operation. This behaves like "Step" except that the next transaction goes ahead automatically after waiting about 1 second.

Pressing **F3** during the delay cuts it short and moves onto the next step.

F4 - Redraw the display window. A useful facility if the analyser window has been over-written by some other screen output.

<ALT>-[any key] - Deactivate the analyser. Removing the analyser has been made a multiple key operation to prevent it happening accidentally.

Remember that these actions only affect the analyser if its input cursor in the bottom right of the display window is active (i.e. flashing).

SECTION 7 - REFERENCE

7.1 INTRODUCTION

This section describes in detail the facilities available with the Q-488 Interface. These fall naturally into three areas:

1: The standard system.

2: The low-level interface.

3: The bus analyser.

7.2 THE STANDARD SYSTEM.

The Q-488 interface is implemented as a standard QDOS device driver. This means that all SuperBASIC input/output statements may be re-directed to use the IEEE interface, instead of microdrive files, or the screen.

7.2.1 Syntax of Q-488 Channel Names.

To use these calls it is necessary to open a channel to the Q-488 IEEE system. This is done via the SuperBASIC **OPEN** statement. This requires a file name to be given which specifies the IEEE system. The syntax of the filename is:

IEEE[slot_number]_*[addresses]*_*[qualifiers]*

The **slot number** need only be given if you have more than one Q-488 interface (plugged into an expansion card for example). If the slot number is not given then the filename will refer to the Q-488 interface in the lowest numbered expansion slot.

The **addresses** indicate the addresses of the IEEE devices which are to be addressed when the channel is used. A maximum of 15 addresses may be quoted for an output channel, an input channel may only use a single address. If a channel with multiple addresses is used for input then only the last addressed device will be addressed to talk.

Each address consists of a primary address (in the range 0..30) and an optional secondary address (also in the range 0..30). If the secondary address is given then it should be separated from the primary address by an 'S' character. Multiple addresses should be separated by underscore characters.

If no addresses are quoted whatsoever then a 'null' channel will be opened. This type of channel is not intended to be used for straight data transfer, rather to allow the interface to be driven directly from low level commands. (It is advisable to avoid using this sort of channel unless you really know what you are doing!)

After the addresses have been specified you may optionally include a number of **qualifiers** at the end of the filename. These are used to control a number of optional channel attributes.

Valid qualifiers are:

D This qualifier indicates that the interface should act as an IEEE device, rather than a controller. This mode of operation will be described fully in a subsequent section.

R This indicates that the channel should be a 'raw' channel, in other words, that it should transmit the characters sent through it literally, without translation. The normal channels translate the QDOS Newline character (CHR$(10)) into ASCII carriage return/linefeed combinations. If you want to send binary data over a channel you should use this type of channel.

E This controls the use of the EOI line in data transfers. It is followed by a single digit (which must be either 0 or 1). If the digit is zero then EOI is never set on output and is always ignored on input. If the digit is one then EOI will be set on the last character of each output string, and will be taken as a terminator if an input character is received with EOI set. The default action on a channel is equivalent to the option 'E1'.

T This option controls the use of terminator characters on input and output. The following options are available:

0 No characters are added to output strings or taken as terminators on input. (EOI must be present to terminate an input string).

1 No characters are added on output, but the line feed character (CHR$(10)) is used as an additional input terminator.

2 All line feeds in output strings are converted to CR LF combinations. On input line feed is still used as a terminator but any other control characters sent to the channel will be removed from the string read.

Examples of valid IEEE file names:

IEEE	A 'null' channel.
IEEE_5	A channel to IEEE device 5.
IEEE3_5S4	A channel to device 5, secondary address 4, using the interface in slot 3.
IEEE_1_2S17_4	A channel to device 1, device 2 (secondary address 17) and device 4, using the interface in the lowest numbered slot.
IEEE_D	A 'device' channel.
IEEE_5_R	A 'raw' channel to device 5.

7.2.2 SuperBASIC Commands used by the Standard System.

The SuperBASIC statements **INPUT#** and **PRINT#** provide the basic method of transferring data on the IEEE bus. All values will be transferred as printable ASCII character strings, which is the format expected by most IEEE devices.

7.3 LOW LEVEL COMMANDS.

7.3.1 I/O Trap for Low Level Commands.

The low level interface to the IEEE system is accessed via the special TRAP #3 call, IB.CMND. This trap call uses D1.B to indicate a subsidiary function code, and where necessary, uses D2 for additional arguments:

TRAP #3 D0=$60 **IB.CMND**

Issue a low level IEEE command

Call parameters		Return parameters	
D1.B	function code	D1	result or ???
D2	parameter	D2	result or ???
D3.W	timeout	D3.L	preserved
A0	channel id	A0	preserved
A1		A1	???
A2		A2	preserved

Error returns:

NC not complete
NO channel not open
NI not implemented (IB.PPOLL in early versions)

Most of these trap calls may be accessed from SuperBASIC by means of additional SuperBASIC keywords provided in the IEEE ROM.

7.3.2 Summary of Low Level Commands.

Function codes values:-			SuperBASIC equivalent:-
CM.TCS	$00		IB_TCS #channel
CM.TCA	$01		IB_TCA #channel
CM.GTS	$03		IB_GTS #channel
CM.RQC	$04		IB_RQC #channel
CM.RLC	$05		IB_RLC #channel
CM.EOI	$06		IB_EOI #channel
CM.UNT	$07		IB_UNT #channel
CM.UNL	$08		IB_UNL #channel
CM.GET	$09		IB_GET #channel
CM.TCS	$0A		IB_TCS #channel
CM.TCT	$0B		IB_TCT #channel
CM.GTL	$0C		IB_GTL #channel
CM.DCL	$0D		IB_DCL #channel
CM.LLO	$0E		IB_LLO #channel
CM.PPC	$0F		IB_PPC #channel
CM.PPU	$10		IB_PPU #channel
CM.SPE	$11		IB_SPE #channel
CM.SPD	$12		IB_SPD #channel
CM.REN	$20	D2 = flag	IB_REN #channel,flag
CM.LON	$21	D2 = flag	IB_LON #channel,flag
CM.TON	$22	D2 = flag	IB_TON #channel,flag
CM.TAG	$30	D2 = address	IB_TAG #channel,address
CM.LAG	$31	D2 = address	IB_LAG #channel,address
CM.SCG	$32	D2 = address	IB_SCG #channel,address
CM.PPE	$33	D2 = sense/bit	IB_PPE #channel,sense,bit
CM.ADRT	$40		IB_ADRT #channel
CM.ADRL	$41		IB_ADRL #channel
CM.BGET	$42	D1 = response	res = IB_BGET(#channel)
CM.BPUT	$43	D2 = byte	IB_BPUT #channel,byte
CM.DEV	$44	D2 = address	IB_DEV #channel,address
CM.SPR	$45	D2 = response	IB_SPR #channel,response
CM.UNADD	$46		IB_UNADDR #channel
CM.CNTRL	$47	D1 = result	flag = IB_CNTRL(#channel)
CM.SYSCN	$48	D1 = result	flag = IB_SYSCN(#channel)
CM.RESET	$49		IB_RESET #channel
CM.STERM	$4A	D2=terminators	IB_STERM #channel,*[term]*
CM.NSTER	$4B	D2 = number	IB_NSTER #channel,number
CM.RTERM	$4C	D2 = term	IB_RTERM #channel,[term]
CM.NRTER	$4D	D2 = number	IB_NRTER #channel,number
CM.SEOI	$4E	D2 = flag	IB_SEOI #channel,flag
CM.REOI	$4F	D2 = flag	IB_REOI #channel,flag
CM.RQC	$50	D2 = flag	IB_RQC #channel,address
CM.IFC	$51		IB_IFC #channel
CM.CCST	$52	D2 = flag	IB_CCST #channel,flag
CM.SPOLL	$53	D1 = response	response = IB_SPOLL(#channel)
CM.PPOLL	$54	D1 = response	response = IB_PPOLL(#channel)
CM.ADDRT	$55		
CM.ADDRL	$56		

7.2.4 Low Level Command Definitions.

TAKE CONTROL SYNCHRONOUSLY (queued)

SuperBASIC:

IB_TCS #channel

Trap call:

D1 = $00 CM.TCS

Description:

Queues a command to take control of the bus. This is done synchronously, i.e. without interrupting any byte transfers currently taking place. The interface must be the active controller.

TAKE CONTROL ASYNCHRONOUSLY (queued)

SuperBASIC:

IB_TCA #channel

Trap call:

D1 = $01 CM.TCA

Description:

Queues a command to take control of the bus. This is done asynchronously, and so data transfers may be interrupted. The interface must be the active controller.

GO TO STANDBY (queued)

SuperBASIC:

IB_GTS #channel

Trap call:

D1 = $03 CM.GTS

Description:

Releases control of the bus (sets ATN false). The interface remains the current controller.

REQUEST CONTROL (queued)

SuperBASIC:

IB_RQC #channel

Trap call:

D1 = $04 CM.RQC

Description:

Take control of the bus after control has been passed from another controller by means of the transfer control command. This command will not normally be needed since the Q-488 interface handles the Pass Control command sequence automatically.

RELEASE CONTROL (queued)

SuperBASIC:

IB_RLC #channel

Trap call:

D1 = $05 CM.RLC

Description:

Release control of the bus after having transferred control to another device by means of the TCT command. This command will not normally be needed since the Q-488 interface handles the Pass Control command sequence automatically.

SEND EOI ON NEXT BYTE (queued)

SuperBASIC:

IB_EOI #channel

Trap call:

D1 = $06 CM.EOI

Description:

Force End or Identify (EOI) on the next date byte to be output.

UNTALK (queued, takes control)

SuperBASIC:

IB_UNT #channel

Trap call:

D1 = $07 CM.UNT

Description:

Send the Untalk command ($5F) on the bus. The command will take control of the bus if it is not already in control.

UNLISTEN (queued, takes control)

SuperBASIC:

IB_UNL #channel

Trap call:

D1 = $08 CM.UNL

Description:

Send the Unlisten command ($3F) on the bus. The command will take control of the bus if neccesary.

GROUP EXECUTE TRIGGER (queued, addressed, takes control)

SuperBASIC:

IB_GET #channel

Trap call:

D1 = $09 CM.GET

Description:

Send the Group Execute Trigger command ($08) on the bus. The command will take control of the bus, and address the devices on the channel (if any) before sending the command.

SELECTIVE DEVICE CLEAR (queued, addressed, takes control)

SuperBASIC:

IB_TCS #channel

Trap call:

D1 = $0A CM.TCS

Description:

Send the Selective Device Clear command ($04) on the bus. The command will take control of the bus, and address the devices on the channel (if any) before sending the command.

TRANSFER CONTROL (queued, addressed)

SuperBASIC:

IB_TCT #channel

Trap call:

D1 = $0B CM.TCT

Description:

Transfer Control of the bus to another device. The command takes control of the bus, addresses the device on the channel as a talker, sends the Transfer Control command ($09) to the new controller then releases control of the bus. This command handles automatically the entire Transfer Control protocol.

GO TO LOCAL CONTROL (queued, addressed, takes control)

SuperBASIC:

IB_GTL #channel

Trap call:

D1 = $0C CM.GTL

Description:

Sends the Go To Local command ($01) to a device or group of devices. The command will take control of the bus if necessary, and address the devices on the channel as listeners.

DEVICE CLEAR (queued, takes control)

SuperBASIC:

IB_DCL #channel

Trap call:

D1 = $0D CM.DCL

Description:

Sends the Device Clear command ($14) over the bus. The command will take control of the bus if necessary.

LOCAL LOCKOUT (queued, takes control)

SuperBASIC:

IB_LLO #channel

Trap call:

D1 = $0E CM.LLO

Description:

Sends the Local Lockout command ($11) over the bus. The command will take control of the bus if necessary.

PARALLEL POLL CONFIGURE (queued, addressed, takes control)

SuperBASIC:

IB_PPC #channel

Trap call:

D1 = $0F CM.PPC

Description:

Sends the Parallel Poll Configure command ($05) over the bus. The command will take control of the bus if necessary.

PARALLEL POLL UNCONFIGURE (queued, takes control)

SuperBASIC:

IB_PPU #channel

Trap call:

D1 = $10 CM.PPU

Description:

Sends the Parallel Poll Unconfigure command ($15) over the bus. The command will take control of the bus if necessary.

SERIAL POLL ENABLE (queued, takes control)

SuperBASIC:

IB_SPE #channel

Trap call:

D1 = $11 CM.SPE

Description:

Sends the Serial Poll Enable command ($18) over the bus. The command takes control of the bus if necessary.

SERIAL POLL DISABLE (queued, takes control)

SuperBASIC:

IB_SPD #channel

Trap call:

D1 = $12 CM.SPD

Description:

Sends the Serial Poll Disable command ($19) over the bus. The command takes control of the bus if necessary.

SET/CLEAR REMOTE ENABLE (queued, system controller)

SuperBASIC:

IB_REN #channel,flag

Trap call:

```
D1 = #20        CM.REN
D2 = flag
```

Description:

Sets or clears the Remote Enable line on the bus. This command may only be executed by the System Controller.

SET/CLEAR LISTEN ONLY (queued)

SuperBASIC:

IB_LON #channel,flag

Trap call:

```
D1 = #21        CM.LON
D2 = flag
```

Description:

Sets or clears listen only mode in the 9914. This command should be used with great caution (if at all)!

SET/CLEAR TALK ONLY (queued)

SuperBASIC:

IB_TON #channel,flag

Trap call:

```
D1 = #22        CM.TON
D2 = flag
```

Description:

Sets or clears talk only mode in the 9914. This command should be used with great caution (if at all)!

TALKER ADDRESS GROUP (queued, takes control)

SuperBASIC:

IB_TAG #channel,address

Trap call:

```
D1 = $30        CM.TAG
D2 = address
```

Description:

Sends a Talker Address Group command over the bus. The actual command byte sent is ($40 OR address). The command takes control if necessary.

LISTENER ADDRESS GROUP (queued, takes control)

SuperBASIC:

IB_LAG #channel,address

Trap call:

```
D1 = $31        CM.LAG
D2 = address
```

Description:

Sends a Listener Address Group command over the bus. The actual command byte sent is ($20 OR address). The command takes control if necessary.

SECONDARY COMMAND GROUP (queued, takes control)

SuperBASIC:

IB_SCG #channel,address

Trap call:

```
D1 = $32        CM.SCG
D2 = address
```

Description:

Sends a Secondary Command Group command over the bus. The actual command byte sent is ($60 OR address). The command takes control if necessary.

PARALLEL POLL ENABLE (queued)

SuperBASIC:

IB_PPE #channel,sense,bit

Trap call:

D1 = $33 CM.PPE
D2 = sense*8+bit

Description:

Sends the Parallel Poll Enable command ($6X) over the bus. The sense bit indicates the sense of the desired parallel poll response, and the response indicates the bit number on which a parallel poll response is required.

ADDRESS AS A TALKER (queued, takes control)

SuperBASIC:

IB_ADRT #channel

Trap call:

D1 = $40 CM.ADRT

Description:

Addresses the device on the channel as a talker. The command takes control of the bus if necessary.

ADDRESS AS A LISTENER (queued)

SuperBASIC:

IB_ADRL #channel

Trap call:

D1 = $41 CM.ADRL

Description:

Addresses the device(s) on the channel as listener(s). The command takes control of the bus if necessary.

GET BYTE (flushes queue)

SuperBASIC:

```
res = IB_BGET(#channel)
```

Trap call:

```
D1 = $42        CM.BGET
result returned in D1.B
```

Description:

Reads a byte from the bus. The command will, as a side effect, also cause the output queue to be emptied.

PUT BYTE (queued)

SuperBASIC:

```
IB_BPUT #channel,byte
```

Trap call:

```
D1 = $43        CM.BPUT
D2 = byte
```

Description:

Outputs a byte onto the bus.

SET DEVICE ADDRESS (queued,device)

SuperBASIC:

```
IB_DEV #channel,address
```

Trap call:

```
D1 = $44        CM.DEV
D2 = address
```

Description:

Sets the primary address of the Q-488 interface. This is only important when the interface is no longer the active controller and can be addressed as a device.

SET SERIAL POLL RESPONSE (queued,device)

SuperBASIC:

IB_SPR #channel,response

Trap call:

D1 = $45 CM.SPR
D2 = response

Description:

Sets the Serial Poll Response of the interface. This is returned when the interface is serial polled. If bit 6 of the response is set then the interface will assert the SRQ line to request service. The SRQ line is deasserted automatically when the interface has been serial polled.

UNADDRESS CHANNEL (queued, takes control)

SuperBASIC:

IB_UNADDR #channel

Trap call:

D1 = $46 CM.UNADD

Description:

Sends an Unlisten or Untalk command (as appropriate) to unaddress any addressed devices on the bus. The command takes control of the bus if necessary.

RETURN CONTROLLER STATUS (immediate)

SuperBASIC:

flag = IB_CNTRL(#channel)

Trap call:

D1 = $47 CM.CNTRL
result returned in D1

Description:

This function reads the current controller status of the interface. The value TRUE (1) is returned if the interface is the currently active controller, or FALSE (0) if it is not. The controller status changes during the Transfer Control sequence.

RETURN SYSTEM CONTROLLER STATUS (immediate)

SuperBASIC:

flag = IB_SYSCN(#channel)

Trap call:

D1 = $48 CM.SYSCN
result returned in D1

Description:

This function reads the system controller status of the interface. The value TRUE (1) is returned if the interface is the system controller, or FALSE (0) if it is not. The system controller status is determined when the QL is first booted (or when an IB_RESET command is executed). The state of the REN line is tested, if it is false then the interface will consider itself to be the system controller.

RESET INTERFACE (clears queue)

SuperBASIC:

IB_RESET #channel

Trap call:

D1 = $49 CM.RESET

Description:

Re-initialise the interface. The output queue is cleared and the controller and system controller status is reset. The default terminator characters are reset to their default values. The command does not close any channels which are open, nor does it change the terminator characters of these channels.

SET SEND TERMINATORS (immediate)

SuperBASIC:

IB_STERM #channel,*[term]* (up to 4 terminators)

Trap call:

```
D1   = $4A          CM.STERM
D2.L = terminators
```

Description:

Set the terminator characters which are to be appended to output strings. The characters should be packed into the high order end of D2. The number of terminator characters is set separately. If the channel is opened to a null channel then the default terminators for the interface will be changed rather than those for the particular channel.

SET NUMBER OF SEND TERMINATORS (immediate)

SuperBASIC:

IB_NSTER #channel,number

Trap call:

```
D1 = $4B       CM.NSTER
D2 = number
```

Description:

Indicates the number of terminator characters to append to output strings (max = 4). The default number of terminator characters for the interface may be set by sending this command to a null channel. Note that most of the widely used input and output terminator character combinations may be selected in the when opening a channel by means of the 'T' option.

SET RECEIVE TERMINATOR CHARACTER (immediate)

SuperBASIC:

IB_RTERM #channel[,term]

Trap call:

D1 = $4C CM.RTERM
D2 = term

Description:

Indicates the receive terminator character. Either 0 or 1 terminator characters may be selected for input. Default input terminators may be set using a null channel.

SET NUMBER OF RECEIVE TERMINATORS (immediate)

SuperBASIC:

IB_NRTER #channel,number

Trap call:

D1 = $4D CM.NRTER
D2 = number

Description:

Indicates the number of input terminator characters (0 or 1).

SEND EOI CONTROL (immediate)

SuperBASIC:

IB_SEOI #channel,flag

Trap call:

D1 = $4E CM.SEOI
D2 = flag

Description:

Controls whether EOI is sent on the last byte of an output string. A flag value of TRUE (1) means that EOI will be set, a value of FALSE that it will not.

RECEIVE EOI CONTROL (immediate)

SuperBASIC:

IB_REOI #channel,flag

Trap call:

D1 = $4F CM.REOI
D2 = flag

Description:

Controls whether EOI is ignored on input. A flag value of TRUE (1) means that EOI is taken as an input terminator, a value of FALSE that it will not.

RAW CHANNEL CONTROL (immediate)

SuperBASIC:

IB_RCC #channel,address

Trap call:

D1 = $50 CM.RCC
D2 = flag

Description:

Controls whether QL newline characters (CHR$(10)) are converted to ASCII Carriage Return-Linefeed in output strings. A flag value of TRUE (1) means that the translation is performed, a value of FALSE (0) that it is not.

SEND INTERFACE CLEAR (queued, system controller)

SuperBASIC:

IB_IFC #channel

Trap call:

D1 = $51 CM.IFC

Description:

Sends an interface clear message over the bus. The IFC line is asserted for about 100 microseconds. All devices will unaddress and the system controller will regain control. This command may only be executed by the system controller.

READ SRQ LINE

Syntax:

```
flag = IB_SRQ(#channel)
```

Description:

Returns the current state (TRUE or FALSE) of the SRQ line. This call is a special case of the preceding IB_LINES call and so no trap call is provided.

LOGICAL VALUES

Syntax:

```
logical = TRUE
logical = FALSE
```

Description:

Two convenient functions returning the values 1 and 0 respectively.

CONTROL CHARACTER STRIPPING (immediate)

SuperBASIC:

IB_CCST #channel,flag

Trap call:

D1 = $52 CM.CCST
D2 = flag

Description:

Controls the stripping of control characters on input. If the flag value is TRUE (1) then all input characters with ASCII values less than 32 (i.e. the control characters) will be removed from the input string. If a terminator character is set it will, however be returned as the last character of the string read. If the flag value is FALSE (0) then no characters will be removed from input strings.

EXECUTE SERIAL POLL (flushes queue, takes control)

SuperBASIC:

response = IB_SPOLL(#channel)

Trap call:

D1 = $53 CM.SPOLL
response returned in D1

Description:

Addresses the device on the channel and performs a serial poll sequence and returns the poll response.

EXECUTE PARALLEL POLL (immediate, takes control)

SuperBASIC:

```
response = IB_PPOLL(#channel)
```

Trap call:

```
D1 = $54        CM.PPOLL
response returned in D1
```

Description:

Executes a parallel poll sequence and returns the result. The polling takes place immediately, temporarily suspending any bus transactions in progress.

(Not implemented before V1.00)

ADDRESS AS A TALKER (NO GTS) (queued, takes control)

SuperBASIC:

not provided

Trap call:

```
D1 = $55        CM.ADDRT
```

Description:

Addresses a channel as a talker but does not then go to standby.

ADDRESS AS A LISTENER (NO GTS) (queued)

SuperBASIC:

not provided

Trap call:

```
D1 = $56        CM.ADDRL
```

Description:

Addresses a channel as a listener but does not then go to standby.

7.4 BUS ANALYSIS FUNCTIONS.

ACTIVATE BUS ANALYSER

Syntax:

ANALYSE [#channel,] [x0,[y0,[mode]]]

Description:

Activate a bus analyser. If the channel number is given then the analyser will be attached to that channel, otherwise a private channel will be opened. 'x0' and 'y0', if given, specify the initial pixel coordinates of the top left corner of the analyser window; 'mode' specifies the initial mode of operation of the analyser. A mode of 1 indicates Single-Step operation, 0 indicates free-running and -1 indicates slow-motion operation. This command is described more fully in chapter 6.

READ BUS LINES

Syntax:

lines = IB_LINES(#channel) (Word result)

Trap Call:

IB.LINES D0 = $52

Returns current bus lines in D1.L

Description:

Returns the current state of the IEEE bus lines. The lines are returned in the following bits of the result:

Bit 15			...				Bit 8
ATN	DAV	NDAC	NRFD	EOI	SRQ	IFC	REN
Bit 7			...				Bit 0
DIO8	DIO7		...			DIO2	DIO1

The upper word of D1 in the trap call will return the internal Q-488 code for the operation currently at the head of the output queue. This is used internally by the bus analyser but it is not expected to be of use to the ordinary user.

A

Acceptors....2-3
Active Controller....2-3
Address as a listener (no GTS)....7-22
Address as Listener....7-14
Address as Talker....7-14
(No GTS)....7-22
Addressing Commands....2-4
Advanced Use....5-1
ANALYSE....7-23
Example....3-2
Activating....6-3
Activating Input Cursor....6-4
Channel....6-3
Deactivating....6-5
Demonstration....6-2
Display Window Layout....6-3
Free-running Mode....6-5
Mode of Operation....6-3
Moving the Window....6-5
Multiple....6-3
Operation Modes....6-1
Redraw Display....6-5
Single-step Mode....6-5
Slow-motion Mode....6-5
Syntax....6-3
the Function Keys....6-5
Window Origin....6-3
ANSI....2-1
Apple....2-2
Assembly Language....5-11
ATN (see "Attention")....2-3
Attention (ATN)....2-3

B

Bad Bus States....2-9
BGET £....4-7
Block Transfers....7-3
Bold Type; Use in Examples....1-1
BPUT £....4-6
Bus Analyser....7-23
Bus Analysers....6-1
Bus Commands....2-3
Bus Transactions; Hanging....6-1, 6-2
Bus Transactions; Monitoring....6-1
Bus Transactions; Typical....6-1

C

C Language....5-11
Cable Adaptor....3-3
Carriage Return Character....5-4
Changing Default Options....5-2
Changing Options....5-2
Channel Number....4-3
Channel Numbers....4-2
Character Translation....5-1
CLOSE £....4-2
CLS £....4-6
CM.ADDRL....7-22
CM.ADDRT....7-22
CM.ADRL....7-14
CM.ADRT....7-14
CM.BGET....7-15
CM.BPUT....7-15
CM.CCST....7-21
CM.CNTRL....7-16
CM.DCL....7-10
CM.DEV....7-15
CM.EOI....7-7
CM.GET....7-8
CM.GTL....7-9
CM.GTS....7-6
CM.IFC....7-20
CM.LAG....7-13
CM.LLO....7-10
CM.LON....7-12
CM.NRTER....7-19
CM.NSTER....7-18
CM.PPC....7-10
CM.PPE....7-14
CM.PPOLL....7-22
CM.PPU....7-11
CM.RCC....7-20
CM.REN....7-12
CM.REOI....7-20
CM.RESET....7-17
CM.RLC....7-7
CM.RQC....7-7
CM.RTERM....7-19
CM.SCG....7-13
CM.SDC....7-9
CM.SEOI....7-19
CM.SPD....7-11
CM.SPE....7-11
CM.SPOLL....7-21
CM.SPR....7-16
CM.STERM....7-18
CM.SYSCN....7-17
CM.TAG....7-13
CM.TCA....7-6
CM.TCS....7-6
CM.TCT....7-9

CM.TON.....7-12
CM.UNADD.....7-16
CM.UNL.....7-8
CM.UNT.....7-8
Connecting Cable; Q-488.....3-3
Connecting Leads; IEEE.....3-4
Connectors; Stacking IEEE.....3-4
Control Character Stripping.....7-21
Control Lines.....2-2
Control of EOI Line.....5-3
Controller.....2-3, 5-15
COPY.....4-8
CST Disc Interface.....4-1
Customised Systems.....1-1

D

DAC (see "Data Accepted").....2-7
Data Accepted (DAC).....2-7
Data Bytes.....2-3
Data Lines.....2-2
Data Transfer between Computers.....5-19
Data Valid.....2-7
DAV (see "Data Valid").....2-7
Default Options.....5-2
Device Addresses.....2-4, 3-4, 4-4
Device Characteristics.....5-1
Device Clear.....7-10
Device Mode Channels.....7-2
Device Mode Operation.....5-15
Device Name Syntax.....7-1

E

End Or Identify (EOI).....2-5
EOI (see "End Or Identify").....2-5
EOI Control.....7-2
    on receive.....7-20
    on send.....7-19
EOI Line.....5-3
Execute Serial Poll.....7-21
Expansion Units.....3-1
EXTRAS.....3-3

F

FALSE.....3-3, 2-7
Fault Finding.....3-2
File Name Syntax.....7-1
Filename Options.....5-1
Filename Qualifiers.....7-2
Front Panel Controls.....2-6

G

General Purpose Interface Bus (GPIB).....2-1
GET (see "Group Execute Trigger").....2-6
Get Byte.....7-15
Go To Local (GTL).....2-6
Go to Local Control.....7-9
Go To Standby.....7-6
GPIB.....2-1
Group Execute Trigger.....7-8
Group Execute Trigger (GET).....2-6
GTL (see "Go To Local").....2-6

H

Handshaking.....2-3, 2-7
    Diagram.....2-8
Hewlett-Packard Interface Bus (HPIB).....2-1
Hewlett-Packard Plotter; Address.....4-1
Hewlett-Packard Plotters.....4-1
High (Voltage Level).....2-7
HPIB.....2-1
Hyphenation; Use in Examples.....1-1

I

IB.CMND Trap.....7-4
IB_ADRL.....7-14
IB_ADRT.....7-14
IB_BGET.....7-15
IB_BPUT.....7-15
IB_CCST.....7-21
IB_CNTRL.....7-16
IB_DCL.....7-10
IB_DEV.....7-15
IB_EOI.....7-7
IB_GET.....7-8
IB_GTL.....7-9
IB_GTS.....7-6
IB_IFC.....7-20
IB_LAG.....7-13
IB_LINES.....7-23
IB_LLO.....7-10
IB_LON.....7-12
IB_NRTER.....7-19
IB_NSTER.....7-18
IB_PPC.....7-10
IB_PPE.....7-14
IB_PPOLL.....7-22
IB_PPU.....7-11
IB_RCC.....7-20
IB_REN.....7-12
IB_REOI.....7-20
IB_RESET.....3-3, 4-7, 7-17
IB_RLC.....7-7
IB_RQC.....7-7
IB_RTERM.....7-19
IB_SCG.....7-13

IB_SDC....7-9
IB_SEOI....7-19
IB_SPD....7-11
IB_SPE....7-11
IB_SPOLL....5-8, 7-21
IB_SPR....7-16
IB_SRQ....5-7, 7-24
IB_STERM....7-18
IB_SYSCN....7-17
IB_TAG....7-13
IB_TCA....7-6
IB_TCS....7-6
IB_TCT....7-9
IB_TON....7-12
IB_UNADDR....7-16
IB_UNL....7-8
IB_UNT....7-8
IEC-625....2-1
    Adaptors....2-1
IEEE Connecting Leads....3-4
IEEE Device Addresses....4-4
IEEE Devices Available....4-1
IEEE-488....2-1
    Connectors....2-2
    Interface....2-1
IFC (see "Interface Clear")....2-7
INKEY$....4-7, 7-3
INPUT £....4-6, 7-3
Input Terminator Characters....5-3, 7-2
Installation of Q-488....3-1
Instruments Available....2-1
Intelligent Controller....5-17
Interface Clear (IFC)....2-7, 2-7, 7-20
Interference....2-2
Interrupt on SRQ....2-6

L

LBYTES....7-3
Line Feed Character....5-4
Line Terminators....5-20
Linear Wiring....2-2, 3-4
LISP....5-11
Listen Only....7-12
Listener Address....7-13
Listener Addresses....2-4
Listener Addressing....2-4
Listeners....2-3
LLO (see "Local Lock Out")....2-6
Local Control State....2-6
Local Lock Out (LLO)....2-6
Local Lockout....7-10
Logic Levels....2-7
Low (Voltage Level)....2-7
Low Level Commands....7-4

M

Maximum Cable Length....2-3
MLA (see "My Listen Address")....2-4
MTA (see "My Talk Address")....2-4
Multiple Controller Systems....2-2, 2-7, 5-16
Multiple Listener Addresses....5-4
Multiple Q-488 Interfaces....3-1
My Listen Address (MLA)....2-4
My Talk Address (MTA)....2-4

N

Negative Logic....2-7
New Line Character....5-4
Not Ready For Data....2-7
NRFD (see "Not Ready For Data")....2-7
Null Channels....7-2

O

OPEN £....3-2, 4-2
Open Collector Drivers....2-7
Options....5-1, 5-2, 7-2
Other Languages....5-1, 5-11
Output Queue....4-5
Output Terminator Characters....5-3, 7-2

P

Parallel Poll....2-5, 2-5, 5-7, 7-22
  Configure....7-10
  Unconfigure....7-11
Pascal....5-11
PET....2-2
Polling....5-6
Powering Up....3-1
PRINT £....3-2, 4-4, 7-3
Put Byte....7-15

Q

Q-488 Connecting Cable....3-3
Q-488 Installation....3-1
Q-488 Multiple Interfaces....3-1
QL Device; IEEE Syntax....4-3
QL Toolkit....4-1
Qualifiers....5-2, 7-2
Queue; Output....4-5, 6-1

R

Raw Channel Control....7-20
Raw Channels....5-3, 7-2
Read SRQ Line....7-24
Ready For Data (RFD)....2-7
Rear Panel Address Switches....2-4
Receiving Data....4-6
Release Control....7-7
Remote Control State....2-6
Remote Enable (REN)....2-6, 2-7

REN (see "Remote Enable").....2-6, 2-7
REN Control.....7-12
Request Control.....7-7
Reset Interface.....7-17
Return 7-17
Return Controller Status.....7-16
RFD (see "Ready For Data").....2-7

S

SBYTES.....7-3
Secondary Address.....2-4, 2-5, 5-4, 7-13
Selective Device Clear.....7-9
Send EOI on Next Byte.....7-7
Send Interface Clear.....7-20
Sending Data.....4-4
Serial Poll.....2-5, 5-7
Serial Poll Disable (SPD).....2-5, 7-11
Serial Poll Enable (SPE).....2-5, 7-11
Serial Poll Mode.....2-5
Serial Poll Response.....7-16
Service Request (SRQ).....2-6, 5-6
Set Device Address.....7-15
Set Number of Receive Terminators.....7-19
Set Number of Send Terminators.....7-18
Set Receive Terminator.....7-19
Set Send Terminators.....7-18
Setting Options.....5-1
Signal Lines.....2-2
Single Byte Transfers.....7-3
Slot Number.....7-1
Sources.....2-3
SPD (see "Serial Poll Disable").....2-5
SPE (see "Serial Poll Enable").....2-5
Specialised Requirements.....1-1
SRQ (see "Service Request").....2-6
SRQ (Service Request).....5-6
Stacking IEEE Connectors.....3-4
Star Wiring.....2-2, 3-4
Start-Up Message.....3-1
Status Byte.....2-5
    Bit 6.....2-6
Status Word.....2-5
Subsets.....2-2
SuperBASIC Extensions.....4-1
SuperBASIC Keywords.....3-3
System Controller.....2-7, 2-7, 5-16

T

Take Control Asynchronously.....7-6
Take Control Synchronously.....7-6
Talk Only.....7-12
Talker Address.....2-4, 7-13
Talkers.....2-3
TCT (see "Transfer Control").....2-7
Terminator Characters.....5-3, 5-20, 7-2
Text Files.....5-20
Transfer Control (TCT).....2-7, 7-9
Transfer Rates.....2-1
Transferring Binary Data.....5-3
Transferring Text Files.....5-20
TRAP Calls.....7-4
Tri-State Buffers.....2-7
Triggering Devices.....2-6
TRUE.....2-7, 3-3

U

Unaddress Channel.....7-16
Unaddressing.....2-4
UNL (see "Unlisten").....2-4
Unlisten (UNL).....2-4, 7-8
UNT (see "Untalk").....2-4
Untalk (UNT).....2-4, 7-8
Use from other languages.....5-11

V

Voltage Levels.....2-7

W

Wired-OR.....2-7
Wiring Routes.....2-2, 3-4
    Linear.....3-4
    Star.....3-4