MEGA - TOOLBOX

for the Sinclair QL Computer

USER GUIDE

by Francesco Balena

Compware Mega-Toolbox: Issue 1 December 7, 1987

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

TABLE OF CONTENTS

1. Introduction

The *Mega-Toolbox* is not just another QL toolkit. It has been designed by an experienced programmer who has intimate knowledge of the Sinclair QL, specifically for use by other experienced programmers. Only a relatively small number of essential facilities can be found in other toolkits; the vast majority of features break new ground, and enable programs to be written in QL Superbasic that simply could not be contemplated before. Simply reading about the facilities available is likely to provide inspiration to programmers.

More than 170 new commands have been added, covering a very wide range of facilities. Throughout, the author has favoured speed of execution rather than compactness of code. The result is a relatively large toolkit, which makes very sophisticated programs possible albeit at the expense of precious RAM. To compensate for this, customised versions can be produced for individuals or software houses wishing to distribute software using the *Mega-Toolbox* (see chapter 7). By removing those facilities that are not required in any particular program, large reductions in the size of the *Mega-Toolbox* can be made, and in order to make the *Mega-Toolbox* even more appealing to such programmers, care has been taken to make it compatible with basic compilers.

1.1. How To Use This Manual

The number of commands and diversity of facilities available has made this manual into a substantial document, making it impractical for most people to read it from cover to cover in order to learn about the *Mega-Toolbox's* facilities. Even if you did have time to read the entire manual, there is so much information to be taken in that it would not be possible to remember it all. In order to tackle this problem the manual has been organised around three main chapters:-

Chapter 4 - Summary Of Toolbox Facilities

Briefly describes every new command, together with all commands of a similar nature, making it very easy to find out exactly what facilities the *Mega-Toolbox* provides in a particular area such as windows and graphics or file handling.

Chapter 5 - Description Of Commands And Syntax

Describes the function and detailed syntax of every command, including some examples and is organised alphabetically for quick reference.

Appendix A - Example Programs

Lists the example programs provided with the *Mega-Toolbox* and the commands which are included in them, making it very easy to find an example of the use of a particular command in an actual program.

These sections make it very quick and easy to find out if the *Mega-Toolbox* can help with a particular programming problem, what commands are available and how to use them.

The rest of the manual provides useful support including a description of the files supplied, how to install the *Mega-Toolbox* for use, notes about how the *Mega-Toolbox* performs certain functions, how to use the *Mega-Toolbox* in compiled programs and what to do if you experience problems. There is also a chapter explaining how you can arrange to use the *Mega-Toolbox* in your own commercial programs.

1.2. QDOS Specific Documentation

This manual provides QDOS specific information that is directly relevant to particular commands, but is unable to provide comprehensive details of QDOS which may be needed by some programmers. For example, although some commands enable you to access the Superbasic and system variable areas, and to find the location of channel and job definition headers, details of the contents of these areas are not provided. If you do require more detailed information about QDOS, we strongly recommend the following comprehensive reference book, which is available from Compware:

"QL Advanced User Guide" by Adrian Dickens, Adder Publishing,
ISBN 0 947929 00 2.

This text provides all the additional information required to use even the most QDOS specific *Mega-Toolbox* commands to the full.

2. What You Get

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

BOOT - a simple program to automatically load the *Mega-Toolbox.*

MEGA_BIN - the *Mega-Toolbox* itself.

MEGA_BIN% - an integer version of *Mega-Toolbox.*

BACKUP - a simple to use general purpose backup program.

README_DOC - a Quill document containing any errata to the manual.

The remaining files on the cartridge contain examples which are described in appendix A.

3. How To Install The Mega-Toolbox

Installing the *Mega-Toolbox* is a very simple process but before you proceed and start to explore its many facilities you should make a working copy of the program. Please **please** don't just press on and use your master cartridge, follow the instructions below closely.

3.1. Making A Working Copy

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

3.2. The BOOT Program

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

Once the *Mega-Toolbox* has been loaded in this way, all the facilities and new commands described in this manual will be available.

You can also modify the BOOT program to so that it starts another program after loading the *Mega-Toolbox,* but you should not include this code within a larger program, or in some instances commands will not be properly linked into Superbasic.

Chapter 6 gives details of how to customise the parameters used in the BOOT program.

3.3. Mega-Toolbox Demonstration

To see a demonstration of just some of the things that are possible with the *Mega-Toolbox,* insert your working cartridge in *mdv1_* and type *lrun mdv1_demo.*

4. Summary Of Mega-Toolbox Facilities

This chapter describes the features provided by the *Mega-Toolbox* which fall into the following main areas:

(1) Enhancements to the keyboard input driver providing easier command line editing.

(2) Improved control over QDOS resources (including memory allocation, keyboard input, file handling, pipes, job control, alarm clocks and tune playing jobs).

(3) Windows, text and graphics commands (including saving, restoring, copying and mirroring of windows as well as general drawing and text printing commands ideal for constructing animated slide-shows for games and advertising etc.).

(4) Commands for handling dual screens (screen copying, swapping and automated screen mode control).

The rest of this chapter summarises these commands and is divided into the following sections. (See chapter 5 for details of individual commands.)

4.1. Keyboard Input Enhancements

The following additional command line editing facilities have been added. In order to prevent clashes with special key combinations used by other programs - Quill for instance - which may occur when using programs enabling you to swap between basic and other jobs, these editing features can be selectively disabled using the KEY_USE command.

Key Combination	Function
SHIFT/ENTER	Recalls the last input command line for editing.
SHIFT/LEFT ARROW	Moves the cursor eight characters to the left.
SHIFT/RIGHT ARROW	Moves the cursor eight characters to the right.
ALT/LEFT ARROW	Moves the cursor to the start of the line.
ALT/RIGHT ARROW	Moves the cursor to the end of the line.
CTRL/SHIFT/LEFT ARROW	Deletes eight characters to the left.
CTRL/SHIFT/RIGHT ARROW	Deletes eight characters to the right.
CTRL/ALT/LEFT ARROW	Deletes from the cursor to start of the line.
CTRL/ALT/RIGHT ARROW	Deletes from the cursor to the end of the line.

4.2. QDOS Resources

4.2.1. Memory Management

Command name(s)	Description
FREE_MEM	Returns the amount of free memory.
ALCHP, ALCHP_L, RECHP, RESPR_L	Memory allocation and release including optional loading of files.
MCOPY, MEXCHANGE, MFILL, MFILL_W, MFILL_L	Fast memory manipulation.
EXCOPY, EXFILL	Copying and filling rectangular areas of screen memory.
STORE, FETCH	For storing and retrieving groups of values to/from RAM.
QPOKE, QPEEK$	Store and retrieve strings to/from memory in QDOS format.
PUTMEM, GETMEM$	Store and retrieve strings to/from memory as a raw string of bytes.
MSEARCH, MSEARCH_W, MSEARCH_A	Search for strings in a given area of memory.
DUMP	Prints memory contents in hexadecimal and ASCII.

4.2.2. File Handling

Command name(s)	Description
FILE_LEN()	Returns the length of a file.
FILE_DSPACE()	Returns the size of an exec'able program's data-space.
SPOS, FPOS()	Setting and reading the position of a file pointer.
FLUSH	Forces all buffers belonging to a file to be emptied.
PUT	Output mixed list of integer, float or string to a channel in QDOS format.
PUT$()	Returns a string of characters corresponding to the QDOS format of an expression.
GET, GET%, GET$()	Convert a string (or series of bytes read from a channel) from QDOS representation into an integer, float or string value.
INPUT$()	Read a given number of characters from a file.
XDIR	Extended directory listing showing file length, and if executable, data space.
XFORMAT	Multiple formatting of a medium with option to continue until given number of sectors are good.
FREE_SECT()	Returns the number of free sectors on a disc or microdrive.
COPY_X	Spooled file copying.

4.2.3. Keyboard Control

Command name(s)	Description
KEYBOARD	Allows keyboard characteristics to be modified (eg repeat speed, capslock, "queue character".)
ENTER	Sends strings of characters to the keyboard queue of a given channel.
ACTIVATE_Q	Connects the keyboard input stream to the input queue of a given channel.
CLEAR_Q	Empties the input queue of a given channel.
PREFETCH$	Returns the next character available on an input channel, but leaves the character to be read properly by another command such as INKEY$.
PROMPT$	Issues the user with a given prompt which may be edited or left unchanged before being returned when the user presses <ENTER>. This allows for prompted input of the form used by Quill commands.
INPUT_X(), ANSWER$()	Enable a basic program to continue executing whilst the user is prompted for input.
FN_KEY	Returns value indicating current state of function key in combination with CTRL, SHIFT or ALT.
STICK, STICK_READ, STICK_USE	Allow reading of X,Y coordinates and state of fire button/space bar under control of a joy-stick or the cursor keys. Coordinates can be restricted to within a given area, and allow multiplication factors to be applied to "movement" dependent on the state of CTRL, SHIFT and ALT.

4.2.4. Function Key Programming

Command name(s)	Description
KEY, KEYS	For programming a function key with a string of characters and listing the strings currently programmed.
KEY_USE	For enabling the function key settings with different combinations of the ALT, SHIFT and ENTER keys, and for controlling screen resolution when using dual screens.

4.2.5. Job Control

Command name(s)	Description
JOBS, JOB_ID, JOB$, JOB_ADDR, JOB_STAT	For interrogation of QDOS to list the jobs return the ID of a job, its name address or status.
REMOVE, SETPRIOR, SUSPEND, RELEASE	For removal of, setting priority of, suspending, and releasing, jobs.
EXECUTE, EXECUTE_W	Similar to EXEC and EXEC_W but allowing job priority and data size to be specified, and a string of characters to be sent to the job as if read from the keyboard.
WAIT_R, WAIT_S	For waiting until a given job is removed and suspended.

4.2.6. Special Jobs

Command name(s)	Description
CLOCK_X	Sets up a digital clock in a given window.
ALARM_X	Sets up alarms with optional display of a countdown and optional execution of a Basic command on completion.
PLAY_X	Sets up a job to play a melody defined by a string.
MULTITASK_X	Sets up a job which repeatedly calls a machine code routine.

4.2.7. Channel Control

Command name(s)	Description
CHAN_ID(), CHAN_ADDR()	For finding the QDOS ID and address of the channel control block, given the QL Basic channel number.
FREE_CHAN	Returns the number of the first free QL Basic channel.
PIPE_ID()	Enables pipes opened by QL Basic to be opened for input.
CONNECT, CONNECT_C	For connecting one QL Basic channel in place of another, enabling input and output to be redirected to the new channel.

4.3. Windows Text And Graphics

4.3.1. Window Management

Command name(s)	Description
CURS, CURS_INC	For enabling, disabling or setting the cursor increments for a given window.
FONT_USE, SET_FONT	For selecting and adding character founts.
ATTR	A single command for setting ink, paper, border, border colour, cursor size and fount of a given window.
INVERSE	For exchanging the paper and ink colours of a window.
XCUR(), YCUR()	Return the current character cursor positions.
ROWS(), COLUMNS()	Return the current character window dimensions.
XSIZE(), YSIZE()	Return the current window dimensions (in pixel units).
XPOS(), YPOS()	Return the position of the window's top left hand corner.
XINC(), YINC()	Return the current cursor increments.
XMAP(), YMAP()	Convert from graphic window coordinates to screen pixel coordinates.
BACKSPACE	For moving the cursor back a number of character positions.
COLOUR()	Returns the colour of a pixel using either screen or window coordinates.
SCR_ADDR()	Returns the memory address of a pixel using either screen or window coordinates.
WSAVE, WSAVE_C, WLOAD	For saving/loading the contents of a window, optionally in a compressed form.
EXPAND	Takes a compressed memory image and expands it to its original form.
EXPAND_L()	Similar to ALCHP_L, but used on images saved by WSAVE or WSAVE_C. The images are expanded during loading if they were saved in compressed form, enabling immediate use of the SHOWAT or ZOOM commands.

Table continued....

Window management continued.

Command name(s)	Description
HIDE(), HIDE_C()	For saving window contents on the common heap. Heap may be allocated automatically, or a given area specified, and contents may optionally be compressed. Heap can subsequently be released using RECHP or SHOW_R.
MEM_LENGTH()	Returns the length of an area allocated using HIDE() or HIDE_C(), useful if you want to subsequently copy the contents or save them in a file.
SHOW, SHOW_R	For retrieving the contents of a window "hidden" using HIDE or HIDE_C. The retrieved window can overwrite, show only selected colours, be AND'ed, be OR'ed, be XOR'ed or have colours swapped or masked as it is copied to the screen.
SHOW_S	For swapping the contents of a "hidden" window with the current window shown on the screen.
SHOWAT	For placing a stored image at a given position on the screen, optionally masked to appear within a given window. The image can be turned upside-down and/or mirrored left to right.
ZOOM	Will show a stored image at a given position on the screen, enlarging or reducing it by a given factor.
MRECOL	Re-colours a stored image according to a string.
PANW, SCROLLW	Versions of PAN and SCROLL which wrap around at the window boundaries.
PAN_X, SCROLL_X	Set up jobs to pan or scroll a given window.
PANW_X, SCROLLW_X	As PAN_X, SCROLL_X, but wrapping round at the window boundaries.
BORDER_X	Sets up a job to alternate the colours of a border at a given interval, enabling flashing borders to be implemented.
RECOL_X	Sets up a job to re-colour a window repeatedly at a given interval.
RECOLSHOW_X	Sets up a job to re-colour and show a stored image at a given interval.
WINDOW_OP	For AND'ing and optionally XOR'ing a window with two specified values. Used for quickly changing the colours in a window or suppressing flashing and so on.

4.3.2. Text And Graphics

Command name(s)	Description
PRINT_3D	Prints text in three dimensions with given depth.
PRINT_X	Sets up a job that prints a string a given number of times. The string can contain instructions to control colours, flashing, underlining, tab'ing, clearing the screen, clearing part of the window, position the cursor, scroll or pan the window, re-colour the window, change character size or founts, pause for a given delay or until the TAB key is pressed, insert a delay between printing each character and to display text as a rolling banner.
FPRINT_X	As PRINT_X, but the string is read from a file.
MPRINT	Prints a message of any given size with independent X and Y axis scaling.
SLIDE_X	Sets up a job which shows a sequence of stored images with a given delay between each display.
DISK	Draws filled circles or ellipses.
BOX	For drawing rectangles.
PAINT	For filling irregular shapes.

See also EXCOPY and EXFILL.

4.4. Dual Screen Control

The table summarises facilities provided for access to the QL's second screen.

Command name(s)	Description
DUAL_SCR, RESET_SCR	Reserve and de-allocate memory for/from a second screen.
DISPLAY	Controls a routine which acts every 50th of a second. This enables the source of the displayed screen to be selected, and its screen mode to be defined for up to four areas of the screen. Different areas of the screen can also be disabled.
DUAL_COPY	Enables either whole screens or windows to be copied from one screen to another.
DMODE	Returns the current contents of the display mode register which is used by the DISPLAY and KEY_USE commands.

4.5. Miscellaneous Functions

Command name(s)	Description
SIGN()	Returns the sign of a numeric string.
BTST(), BCLR(), BSET()	Bit test, clear and set.
ROL(), ROL_W(), ROL_L()	Byte word and long word binary rotations.
MIN(), MAX()	Return the minimum and maximum values in an argument list.
FACT()	Factorial.
DAYS()	Used for testing the validity of a date, and for calculating the number of days between two dates.
UPPER$(), LOWER$	Converts strings to upper/lower case.
CHOOSE$()	Returns the nth value from a given list.
COUNT()	Returns the number of occurrences of one string within another.
SEARCH(), SEARCH_N(), SUBSTR()	For searching one string for matches/non-matches in another string.
REVERSE$()	Returns a the reverse of a given string - i.e. the first character becomes the last etc.
RIGHT$()	Returns the N rightmost characters of a string.
STR$()	Concatenates a list of arguments into a string. eg PRINT #n,STR$(27,'E',TITLE$);
TRIM$()	Trims trailing blanks from a string. Can be used with REVERSE$ to trim leading blanks.
HEX(), BIN(), HEX$(), BIN$()	Conversion from hexadecimal or binary to decimal and vice versa.
DEC$()	Returns a string containing a floating point number rounded to a given number of digits up to the maximum stored by the QL. This also enables you to print out the full precision of the QL's floating point arithmetic.
FDEC$()	"Print Using" function, for controlling the format of numeric output.
SWAP	For swapping the values of two arguments, or rotating values round a list of arguments.

Table continued...

Miscellaneous functions continued.

Command name(s)	Description
QCALL	For calling machine code routines with given register values and returning the modified contents of registers.
QTRAP	As QCALL, but for executing a single QDOS trap.
QREG	Returns the contents of a 68008 processor register following use of QCALL or QTRAP
OPTION_USE	Enables functions that return values through their arguments to be used in a compiled program.
TEST_SET	Performs the 68000 TAS (test and set) instruction
BASIC_ADDR	For finding the address of the QL Basic storage area and for reading the values of individual interpreter working variables.
SYSTEM_ADDR	Used to find the location of the QL's system variables and to read the values of specified system variables.
VAR_ADDR	For finding the address of variables or expressions.
VAR_TYPE	For determining the type of a variable, expression or argument.
BUFF_ADDR	Returns the address of a permanent 1k buffer that can be used to exchange information between Basic or other programs.

5. Description Of Commands And Syntax

This chapter details the syntax and describes the function of all *Mega-Toolbox* commands, which are listed in alphabetical order for ease of reference. You should refer to chapter 4 in order to find out what commands are available for given types of function.

The notation used in the descriptions is as follows:-

Keywords are shown in upper case and arguments in lower case.

Optional arguments are enclosed within square brackets []

Parameters inside curly brackets {} may be repeated any number of times (which means they can also be omitted altogether).

In most cases, a *#n* (channel) parameter is optional, and unless otherwise stated, channel #1 will be used by default.

Multi-tasking commands have an _X suffix and in general set up a job whose name (as returned by JOB$) is the same as the command but without the suffix. For example, PAN_X sets up a job called PAN.

ACTIVATE_Q

See also:

CLEAR_Q, CURS, ENTER

Syntax:

```
ACTIVATE_Q #n
```

Function:

Links the console channel *#n* to the keyboard; if the keyboard buffer of the console channel which is currently linked to the keyboard contains some characters, they are transferred to the keyboard buffer of channel *#n*.

ALARM_X

See also:

CLOCK_X, PLAY_X

Syntax:

```
ALARM_X [#n,]time[,command$]
```

Function:

This command creates a job which waits for *time* seconds before it either issues an alarm sound or sends a command string to the Superbasic interpreter.

If *command$* is omitted, a beep sounds to indicate the alarm, otherwise the basic program will be stopped (by simulating a break) and the *command$* string will be sent to the basic interpreter and will be executed as if it had been typed on the keyboard. Compiled basic programs cannot be "broken" in this way, and so should not include a *command$* string.

If a channel number is given, the number of seconds remaining will be repeatedly printed in the window indicated by *#n*. In this case, the given channel should not be used for anything else.

ALCHP()

See also:

ALCHP_L(), FREE_MEM, RECHP, RESPR_L

Syntax:

```
address = ALCHP( bytes )
```

Function:

Allocates memory in the common heap and returns the address of the area, or -3 for "Out of memory".

ALCHP_L()

See also:

ALCHP(), FREE_MEM, RECHP, RECHP_L

Syntax:

```
address = ALCHP_L( filename )
```

Function:

Allocates memory in the common heap and loads a file into it; it may also return a negative QDOS error code for any I/O error (not found, in use, etc.).

ANSWER$()

See also:

INPUT_X(), PROMPT$()

Syntax:

```
string$ = ANSWER$( id )
```

Function:

Used in conjunction with INPUT_X() to allow input to be obtained from the user whilst a basic program continues to perform other activities.

INPUT_X() is used to set up the input and ANSWER$() returns the final result. For full details, see the explanation of INPUT_X().

ATTR

See also:

FONT_USE, SET_FONT

Syntax:

```
ATTR #n[,ink[,paper[,border[,bordercol[,csizex[,csizey[,font1[,font2]]]]]]]]
```

Function:

This command sets several window attributes in one go; the ink and paper colours, the border width and color, the size and the fonts of characters. Note that all arguments except the first one are optional. If border is specified the window is cleared (CLS), the cursor is disabled and the FLASH, UNDER, OVER and FILL attributes are reset.

Font1 and *font2* may be either the addresses of special fonts which you have provided, or small negative numbers in order to select fonts that have been linked into the *Mega-Toolbox* using FONT_USE.

A *bordercol* of 128 will draw the border with a transparent colour.

BACKSPACE

See also:

XCUR(), YCUR(), ROWS(), COLUMNS()

Syntax:

```
BACKSPACE #n[,steps]
```

Function:

Moves the current cursor position backwards by *steps* character positions (default is one). Stepping back past the start of the line causes the cursor to jump to the last column of the previous line unless the current line is the first line of a window, in which case the cursor position is unchanged.

BASIC_ADDR, BASIC_ADDR()

See also:

SYSTEM_ADDR, VAR_ADDR

Syntax:

```
address = BASIC_ADDR
address = BASIC_ADDR( n )
```

Function:

When used without an argument, BASIC_ADDR() returns the address of Superbasic's working variables; note that this value cannot be guaranteed to be correct if there is any other job in the system or if Superbasic itself allocates or de-allocates memory (using DIM, ALCHP etc.). Hence you should only use the latter form for accessing values held in this area.

In the latter form this function is equivalent to:

```
address = BASIC_ADDR + PEEK_L( BASIC_ADDR + n )
```

which is useful for reading the values of interpreter's working variables. For example, BASIC_ADDR(0) will return the address of the Superbasic buffer, and BASIC_ADDR(24) returns the address of the name table.

BCLR()

See also:

BSET(), BTST(), ROL()

Syntax:

```
value = BCLR( bit, number )
```

Function:

Resets bit *bit* in 32-bit value *number*. Bit 0 is the least significant bit.

BIN$()

See also:

BIN(), DEC$(), HEX$()

Syntax:

```
string$ = BIN$( number[,digits] )
```

Function:

Returns a string containing *digits* digits representing the value of *number* in binary. If *digits* is omitted, eight digits will be returned by default. For example, to find the binary representation of thirty one you would type:

```
PRINT BIN$(31)
```

BIN()

See also:

BIN$(), DEC$(), HEX()

Syntax:

```
value = BIN( a$ )
```

Function:

Returns the value of a binary coded string. Zero will always be returned when the string contains invalid characters. Leading and trailing blanks (space characters) must have been removed (see TRIM$()). For example, to find the decimal value corresponding to 011111 binary, you would type:

```
PRINT BIN("011111")
```

BORDER_X

See also:

PAN_X, PRINT_X, RECOL_X, SCROLL_X

Syntax:

```
BORDER_X [#n,]colour$[,times[,timeout]]
```

Function:

Sets up a job to repeatedly alter the border colour of a given window.

colour$

is a string containing any number and combinations of the characters 0 to 7, representing colour 0 to colour 7. The colour of the border will be changed to the colour corresponding to each number in the string in turn.

times

is the number of times that the job is to cycle through all the colours in the string. A value of -1 causes the process to be repeated indefinitely.

timeout

is the number of fiftieths of a second between each change in colour.

The width of the border will remain unchanged.

Examples:

```
BORDER_X #2,'70',-1,20
```
will create a flashing border

```
BORDER_X '01234567',10,50
```
will cycle through all the colours ten times, holding each colour for one second.

BOX

See also:

DISK, PAINT

Syntax:

```
BOX [#n,]x1,y1[,x2,y2]
```

Function:

Draws a rectangle given the graphic coordinates of the two extreme points of its diagonal. If only one pair of coordinates is supplied, the current graphics cursor position is taken as the second pair.

BSET()

See also

BCLR(), BTST(), ROL()

Syntax:

```
value = BSET( bit, number )
```

Function:

Sets bit *bit* in 32-bit value *number*. Bit 0 is the least significant bit.

BTST()

See also:

BCLR(), BSET(), ROL()

Syntax:

```
value = BTST( bit, number )
```

Function:

Tests bit *bit* in 32-bit value *number*, returning zero or one corresponding to the state of the bit. Bit 0 is the least significant bit.

BUFF_ADDR

see also:

TEST_SET()

Syntax:

```
address = BUFF_ADDR
```

Function:

Returns the address of a permanent 1K buffer which may be used for exchanging information between programs, jobs, or QTRAP instructions etc. You may find the TEST_SET() instruction useful for preventing clashes between multi-tasking jobs wanting access to the buffer at the same time.

CHAN_ADDR()

See also:

CHAN_ID()

Syntax:

```
address = CHAN_ADDR( #n )
```

Function:

Returns the address of the QDOS channel entry for channel *#n;* the organization of data in this table is depends on which kind of channel it is.

CHAN_ID()

See also:

CHAN_ADDR()

Syntax:

```
value = CHAN_ID( #n )
```

Function:

Returns the long word *id* of the QDOS channel corresponding to the Superbasic channel *#n*, or -6 if that channel is not open.

CHOOSE$()

See also:

MAX(), MIN()

Syntax:

```
value = CHOOSE$( n, arg {,arg})
```

Function:

Returns the nth argument in the list; *n=0* will select the first argument, *n=1* the second and so on. If *n* is negative the first argument is returned, and if it is greater than the length of the argument list, the last argument is returned. The *n* argument may be a logical expression, as in the following example:

```
PRINT "The result is ";
PRINT CHOOSE$( n>100, "less than or equal to", "greater than" );
PRINT "100"
```

In the above example, an IF statement has been replaced but CHOOSE$() may also be used to replace a SELECT statement as in:

```
DEFine FuNction MONTH$( n )
RETurn CHOOSE$( n-1, 'January', 'February', 'March',
   'April', 'May', 'June', 'July', 'August', 'September',
   'October', 'November', 'December' )
END DEFine
```

This has not only replaced a long SELECT block, but has also avoided the need to use a string array or set of DATA statements.

CLEAR_Q

See also:

ACTIVATE_Q, CURS, ENTER

Syntax:

CLEAR_Q [#n]

Function:

Empties the keyboard buffer of console channel *#n*, or if no channel number is given, of the channel which is currently linked to the keyboard (indicated by the flashing cursor).

CLOCK_X

See also:

ALARM_X

Syntax:

CLOCK_X #n[,mode]

Function:

Creates a digital clock in the window of *#n;* if *mode* is omitted or given as zero, time will be shown in the format "hh:mm:ss", if *mode* has any other value, the format will be "hh:mm". Channel *#n* should not be used for any other purpose.

COLOUR()

Syntax:

COLOUR([#n,] x, y)

Function:

Returns the colour of a pixel. If no channel number is given, the coordinates refer to pixel coordinates, i.e. referenced to the top left hand corner of the *screen*. If the channel number is given, then graphic coordinates are used with reference to the bottom left corner of the given *window*. The function works in both high and low resolution modes. In hi-res, white is returned as 6.

COLUMNS()

See also:

    ROWS()

Syntax:

```
value = COLUMNS( #n )
```

Function:

Returns the width of the given window in character units (which depends on the current character size set using CSIZE).

CONNECT, CONNECT_C

See also:

    PIPE_ID()

Syntax:

```
CONNECT #n, newid
CONNECT_C #n, newid
```

Function:

Replaces the current Superbasic channel id with a new value. This affects only the Superbasic and not the QDOS channel tables. All subsequent operations on channel *#n* will act on a different QDOS channel, providing a convenient way to access input pipes created by PIPE_ID(). Both CONNECT and CONNECT_C have the same effect, except that the latter also closes the QDOS channel previously linked to channel *#n*. The following example reads a disc directory into an array.

```
10 OPEN #3,pipe_5000 : OPEN #4,scr : REMark a dummy channel
20 CONNECT_C #4,PIPE_ID( #3 )
30 DIR #3 : CLOSE #3 : DIM file$( 50, 32 )
40 FOR k=0 TO 50
50  IF EOF( #4 ) THEN EXIT k
60  INPUT #4,file$( k )
70 END FOR k : CLOSE #4
```

COPY_X

Syntax:

```
COPY_X sourcefile, destfile
```

Function:

Creates a job which copies a file in the background, i.e. while the Superbasic interpreter program continues execution. This is useful for spooling files to a printer whilst you continue with other activities.

COUNT()

See also:

SUBSTR(), SEARCH(), SEARCH_N()

Syntax:

```
COUNT( a$, b$ [,n] )
```

Function:

Returns the number of occurrences of *a$* found within *b$*. If the third argument is given, the search starts at the *nth* character in the second string.

CURS

See also:

ACTIVATE_Q, CLEAR_Q

Syntax:

```
CURS #n[,flag]
```

Function:

If *flag* is zero, the cursor of the given window is disabled. For all other values, the cursor will be enabled.

CURS_INC

Syntax:

```
CURS_INC [#n,] xinc, yinc
```

Function:

Sets the cursor increment. This makes it possible to display more text than usual in a window. If the spacing is less than the current character size and the window extends near the bottom or right hand edge of the screen, care has to be taken to prevent characters written in this area overwriting the QDOS variable table which is situated immediately after screen memory.

DAYS()

Syntax:

```
DAYS( year, month, day )
```

Function:

Returns the number of days since 31 December 1599, or -15 (a bad parameter error) if the date is invalid (for example, 29th February 1961). This is useful for calculating the number of days between two dates, or for validating a date input by the user.

DEC$()

See also:
BIN$(), HEX$()

Syntax:
```
DEC$( number [,digits] )
```

Function:

Returns a string containing all the digits of a floating point number (unlike a normal PRINT command which reveals up to seven decimal places). The result is rounded to the specified number of digits (9 max), or to the nearest integer if the second argument is omitted.

DISK

See also:
BOX, PAINT

Syntax:
```
DISK #n, x, y, radius [,eccentricity, angle]
```

Function:

Draws filled filled circles or ellipses. It always leaves the *fill* attribute reset.

DISPLAY

See also:
DMODE, DUAL_COPY, DUAL_SCR

Syntax:
```
DISPLAY [value, delay {,value, delay}]
```

Function:

This command is used to control the video display mode by altering the contents of an eight bit register. The *delay* determines the amount of time during the video refresh cycle in which each setting will be in effect. Using multiple pairs of parameters, the effect is to divide the screen up into horizontal bands with different characteristics. For example, it is possible to have the top half of the screen in low resolution (8 colour) mode, and at the same time, have the lower half of the screen in high resolution mode.

The *value* given will be written to the register for the associated delay (from the start of each screen refresh cycle). Only the following three bits are significant:-

Bit	Function
1	when set, disables the screen
3	determines the screen resolution (low-res when set)
7	when set, selects the alternative screen which starts at $28000

Note: Bit 0 is the least significant bit.

It is possible to divide the display into up to four distinct areas. It is also acceptable to omit the *delay* argument for the last pair, in which case the last area will extend to the bottom edge of the display. Below are a few examples:

```
    DISPLAY 0,2500,8
```

shows the top half (approximately) of the screen in high resolution, and the rest in low resolution

```
    DISPLAY 8,2200,128+8
```

shows the top third of the main screen (in low-res), followed by the QL's second screen in the lower two thirds of the display.

```
    DISPLAY 2,2500,128,600
```

blanks the top half of the screen, shows a portion of the second screen in the middle of the display (in high-res) then restores the original contents of the display status register, showing the bottom part of the main screen in the current resolution mode (as set by the latest MODE command).

DISPLAY with no arguments restores the normal display settings.

See chapter 6 for more details of how to use the QL's alternative screen.

NOTE: the QL serial port and microdrives have an effect on timing, and so the dual screen feature is always disabled when either is in use. During use of the dual screen feature, some problems may also be encountered with keyboard operation.

DMODE

See also:

DISPLAY

Syntax:

```
value = DMODE
```

Function:

It is a function which returns the current value of the display control register. See DISPLAY for details of register contents.

DUAL_COPY

See also:

DISPLAY, DUAL_SCR

Syntax:

```
DUAL_COPY [#n]
DUAL_COPY #n[,flag]
```

Function:

This procedure copies part of the displayed screen memory to the area corresponding to the QL's second screen or vice versa. The value of *flag* determines the direction in which the copy will be performed. Note that this command will not work until DUAL_SCR has been used to reserve memory for the second screen.

If *#n* is omitted, the whole screen is copied. If *#n* is given, then only the area covered by the corresponding window will be copied. In fact, the second screen is not large enough to cover the entire QL display and so only the portion of the main screen that corresponds to the equivalent area of the second screen will actually be copied.

If *flag* is non-zero, the copying will be from the second screen to the main screen. If it is zero or omitted, the copy will be in the opposite direction.

Various other manipulations can be performed using the MCOPY, MEXCHANGE, MFILL, EXCOPY and EXFILL commands which makes it relatively easy to produce animation and other special effects.

DUAL_SCR

See also:

DISPLAY

Syntax:

```
address = DUAL_SCR
```

Function:

This is a function which reserves memory for a second screen and returns its start address. If the dual screen has already been reserved, its address will be returned and no further action taken. Because the system variables start at address $28000, the dual screen may be never completely used, and its precise dimensions depend on how much common heap has already been allocated. Therefore, programs using this feature should use the DUAL_SCR command before doing anything else.

DUMP

Syntax:

```
DUMP #n, address[,bytes]
```

Function:

Prints the contents of a given portion of memory, both in hexadecimal and ASCII. If the *bytes* argument is omitted, the dump proceeds until a break (CTRL-SPACE) key is pressed. Note that break cannot be used within compiled programs.

ENTER

See also:

ACTIVATE_Q, CLEAR_Q

Syntax:

```
ENTER #n,a$
```

Function:

Enters a string of characters into the keyboard buffer related to channel *#n*, which must be a console channel. A "Not complete" error is issued if the buffer becomes full. (Note that if you don't specify a buffer size when opening a CON_ channel, the buffer will default to a capacity of 128 characters.)

EXCOPY

See also:

MCOPY, MEXCHANGE

Syntax:

```
EXCOPY addr1,addr2,bytes,incr1,incr2,times1[,incr3,incr4,times2][,pause]
```

Function:

This is a sophisticated variant of the MCOPY command, designed specifically for moving "rectangular" areas of memory. It is best used for copying areas of screen memory or uncompressed stored images (see HIDE()) for fast graphic effects. The following example program is equivalent to the form with 6 or 7 arguments:-

```
10 FOR i=1 TO times1
20  MCOPY addr1,addr2,bytes
30  addr1 = addr1 + incr1: addr2 = addr2 + incr2
40  SUSPEND -1,pause
50 END FOR i
```

and next program is equivalent to the form using 8 or 9 arguments:-

```
10 FOR i=1 TO times2
20  EXCOPY addr1,addr2,bytes,incr1,incr2,times1
30  addr1 = addr1 + incr3: addr2 = addr2 + incr4
40  SUSPEND -1,pause
50 END FOR i
```

In other words, the short form moves a rectangular area of memory, whereas the long one moves multiple rectangular areas of the same size. In both cases the pause argument defaults to zero.

EXECUTE, EXECUTE_W

Syntax:

```
EXECUTE filename[,command$][,priority[,dataspace]]
EXECUTE_W filename[,command$][,priority[,dataspace]]
```

Function:

The above are enhanced versions of the existing EXEC and EXEC_W commands. They allow both the job priority (normally 32) and the job's data space to be specified when the job is started.

The capability to change the default data space is particularly useful when trying to determine the most suitable size for compiled programs.

The most interesting feature is the ability to pass a string to the program as if it had been typed on the keyboard. This makes it possible, for example, to invoke QUILL on a given document:

```
EXECUTE_W mdv1_quill,STR$(240,'L',document$,10)
```

Note that the JOB_ID function (no arguments) will not work to read the job *id* of a job started with EXECUTE.

EXFILL

See also:

MEXCHANGE, MFILL

Syntax:

```
EXFILL addr,bytes,value,incr,times[,incr2,times2][,pause]
```

Function:

This is a variant of MFILL_W, and is mainly used to change the colour of the screen or a window smoothly. The form with only 5 or 6 arguments corresponds to the following Basic program:-

```
10 FOR i = 1 TO times
20  MFILL_W addr1, bytes, value
30  addr = addr + incr
40  SUSPEND -1, pause
50 END FOR i
```

The form with 7 or 8 arguments corresponds to:-

```
10 FOR i = 1 TO times2
20 EXFILL addr, bytes, value, incr1, times
30  addr = addr + incr2
40  SUSPEND -1, pause
50 END FOR i
```

The *pause* argument defaults to zero.

EXPAND()

See also:

EXPAND_L(), WSAVE_C

Syntax:

```
address1 = EXPAND( address2 )
```

Function:

If *address2* is the address of an un-compressed stored image, this function returns zero, otherwise it expands it and returns the address of the new image, or -3 for "Out of memory". The original image is not de-allocated.

EXPAND_L()

See also:

EXPAND(), WSAVE_C

Syntax:

```
address = EXPAND_L( filename )
```

Function:

This function is similar to ALCHP_L(): The file should contain an image saved using either WSAVE or WSAVE_C, and will be loaded into memory. If it was stored in compressed form, the image will be expanded automatically.

The routine returns the address of the expanded image or a negative QDOS error code. Once loaded, SHOWAT or ZOOM can be used to display the image without restrictions.

FACT()

Syntax:

```
value = FACT( n )
```

Function:

Returns the factorial of a number, i.e. 1*2*...*(n-1)*n. If *n* is greater then 300 an overflow occurs. It is very easy to produce fast versions of various useful functions using FACT() as in the following examples.

```
        DEFine FuNction PERMUTATIONS( n, k )
        RETurn FACT( n )/FACT( k )
        END DEFine
```

which returns the number of permutations of k objects taken out of n, and

```
    DEFine FuNction BINOMIAL( n, k )
    RETurn FACT(n)/(FACT( k )*FACT( n - k ))
    END DEFine
```

which returns the binomial coefficient of (n, k), i.e. the number of combinations of n objects taken k at a time.

FDEC$()

Syntax:

```
string$ = FDEC$( number, mask$ )
```

Function:

This is similar to a "PRINT USING" type command sometimes found in sophisticated implementations of Basic. The *number* is formatted according to the specified mask, which may contain the following special characters:-

Character	Function
#	If insignificant, the corresponding digit will be shown as a space.
0 (zero)	The corresponding digit will be shown even if not significant.
V	Indicates the virtual position of the decimal point. This may be omitted if it is the rightmost character in the mask.
S	Must be the first character of the mask, indicating that the number should be signed (whether positive or negative).
M	Must be the first character of the mask, indicating that a minus sign should be shown if the number is negative, but that no sign should be shown otherwise).

All other characters in the mask are printed as they are, or are replaced by blanks if they occur between insignificant digits that are printed as spaces. This makes it possible insert currency signs, thousands separators, and decimal points/commas for foreign countries which use different notations.

In all cases, the number is rounded to the number of decimal digits in the mask. If an overflows occurs, all the digits are replaced by asterisks. If the mask contains only '#'s, zero values will not be shown at all.

The number of decimal digits (i.e. those following the decimal point) specified in the mask for DEC$() or FDEC$() should never be greater than the number of digits required to represent the numeric argument, otherwise all digits in excess will be wrong. (These digits would be expected to be zero, but rounding errors in the binary to decimal conversion will prevent this.)

FETCH()

See also:

STORE

Syntax:

```
value = FETCH( x )
```

Function:

Reads a floating point number stored by the STORE command. *X* must be in the range 0-15.

FILE_DSPACE()

Syntax:

```
value = FILE_DSPACE( filename )
value = FILE_DSPACE( #n )
```

Function:

Returns the size of the data-space of an executable file. If the file is not executable, a value of -15 (bad parameter) will be returned.

FILE_LEN()

Syntax:

```
value = FILE_LEN( filename )
value = FILE_LEN( #n )
```

Function:

This function returns the length of a file. In the first case, return of a negative value indicates the QDOS error code encountered when trying to open the file. This makes it possible to test for errors such as non existent files, or files being used already before attempting to open them.

FLUSH

See also:

KEY

Syntax:

```
FLUSH #n
```

Function:

Forces all buffers relating to the file associated with *#n* to be written to the medium. (Ensures that data held in slave blocks has been written to the medium, which can not normally be guaranteed until the file is closed.)

FN_KEY

See also:
KEY

Syntax:

```
value = FN_KEY
```

Function:

This is a function which returns a value corresponding to the function key currently pressed, or zero if none is being pressed. Shifted function keys return values in the range 6-10, whereas CTRL'ed and ALT'ed function keys return values in the range 11-15 and 16-20 respectively.

FONT_USE

See also:
ATTR, SET_FONT

Syntax:

```
FONT_USE #n [,font1 [,font2]]
```

Function:

This function is used to select character fonts. If one or both arguments are zero or are omitted, the default ROM fonts are selected.

font1, font2
Should contain the addresses of the fonts to be used in the given window. A zero value will cause the default QL fonts to be selected. Small negative values refer to fonts linked into the *Mega-Toolbox* using SET_FONT. Use FONT_USE #n on its own to select the default fonts for a given window.

FPOS()

See also:
SPOS

Syntax:

```
value = FPOS( #n )
```

Function:

Returns the value of the file position pointer of the file open on the given channel.

FPRINT_X

See also:
  PRINT_X

Syntax:

```
FPRINT_X #n, filename [,times]
```

Function:

This command is similar to PRINT_X, except that the controlling characters are collected from the given file.

FREE_CHAN

Syntax:

```
value = FREE_CHAN
```

Function:

This function returns the number of the first free Superbasic channel, i.e. the closed channel with the lowest channel number. It is useful for saving memory or for writing procedures which use temporary channels.

FREE_MEM

Syntax:

```
value = FREE_MEM
```

Function:

This function returns the amount of free memory. The *value* returned corresponds to the amount of RAM available for filing system slave blocks, less the space used by one block.

FREE_SECT()

Syntax:

```
value = FREE_SECT( devicename )
```

Function:

This function returns the number of free sectors on a given medium, or a small negative number indicating the QDOS error encountered when attempting to open the directory. It is useful for avoiding "Drive full" errors or before issuing a DIR or XDIR command.

GET(), GET%(), GET$()

See also:
  PUT()

Syntax:

```
value = GET( #n )
value = GET( a$ )
value = GET%( #n )
value = GET%( a$ )
string$ = GET$( #n )
```

```
string$ = GET$( a$ )
```

Function:

These functions allow values stored in QDOS internal format to be read into Superbasic variables. GET(), GET%() and GET$() read (and return) integer, real and string values respectively.

In all cases the source may be a channel (i.e. a file) or a string variable. In the latter case the string must be of exactly the correct length determined by the type of value being read. That is 6 characters for GET(), 2 characters for GET%(), and 2+n for GET$() (where n is the length of the resulting string, and is read from the first two bytes in the argument).

NOTE: The first nibble (4 bits) of a real number in QDOS format must be zero, or a "Bad parameter" error will occur.

GETMEM$()

See also:

QPEEK$, PUTMEM, QPOKE

Syntax:

```
string$ = GETMEM$( address, bytes )
```

Function:

This function reads a given area of memory into a string. It may be use together with GET() to read floating point numbers stored in memory in QDOS format.

HEX$()

See also:

BIN$(), DEC$(), HEX()

Syntax:

```
string$ = HEX$( number [,digits] )
```

Function:

This function returns a hexadecimal string representing the value of *number*. *Digits* is the number of digits in the result, and is assumed to be two if omitted.

For example, to find the hexadecimal representation of one hundred:-

```
PRINT HEX$(100)
```

HEX()

See also:

BIN(), DEC$(), HEX$()

Syntax:

```
value = HEX( string$ )
```

Function:

This function returns the decimal representation of a hexadecimal string, or zero if the argument contains invalid characters. The case of characters in *string$* is not significant.

For example, to find what $64 (hexadecimal) is in decimal:-

```
PRINT HEX("64")
```

HIDE(), HIDE_C()

See also:

MEM_LENGTH(), SHOW, SHOWAT, RECHP, ZOOM

Syntax:

```
address = HIDE( #n )
address = HIDE( #n, address )
address = HIDE_C( #n )
```

Function:

These functions save the contents of a screen window in the common heap and return the address of the allocated area. The second form of HIDE() (which has two parameters) allows you to specify the area where the image is to be stored and is faster because the allocation step is not necessary. When an *address* parameter is given, it should always be a value originally returned from a previous HIDE(). Hence you should use HIDE(#n) the first time you save a window, and then subsequently use HIDE(#n, address) afterwards.

HIDE_C() is similar to HIDE() except that it will save the image in a compressed form if this will result in a memory saving. This is slower than using HIDE(), and prevents the use of ZOOM and SHOWAT commands.

INPUT$()

Syntax:

```
string$ = INPUT$( #n [,timeout[,chars]] )
```

Function:

This function returns a string containing a given number of characters from a channel. If only a channel number is given, it behaves like a normal INPUT, i.e. it reads until an ASCII 10 is encountered.

If a channel number and *timeout* (in fiftieths of a second) are specified, the function may return before the user has pressed <ENTER> in which only the characters typed so far will be returned (and no ASCII 10 will be present at the end of the string).

If all three parameters are given, then the function will return prematurely (i.e. before the user has pressed <ENTER>) if either the *timeout* expires or the specified number of characters have been typed. In this form, characters typed are not echoed to the screen as they are typed.

INPUT_X()

See also:

ANSWER$()

Syntax:

```
id = INPUT_X( #n )
```

Function:

INPUT_X() and ANSWER$() are used to obtain input from the user whilst a basic program continues to execute.

INPUT_X() is given a channel number relating to a console channel that must not be used by anything else. It sets up a job to read input from that channel and returns the *id* of the job so that it may be tested to detect when the user has finished typing. The basic program should periodically call ANSWER$() with the relevant job *id* and then use JOB_STAT to see if the job has been removed. If the job has been removed, the string returned by ANSWER$() is the string that was input by the user. If the job still exists, the user has not yet finished typing his input, and the string returned by ANSWER$() will correspond to the current state of the line being typed. In the mean time, the program can continue to perform other functions.

Several inputting jobs can be set up simultaneously allowing the user to select between them using CTRL-C. The input channel used by each job must be different in each case and must not be used by anything else - otherwise the program will crash.

Example:

```
.
.
.
1000 reply$ = ""
1010 OPEN #ix,"con_"
1020 ixid = INPUT_X( #ix )
1030 REPeat get_input
1040   IF JOB_STAT( ixid ) = -2 THEN EXIT get_input
1050   reply$ = ANSWER$( ixid )
1060   do_other_things
1070 END REPeat get_input
.
.
.
```

INVERSE

Syntax:

    INVERSE [#n]

Function:

This function exchanges the ink and strip colours for the given window providing a quick way of highlighting text. The paper colour is not modified.

JOB$()

Syntax:

    string$ = JOB$(jobid)

Function:

This function returns the name of the job (if any) whose *jobid* was given, or a null string if such a job does not exist.

JOBS

Syntax:

```
JOBS [#n]
```

Function:

This command prints information on all current jobs (apart from the Superbasic interpreter). This includes the job name name (if any), job id, address, length, priority, and a flag which indicates if the job is suspended ('S') or waiting for another job to complete ('W').

JOB_ADDR()

Syntax:

```
address = JOB_ADDR( jobid )
```

Function:

This function returns the header address of the the given job, or -2 to indicate an invalid job id. For standard jobs (such as those started by the *Mega-Toolbox*) the job code starts $68 bytes after such an address, although this not always so, as in the case of cloned jobs for example.

The job header contains various information that can be accessed, including the length of the job, job priority, owner job, register contents, etc.

JOB_ID, JOB_ID()

Syntax:

```
value = JOB_ID [(jobname$)]
```

Function:

Returns the id of a given job, or -2 (invalid job) if the job does not exist. If used without any arguments it returns the id of the most recently created job, or -2 if this job has already been removed from the system. As a special case, JOB_ID('') returns the id of the current job: this enables a program to determine whether it is being interpreted (a zero value) or if it is a compiled task (any other value).

Note that JOB_ID with no arguments will always return -2 if called after a job created by an EXECUTE command.

JOB_STAT()

Syntax:

```
value = JOB_STAT( job_id )
```

Function:

This function returns an integer value indicating the status of the job according to the following table.

Value	Meaning
0	The job is active.
>0	The job has been suspended and the value returned is the number of fiftieths of a second to go before it will be released.
-1	The job has been suspended for an indefinite timeout
-2	The job has been removed from the system.
-3	The job is waiting for another job to complete.

KEY

See also:

KEYS

Syntax:

```
KEY n,string$
```

Function:

Assign a string or machine code routine to the *n*-th function key; shifted keys are referenced with *n* from 6-10. CTRL'ed keys correspond to the values 11-15 and ALT'ed keys to 16-20. Values outside the range 1-20 will produce an "Out of range" error.

KEYS can be used to display the current settings.

The maximum number of keys that can be programmed and the number of characters allowed in the programmed string is determined when the *Mega-Toolbox* is first loaded (see section 6.1)

If an ordinary string is assigned to a function key, then it will be put into the keyboard buffer of the currently active channel each time the corresponding function key is pressed.

To assign a machine code routine to a function key, the *string$* must have the special format described below. Imagine the string to be an array of characters where *string$(0)* is its first character, *string$(1)* its second and so on:-

string$(0)	should be CHR$(0)
string$(1)	should be CHR$(0)
string$(2)	
string$(3)	are four "bytes" that represent
string$(4)	the address of the routine.
string$(5)	
string$(6)	
string$(7)	represent a value to be
string$(8)	assigned to register D1
string$(9)	
string$(10)	
string$(11)	represent a value to be
string$(12)	assigned to register D2
string$(13)	

The following example shows how you would construct such a string and use it with the KEY command:-

```
100 POKE_W BUFF_ADDR,0           : REMark Two starting zeros
110 POKE_L BUFF_ADDR+2,address : REMark Address of machine code routine
120 POKE_L BUFF_ADDR+6,-1        : REMark A value for D1
130 POKE_L BUFF_ADDR+10,169    : REMark A value for D2
140 KEY 1,GETMEM$(BUFF_ADDR, 14): REMark Assign to F1 (14=string length)
```

Note that a machine code routine called by pressing a function key will always be invoked as part of the QDOS scheduler loop. This means that during execution of your routine, multi-tasking will be suspended and you will not be able to use features such as QDOS input or printing functions, since they are also invoked as part of the scheduler loop.

KEYBOARD

Syntax:

    KEYBOARD capslock [,startdelay [,delay [,queuecode]]]

Function:

Sets the following parameters of the keyboard handling routine:

capslock
: if non-zero, will enable upper case, otherwise lower case will be enabled.

startdelay
: specifies the number of frames (50ths of a second) before a key will start to autorepeat.

delay
: sets the number of frames between each automatic key repetition.

queuecode
: selects the code of the key used to cycle among active windows (normally this is is 3, which corresponds to CTRL-C)

All but the first argument are optional. A negative value for any parameter will leave its state unchanged. For example:

    KEYBOARD -1, -1, 2

alters the delay between key repetitions without affecting anything else.

KEYS

See also:
    KEY

Syntax:

    KEYS [#n]

Function:

Prints the current assignments of the function keys defined by KEY. Carriage returns (ASCII 10) are shown as backward arrows.

KEY_USE

Syntax:

KEY_USE mode

Function:

This function enables you to selectively disable the keyboard input enhancements described in section 4.1.

The first five bits of *mode* enable or disable each one of the three new keyboard functions and the QL's second screen and screen freeze features:

Bit	Function enabled/disabled
0	Function key programming.
1	Recalling the command line using the SHIFT-ENTER combination.
2	Command line editing features related to the ALT and SHIFT keys.
3	The QL's second screen.
4	The ability to pause the QL screen using SHIFT-F5 (is disabled if the bit is set)

NOTE: Unless stated otherwise, setting a bit enables the corresponding feature and resetting it disables the feature.

For example, use 16 (binary 10000) to disable all the above features, 17 (binary 10001) to enable function key programming only, 15 (binary 01111) to enable all of them, and so on.

As a special case, any negative value will not only disable everything, but will also deactivate the interrupt routine (which might be useful when a program is malfunctioning.)

The *Mega-Toolbox* is started with a default value of 15, which activates all features.

LOWER$()

See also:

UPPER$()

Syntax:

string2$ = LOWER$(string1$)

Function:

Converts a string to lower case.

MAX()

See also:

CHOOSE$(), MIN()

Syntax:

value = MAX(x, y {,z})

Function:

Returns the maximum value in a list of arguments. Two or more arguments are allowed.

MCOPY

See also:

EXCOPY, MEXCHANGE

Syntax:

```
MCOPY sourceaddr, destaddr, bytes
```

Function:

Copies memory from *sourceaddr* to *destaddr*. This operation is performed correctly even if the two areas partially overlap. Memory is transferred in words or long words if possible to maximise speed.

MEM_LENGTH()

See also:

HIDE(), HIDE_C()

Syntax:

```
value = MEM_LENGTH( address )
```

Function:

Returns the length (in bytes) of an image stored by a HIDE() or HIDE_C() instruction. This is useful when you want to copy from from one area to another one, or to save an image on disk.

MEXCHANGE

See also:

EXCOPY, MCOPY

Syntax:

```
MEXCHANGE addr1, addr2, bytes
```

Function:

Exchanges the contents of two areas of memory (which may overlap). Copying is optimised for speed as with MCOPY.

MFILL, MFILL_W, MFILL_L

See also:

EXFILL

Syntax:

```
MFILL address, bytes, value
MFILL_W address, bytes, value
MFILL_L address, bytes, value
```

Function:

Fills an area of RAM of length *bytes* bytes starting at *address* with the given *value*. MFILL writes byte values, MFILL_W word (16 bit) values and MFILL_L long word (32 bit) values.

For example, to clear the whole display area without altering the current cursor positions, you could use:

```
    MFILL_L 131072, 32768, 0
```

MIN()

See also:

CHOOSE$(), MAX()

Syntax:

```
    MIN( x, y {,z} )
```

Function:

Returns the minimum value in a list of arguments. Two or more arguments are allowed.

MPRINT

See also:

PRINT_3D, PRINT_X

Syntax:

```
    MPRINT #n, message$, xdim, ydim
```

Function:

This function enables you to print messages in **any** character size. The *xdim* and *ydim* values are scaling factors which will be applied to the x and y dimensions of the printed characters. The scaling factors can take any value greater than or equal to 1 which means that characters of any width or height can be produced.

This command can be used to produce giant 3D messages as in the following example.

```
    DEFine PROCedure MPRINT_3D( ch, mess$, xdim, ydim, colour, width)
      LOCal xtemp, ytemp, inktemp, overtemp, ad
      ad = CHAN_ADDR(#ch): xtemp = XCUR(#ch): ytemp = YCUR(#ch)
      inktemp = PEEK(ad + 70): overtemp = PEEK(ad + 66)
      CURSOR #ch, xtemp - width, ytemp - width: OVER 1: INK #ch, colour
      MPRINT #ch, mess$, xdim, ydim
      CURSOR #ch, xtemp, ytemp: INK #ch, inktemp
      MPRINT mess$, xdim, ydim: POKE ad + 66, overtemp
    END DEFine MPRINT_3D
```

Beware: this routine may crash the system if executed under the interpreter, because of a Superbasic bug which limits the total number of arguments and local variables in a procedure or function to eight. If the program is to be interpreted you should change the second line to

```
    LOC temp( 5 )
```

and change the rest of the routine by replacing 'xtemp' with temp(1), 'ytemp' with temp(2), and so on.

MRECOL

See also:

RECOL_X, WINDOW_OP

Syntax:

```
MRECOL colour$, address
```

Function:

Changes the colour of a stored image. Unlike the Superbasic RECOL instruction, colours are specified in a 8-character string. For example "01432657" will exchange red with green, and cyan with yellow (and vice versa). It works with both normal and compressed images, but in the latter case only if the colour$ argument does not contain any duplicate characters (e.g. the string "10123576" will be rejected if the stored image is compressed).

Note that this command uses a faster algorithm than the QL's built in RECOL function.

MSEARCH(), MSEARCH_W()

Syntax:

```
address = MSEARCH( string$, address, bytes )
address = MSEARCH_W( string$, address, bytes)
```

Function:

MSEARCH() looks for a given string in the specified area of memory, and returns the address of the first match, or zero if none is found. MSEARCH_W() is identical, except that it assumes that the string is word aligned - as with all QDOS format strings. This is faster and avoids false matches.

Searches are case sensitive.

Using the PUT$() function, it is possible to generate strings corresponding to integer and floating point numbers, enabling these to be searched for using MSEARCH_W().

MULTITASK_X

See also:

EXECUTE, EXECUTE_W

Syntax:

```
MULTITASK_X priority, times, timeout, address {,register_value}
```

Function:

MULTITASK_X creates a job which will repeatedly execute a user-defined machine code routine.

priority
: should be given (as a number from 1 to 127), which will determine the effective speed of execution.

times
: is the number of repetitions (-1 means "forever") before the job will be terminated.

timeout
: is the number of frames between one execution and the next.

address

is the entry point of the machine code routine to be called.

All further argument values, if any, will be loaded into the 68008 registers each time the routine is invoked, i.e. at the beginning of each repetition, from D1 onward. All other registers will be left unaltered, and will retain their values from the previous repetition. At the start of each repetition D0.W will be loaded with the number of repetitions so far (starting with zero).

On exit from the user routine i.e. at the final RTS instruction, D0.L should be set to zero, a positive value, or a QDOS error code. In the latter case, (D0.L contains a negative value) the job will be removed. If D0.L contains a positive value, the job will be suspended for the given number of frame periods instead of the default number specified by the *timeout* parameter.

All registers may be used and modified, although the stack pointer must of course be restored to its original value before executing the final RTS. On entry A6 points to a 128-byte buffer which may be used for the intermediate results of QDOS traps and for other local storage.

OPTION_USE

See also:

QCALL, QREG, QTRAP, SWAP, VAR_ADDR

Syntax:

OPTION_USE value

Function:

This command determines how values are to be returned through arguments. Only three *Mega-Toolbox* commands may return values through their arguments: SWAP, QCALL and QTRAP.

Normally (by default) if the program is being interpreted, these routines will return values through their arguments. In a compiled program, the SWAP command will causes a "Not implemented" error, whereas QCALL and QTRAP will executed normally, but require that returned values be read using the QREG() function. To enable this, the defaults must first be modified using the OPTION_USE command.

If *value* < 0 the *Toolbox* acts as if the calling program is compiled.

If *value* > 0 the *Toolbox* acts as if the calling program is interpreted (i.e. "Not implemented" errors never occur).

If *value* = 0 the default behaviour is selected.

PAINT

See also:

BOX, DISK

Syntax:

PAINT [#n,]x,y

Function:

Fills an irregular shaped area given the graphic coordinates of an inside pixel using the current ink colour for the window.

If the current ink is a "stipple" (see QL User Guide), then the pixel specified should not be the same colour as either of the two colours making up the stipple.

PANW, SCROLLW

See also:

PAN_X, SCROLL_X, PANW_X, SCROLLW_X

Syntax:

```
PANW [#n,] distance
SCROLLW [#n,] distance
```

Function:

These are "wrap-around" versions of the Superbasic PAN and SCROLL commands. Pixels leaving one edge of the screen reappear on the opposite edge. The movement is always smooth.

PANW_X, SCROLLW_X

See also:

PAN_X, SCROLL_X, PANW, SCROLLW

Syntax:

```
PANW_X [#n,] distance [times [,timeout]]
SCROLLW_X [#n,] distance [times [,timeout]]
```

Function:

These commands create a job which repeatedly pans or scrolls a window while the Superbasic program continues its execution. Pixels leaving one edge of the window re-appear on the opposite edge as in the PANW and SCROLLW commands.

distance

is the amount of the pan or scroll.

times

is the number of repetitions (the default value is one, and -1 means 'forever').

timeout

is the delay between consecutive repetitions, expressed in 50th's of a second (default is zero - no delay).

PAN_X, SCROLL_X

See also:

PANW, SCROLLW, PANW_X, SCROLLW_X

Syntax:

```
PAN_X #n,distance [times [,timeout]]
SCROLL_X #n,distance [times [,timeout]]
```

Function:

These commands are equivalent to the PANW_X and SCROLLW_X commands, except that no wrap-around occurs.

PIPE_ID()

See also:

CONNECT, CONNECT_C

Syntax:

```
channel_id = PIPE_ID( #n )
channel_id = PIPE_ID( link_id )
```

Function:

Superbasic allows pipes to be opened, but currently provides no mechanism for connecting them together. A pipe acts like a first in first out (FIFO) buffer enabling data to be put in at one end and taken out at the other.

The following example opens two output pipes, one capable of holding up to 9000 characters and the other up to 1000 characters. (Note that both OPEN and OPEN_IN create **output** pipes.)

```
OPEN #out,pipe_9000
OPEN_IN #in,pipe_1000
```

Thus when you open an output pipe, you can put data into it using the channel opened to it (*#out* in the above example) but you have no channel with which to read data from the other end of the pipe.

PIPE_ID() will open an input channel for a given output pipe. The channel id returned is a QDOS channel id and **not** a Superbasic channel number. In order to access the new channel from Superbasic you must connect it to a Superbasic channel using the CONNECT command.

Link_id is the channel id of the output pipe, and so PIPE_ID(#n) and PIPE_ID(CHAN_ID(#n)) are equivalent. The latter form is useful in a multi-tasking environment, when one job sends data to another job via a pipe, perhaps having sent the pipe id using the STORE command.

In all cases, a negative returned value indicates that an error has occurred, such as no pipe found corresponding to the given *#n* or *link_id*.

PLAY_X

See also:

ALARM_X

Syntax:

```
PLAY_X times, melody$
```

Function:

Plays a melody while the Superbasic program continues executing.

times

is the number of repetitions (-1 means 'forever').

melody$

is a string containing one pair of characters for each note in the tune. The first character is the duration, the second one is the pitch of a note. A pitch of 255 produces no sound and can be used to introduce pauses.

If this job is removed using a REMOVE command, a BEEP instruction should follow to turn the sound off.

PREFETCH$()

See also:

PROMPT$(), INPUT_X()

Syntax:

```
character$ = PREFETCH$( #n )
```

Function:

Similar to INKEY$ but does not actually read the character. That is, the character pressed will be returned as a single character string, but will still be present in the input buffer of channel *#n*.

PRINT_3D

See also:

PRINT_X

Syntax:

```
PRINT_3D #n, string$, ink, width
```

Function:

Prints 3D strings. *String$* is the text string to be printed, *ink* is the secondary ink colour which will be used for the sides of the characters and *depth* determines the depth of the three dimensional image (0 to 8). The current ink colour for the given window will form the "top" or closest "side" of the text.

PRINT_X

See also:

FPRINT_X

Syntax:

```
PRINT_X [#n,] string$ [,times]
```

Function:

Creates an independent job which repeatedly prints a message on a given channel. *Times* is the number of repetitions (-1 means 'forever'), before the job finishes, and has a default value of one.

A pair of characters, always starting with a backslash '\', is used to perform special functions. The character immediately following the backslash selects which function is to be performed (for example to change the ink colour) and is immediately followed by zero or more parameters. Where more than one parameter is required, they should be separated by spaces or commas. For example, to change a border to a width of 15 with colour 243 the sequence "\b15,243" or "\b15 243" should be included in the *string$* parameter. The following table gives full details of the special functions and their parameters.

Control	Parameter 1	Parameter 2	
\\	Prints a backslash.	-	-
\i	Changes ink colour.	Colour	-
\p	Changes paper colour.	Colour	-
\s	Changes strip colour.	Colour	-
\b	Changes border width and colour.	Width	Colour
\c	Enables or disables cursor.	0 or 1	-
\f	Enables or disables flashing.	0 or 1	-
\u	Enables or disables underlining.	0 or 1	-
\o	Controls the OVER attribute.	0 or 1	-
\n	Prints a newline.	-	-
\t	Tabulate (move) to a given column.	Column	-
\C	Executes a CLS.	0, 1, 2, 3 or 4	-
\A	Sets cursor position, as AT in Superbasic.	Row	Column
\a	Sets cursor position, as CURSOR in Superbasic.	Column	Line
\S	Scrolls the window.	Distance	-
\P	Pans the window.	Distance	-
\R	Re-colour a window (this is followed by a string of eight characters, as in the MRECOL instruction, e.g. "R23456701").	String	-
\H	Changes the size of characters (as CSIZE).	xsize	ysize
\F	Selects new fonts of characters (parameters as for font_use).	Font1	Font2
\W	Waits for a given timeout or, if the timeout is -1, until the TAB key is pressed.	Timeout	-
\D	Sets a delay between each character, in order to slow down the job and give a 'typewriter' feeling; a delay of zero restores normal printing.	Delay	-
\R	Invokes rolling messages, i.e. all the text that follows (until the end or a newline) is shown on the same line, which rolls to the left to make room for new characters. The argument sets the speed; use a delay of zero to return to normal printing mode.	Delay	-

Parameters are written using ASCII digits, and if the text that follows starts with a decimal digit another backslash should be used to end the parameter.

PROMPT$()

See also:

PREFETCH$(), INPUT_X()

Syntax:

```
PROMPT$( #n, string$ [,length] )
```

Function:

This function allows you to write input routines similar to those in the four PSION programs which offer a default reply that can be edited if required.

string$

The *string$* is output on channel *#n* and the user may accept such a prompt (by pressing the ENTER key), edit it (by pressing an editing key) or overwrite it (by pressing any other key).

length

is the maximum length allowed for the answer (the default is 128 which is also the maximum value allowed for *length*).

PUT

See also:

PUT$(), GET()

Syntax:

```
PUT [#n,] arg {,arg}
```

Function:

Writes data to a channel in QDOS format, i.e. 2 bytes for integers, 6 bytes for floating point numbers and 2 bytes followed by the characters for strings. If the argument is a variable its type may be deduced by the last character of its name (i.e. $ for strings, % for integers, real numbers otherwise). If the argument is an expression, it is possible to force it to become a given type by appending one of the following sequences of characters. In the examples, "EXPR" represents the expression you wish to force to a given type:-

EXPR||0	forces integer type
EXPR+0	forces real type
EXPR&""	forces string type

PUT$()

see also:

MSEARCH_W(), PUT, GET$()

Syntax:

```
PUT$( arg )
```

Function:

Returns a string containing the bytes representing the argument in QDOS format. That is, a string of 2 characters for integers, 6 characters for floating point numbers, and 2+n characters for strings.

The result can be sent to a file, or stored in a string variable and read afterwards using GET(), GET$() or GET%().

See the PUT command for details of how to force conversion of a particular type.

The PUT$() function may also be useful if used in an MSEARCH_W() command, find the location of a variable containing a given value as in the following example

```
MSEARCH_W( PUT$(PI), 0, 48*1024 )
```

which returns the address of the floating point representation of π stored in the QL ROM (which is 48K long).

PUTMEM

See also:

 GETMEM$(), QPOKE, QPEEK$()

Syntax:

```
PUTMEM address, string$
```

Function:

Pokes a string into a given address in memory. The *address* may be an odd number.

QCALL

See also:

 QREG(), QTRAP

Syntax:

```
QCALL address[,d1[,d2[,d3[,d4[,d5[,d6[,d7[,a1[,a2[,a3[,a4]]]]]]]]]]]
```

Function:

This function is similar to the Superbasic CALL instruction but has no bugs and returns the values of the 68008 registers through the routine's arguments.

All register arguments are optional and on exit from the routine, assignments are performed only on those arguments which were specified as floating or integer variables. Returned values may also be read using the QREG() function.

QPEEK$()

See also:

 QPOKE, QTRAP

Syntax:

```
string$ = QPEEK$( address )
```

Function:

Peeks memory for a string stored in QDOS format (i.e. as a word - length - followed by the bytes of the string). Because the string starts with a word, *address* must be an even number.

This can be used to read values from machine code routines or QDOS traps (using QTRAP).

QPOKE

See also:

QPEEK$(), QTRAP

Syntax:

```
QPOKE address, string$
```

Function:

Pokes a string into memory in QDOS format, i.e. a word for its length followed by the characters. This is the converse of QPEEK$() and can be used to communicate basic variable values to machine code routines and QDOS traps (using QTRAP).

QREG()

See also:

OPTION_USE, QCALL, QTRAP

Syntax:

```
value = QREG( renumber )
```

Function:

Returns the contents of a 68008 register as passed back by the latest QCALL or QTRAP command.

In compiled basic programs it is the only way to read such values. Use a *regnumber* value in the range 0-7 to access registers D0-D7, or in the range 8-12 for registers A0-A4. This function must be executed immediately after a QCALL or QTRAP, otherwise it is likely to return meaningless values.

QTRAP

See also:

QCALL, QPEEK$(), QPOKE, QREG()

Syntax:

```
QTRAP trapnumber, d0[,d1[,d2[,d3[,a0[,a1[,a2[,a3]]]]]]]
```

Function:

Executes a QDOS trap, passing values to/from 68008 registers, similar to the QCALL command. Errors are signaled by a non-zero value returned in the D0 argument.

trapnumber

may be equal to 1,2,3,-2 and -3. The negative values mean that trap #2 or #3 will be preceded by a trap #4 (which makes all addresses relative to the A6 register, for use with the Superbasic interpreter).

All register arguments (apart from D0) are optional. Register values are passed back to the Superbasic program as in QCALL, and may be also read using the QREG() function.

RECHP

See also:

ALCHP()

Syntax:

    RECHP address {,address}

Function:

Releases memory reserved in the common heap. To prevent crashes, the address argument(s) are checked and a "Bad parameter" error occurs if one of them is outside the common heap.

RECOLSHOW_X

See also:

HIDE(), RECOL_X, SHOW, SHOWAT, WLOAD, WSAVE, ZOOM

Syntax:

    RECOLSHOW_X [#n,]colour$, address [,times [timeout]]

Function:

This command creates a job which re-colours a stored image and shows it at regular intervals.

colour$

is an 8 character string as for MRECOL.

address

is the location of the stored image.

times and *timeout*

See PANW_X().

This command makes it easy to create simple but effective colour animations. If the image is compressed the colour string cannot contain duplicate characters, or a "Not implemented" error will occur.

RECOL_X

See also:

HIDE(), RECOLSHOW_X, SHOW, SHOWAT, WLOAD, WSAVE, ZOOM

Syntax:

    RECOL_X #n,colour$ [,times [,timeout]]

Function:

Performs a multi-tasking RECOL of the selected window.

colour$

is an 8 character string indicating the colour changes to be made.

For example

    RECOL_X '02345671',-1

will cause all colours but black to cycle among themselves (in low resolution mode). Note that this command makes use of the Superbasic RECOL command in the QL's ROM, which takes the control of the system and freezes all other jobs while executing. To avoid this,use RECOLSHOW_X instead.

times and *timeout*

See PANW_X().

RELEASE

See also:

JOBS, REMOVE, SETPRIOR, SUSPEND

Syntax:

```
RELEASE jobid
```

Function:

Releases a suspended job.

There are two special values for *jobid:* -1 means "this job", -3 (or lower) means "all jobs but this one".

REMOVE

See also:

JOBS, RELEASE, SETPRIOR, SUSPEND

Syntax:

```
REMOVE jobid [,error]
```

Function:

Forces removal of a job. If another job is waiting for it to complete, the specified error code will be returned to that job. The default value for error is zero.

There are two special values for *jobid:* -1 means "this job", -3 (or lower) means "all jobs but this one".

No error occurs and no action is undertaken if *jobid* refers to a job which does not exist. Job zero (the Superbasic interpreter) cannot be removed.

RESET_SCR

See also:

DUAL_SCR

Syntax:

```
RESET_SCR
```

Function:

This procedure de-allocates the memory used by the alternative screen.

RESPR_L()

See also:

ALCHP_L()

Syntax:

```
value = RESPR_L( filename )
```

Function:

This command is similar to ALCHP_L(), but allocates memory in the resident procedure area instead of the common heap. It is useful for loading in and linking new extensions to the interpreter, which may be done in just one instruction, for example.

```
    CALL RESPR_L(mdv1_extension_bin)
```

Note that unlike ALCHP_L(), QDOS errors do not stop execution.

REVERSE$()

Syntax:

```
    string2$ = REVERSE$( string1$ )
```

Function:

Returns *string1$* in reverse order, e.g. REVERSE$('ABCDE') returns 'EDCBA'. This can be used together with SUBSTR() and SEARCH for backward searches, or with TRIM$() to truncate leading blank, as in the following.

```
        DEFine FuNction TRUNCATE_LEADING_SPACE$( a$ )
           RETurn REVERSE$(TRIM$( REVERSE$(a$) ))
        END DEFine
```

Another example:

```
        DEFine FuNction LAST_MATCH( a$, b$ )
           LOCAL k
           k = SUBSTR( REVERSE$( a$ ), REVERSE$( b$ ) )
           IF k THEN k=LEN( b$ ) - LEN( a$ ) - k + 2
           RETurn k
        END DEFine
```

RIGHT$()

Syntax:

```
    string$ = RIGHT$( a$ [,n] )
```

Function:

This is similar to the Microsoft basic function with the same name, which returns the *n* rightmost characters of the string *a$*, or its last character if the second argument is omitted. Its Superbasic counterpart would be "a$(LEN(a$)-n+1 TO)", which is long-winded and takes longer to evaluate.

ROL(), ROL_W(), ROL_L()

See also:

    BCLR(), BSET(), BTST()

Syntax:

```
    value = ROL( bit, number )
    value = ROL_W( bit, number )
```

```
value = ROL_L( bit, number )
```

Function:

These functions are similar to the 68008 assembly instruction "ROL". The returned value is *number* rotated left by *bit* bits or to the right if *bit* is negative. In both cases *bit* is taken modulo 32.

ROWS()

See also:

COLUMNS()

Syntax:

```
value = ROWS( #n )
```

Function:

Returns the height of the given window in character units (which depends on the current character size set using CSIZE).

PANW, SCROLLW

See also:

PANW_X, SCROLLW_X

Syntax:

```
PANW #n, distance
SCROLLW #n, distance
```

Function:

These commands are "wrap-around" versions of the existing PAN and SCROLL commands. That is, pixels leaving one edge of the screen reappear at the opposite edge.

PANW_X, SCROLLW_X

See also:

PANW, SCROLLW

Syntax:

```
PANW_X [#n,]distance [times [,timeout]]
SCROLLW_X [#n,]distance [times [,timeout]]
```

Function:

These commands create a job which repeatedly pans or scrolls a window while the Superbasic program continues its execution. Pixels leaving one edge of the window reappear at the opposite edge - i.e. they wrap-around.

distance

is the amount of the pan or scroll.

times

is the number or repetitions. The default value is one, and -1 means "forever".

timeout

is the delay between repetitions expressed in 50th's of a second. Its default value is zero (no delay).

PAN_X, SCROLL_X

Syntax:

```
PAN_X #n,distance [times [,timeout]]
SCROLL_X #n,distance [times [,timeout]]
```

Function:

These are non wrap-around versions of PANW_X and SCROLLW_X.

SCR_ADDR()

Syntax:

```
address = SCR_ADDR( x, y )
address = SCR_ADDR( #n, x, y )
```

Function:

Both functions return the memory address of a pixel. See COLOUR() for more details of syntax.

SEARCH()

See also:

COUNT(), SEARCH_N(), SUBSTR()

Syntax:

```
value = SEARCH( a$, b$ [,n] )
```

Function:

This is a variant of the SUBSTR() command. The string *b$* is searched for **any** character contained in *a$*. The position of the first character in *b$* that matches any character in *a$* is returned, or zero if none of the characters in the first string appears in the second. The search is case sensitive, but this can be circumvented by using the UPPER$() or LOWER$() functions. For example

```
PRINT SEARCH( 'AEIOU', UPPER$(phrase$) )
```

SEARCH_N()

See also:

COUNT(), SEARCH(), SUBSTR()

Syntax:

```
value = SEARCH_N( a$, b$ [,n] )
```

Function:

This function is the same as SEARCH(), except that it returns the position of the first character in *b$* which does **not** appear in *a$*. The search is case sensitive.

SETPRIOR

See also:

JOBS, RELEASE, REMOVE, SUSPEND

Syntax:

```
SETPRIOR jobid, priority
```

Function:

This command changes the priority of a job. A priority of zero deactivates the job.

There are two special values for *jobid:* -1 means "this job", -3 (or lower) means "all jobs but this one".

SET_FONT

See also:

ATTR, FONT_USE

Syntax:

```
SET_FONT fontnum, address
```

Function:

Links a font into the *Mega-Toolbox.*

address

is the memory address of the font.

fontnum

is a small negative integer in the range -1...-16. After a font has been installed, it can be referenced using this font-number.

If the font is to be read from disk, you may use the ALCHP_L() or RESPR_L() function. For example.

```
SET_FONT -1, RESPR_L( mdv1_boldfont )
```

SHOW, SHOW_R

See also:

HIDE(), SHOWAT, WSAVE

Syntax:

```
SHOW #n [,address [,mode]]
SHOW_R #n [,address [,mode]]
```

Function:

Both commands show a window previously saved with a HIDE() or HIDE_C() command, but SHOW_R also de-allocates the storage holding the image.

address

is of course the address of the stored image in the common heap. If this is given as zero or omitted, then the address returned by the most recent HIDE() instruction relating to the given window will be used instead.

mode

determines the transfer mode, as follows.

Mode	Function
0 (default)	Normal, i.e. overwrite mode.
<= 255	Is an 8 bit value where each pixel corresponds to a colour (bit 0 to black, bit 1 to blue, etc.) and only those pixels whose colour is selected (i.e. whose corresponding mask bit is set) are transferred.
= 256	The image is AND'ed with the current contents of the window.
= 257	The image is OR'ed with the current contents of the window.
= 258	The image is XOR'ed with the current contents of the window. (Two consecutive commands restore the original contents.)
< 0	All the words of the stored image are AND'ed with the absolute value of this number and then transferred. This feature allows a (limited) control over the colours of the resulting image on the screen. For example a value of -43775 (=-BIN('1010101011111111') suppresses all flashing (in low resolution mode), and the value -255 turns green to black and white to red (in high resolution mode).

Note that the image could also have been on microdrive by a WSAVE or WSAVE_C command, and loaded in memory using ALCHP_L() or EXPAND_L().

SHOW_S

See also:

SHOW

Syntax:

SHOW_S [#n][,address]

Function:

This is a variant of the SHOW command which swaps the window contents and the given memory area instead of just copying the hidden image to the screen. The transfer mode is always "overwrite" and the stored image is must not have been compressed.

SHOWAT

See also:

SHOW

Syntax:

```
SHOWAT [#n,] address,x,y [,mode]
```

Function:

This command places a stored image anywhere on the screen, at pixel coordinates **x**, y. The window *#n* defines the region of visibility and all pixels outside it will be masked. If the *#n* argument is omitted no masking takes place.

address

is the address of the image, which must have been stored by a HIDE() command (i.e. not in a compressed form), or a "Not Implemented" error will occur

mode

is similar to the equivalent argument in SHOW, but has four additional functions (values 259-262). The meanings of *mode* are as follows.

Mode	Function
0 (default)	Normal, i.e. overwrite mode.
<= 255	Is an 8 bit value where each pixel corresponds to a colour (bit 0 to black, bit 1 to blue, etc.) and only those pixels whose colour is selected (i.e. whose corresponding mask bit is set) are transferred.
= 256	The image is AND'ed with the current contents of the window.
= 257	The image is OR'ed with the current contents of the window.
= 258	The image is XOR'ed with the current contents of the window. (Two consecutive commands restore the original contents.)
= 259	Normal (overwrite) mode.
= 260	The image is shown upside-down.
= 261	The image is mirrored (left-right inversion).
= 262	The image is rotated through 180 degrees.
> 262	Any other positive value is assumed to be the address of another stored image (not compressed) which will be used to mask the first image. The two images need not have the same dimensions, and if the mask is smaller along one or both dimensions, it will be repeated as many times as required. This transfer mode may be used for special effects, such as smoothly revealing a stored image.
< 0	All the words of the stored image are AND'ed with the absolute value of this number and then transferred. This feature allows (limited) control over the colours of the resulting image on the screen. For example a value of -43775 (=-BIN('1010101011111111') suppresses all flashing (in low resolution mode), and the value -255 turns green to black and white to red (in high resolution mode).

Note that the x, y coordinates may be negative in order to show only part of the image.

SHOWAT works well if the original window was "word-aligned", i.e. its right and left borders were on the boundary between two memory words in the display RAM or, in other words, both its width and X-position were multiples of eight.

SIGN()

Syntax:

```
value = SIGN( string$ )
```

Function:

Returns the sign value of its argument, i.e. -1,0 or 1 for negative, null or positive numbers respectively. It may also return -17 if the argument cannot be converted into a valid floating number. This function is useful for writing error-trapped input routines or to implement procedures or functions with default values for their arguments.

Note that if the *string$* argument is not a number in a valid format, the QL expression evaluator may give erroneous results or even crash the system. To avoid this,you should not use SIGN() inside expressions. So instead of writing

```
    result = SIGN( a$ )*100
```

you should use

```
    t = SIGN( a$ )
    result = t*100
```

When checking input from the user, you may find the following function useful:

```
    DEFine FuNction IS_NUMBER( a$ )
       RETurn "9"&a$<>9
    END DEFine
```

SLIDE_X

See also:

WSAVE, WSAVE_C

Syntax:

```
    SLIDE_X [#n,] times, timeout, address {,address}
```

Function:

Creates a job which shows a sequence of stored images (compressed or not) the given number of *times* (-1 means "forever") with a given *timeout* (interval) between one image and the next one.

SPOS

See also:

FPOS()

Syntax:

```
    SPOS #n, pointer
```

Function:

Sets the position pointer of the file linked to channel *#n*. The pointer indicates the location in the file of the next byte to be read or written. Negative values or values greater than the actual length of the file do not cause errors, but the pointer will be set to the start or end of the file respectively.

STICK()

see also:

STICK_READ, STICK_USE

Syntax:

value = STICK(n)

Function:

Returns the current position of a virtual cursor that is controlled by either the joy-stick or cursor keys. It is also able to read the state of the space bar/joy-stick fire button. The value of *n* determines the meaning of the returned value according to the following table.

n	Returned value
0	The state of the fire button (zero or one).
1	The current x-increment.
2	The current y-increment.
3	The x-coordinate of the virtual cursor.
4	The y-coordinate of the cursor.
5	The lower limit of the x-coordinate.
6	The upper limit of the x-coordinate.
7	The lower limit of the y-coordinate.
8	The upper limit of the y-coordinate.
9	The increment for un-SHIFT'ed keys.
10	The increment for SHIFT'ed keys.
11	The increment for CTRL'ed keys.
12	The increment for ALT'ed keys.

The returned values refer to the latest execution of STICK_READ.

STICK_READ

See also:

STICK, STICK_USE

Syntax:

value = STICK_READ

Function:

This procedure reads the arrows keys (or the CTL1 port) and prepares the values that will be returned by the STICK function.

STICK_USE

See also:

STICK(), STICK_READ

Syntax:

STICK_USE xstart,ystart [,xmin,xmax,ymin,xmax [,normal,shift,ctrl,alt]]

Function:

Sets the initial values for the STICK function.

xstart,ystart

are the initial values of the two coordinates.

xmin,xmax,ymin,ymax

if specified, set limits for the x and y coordinates.

normal,shift,ctrl,alt

are also optional, and specify scaling factors that will be applied to the cursor movement when the SHIFT, CTRL or ALT keys are being held down. The defaults are as follows: normal is x1; SHIFT is x4; CTRL is x16; and ALT is x64. For example, to ignore all but the arrow keys, use

```
STICK_USE 0,0,0,512,0,256,1,1,1,1
```

NOTE: Due to A QL bug, the state of the CTRL and ALT keys cannot be correctly determined when the left and up arrow keys are pressed simultaneously.

STORE

See also:

FETCH()

Syntax:

```
STORE x, number {,number}
```

Function:

This provides a sophisticated POKE command to a protected area of RAM reserved by the *Mega-Toolbox*. When used with FETCH, it enables you to communicate floating point values between programs and jobs or to preserve values during a CLEAR command.

Storage for sixteen values is available, referenced 0-15. You can store several values at once, with *x* indicating the reference of the first value in the list. Thus

```
STORE 4,1,2,3
```

would store 1 in the 5th slot, 2 in the 6th and 3 in the 7th. To retrieve the 5th value (=1), you would use

```
value = FETCH( 4 )
```

Clashes between multi-tasking jobs trying to access these values simultaneously are automatically prevented by the *Mega-Toolbox* which ensures that no job can start a STORE or FETCH() whilst another job has a STORE or FETCH() in progress.

STR$()

Syntax:

```
string$ = STR$( arg {,arg} )
```

Function:

This is an enhanced version of the CHR$() function, which will concatenate any number of mixed numeric and string arguments into a single string.

For example

```
PRINT STR$( "This is ", 49, " example." )
```

would print "This is 1 example.", and

```
PRINT #n, STR$( 27, "E", "The Title", 27, "F" );
```

would print "The Title" in bold type on an Epson compatible printer attached to *#n*.

SUBSTR()

Syntax:

```
value = SUBSTR( a$, b$ [,n] )
```

Function:

This performs a similar function to the existing Superbasic INSTR operator but works on the entire ASCII character set. Case is not significant.

The returned value indicates the start of the first occurrence of *a$* in *b$*. Zero will be returned to indicate that no match was found. If *n* is given, the search starts at the nth character in *b$*.

SUSPEND

See also:
JOBS, REMOVE, RELEASE, SETPRIOR

Syntax:

```
SUSPEND jobid [,timeout]
```

Function:

Suspends a job for a given *timeout* expressed in 50th's of a second. If the *timeout* is omitted, or is given as -1 the job will be suspended indefinitely.

There are two special values for *jobid:* -1 means "this job", -3 (or lower) means "all jobs but this one".

SWAP

See also:
OPTION_USE

Syntax:

```
SWAP arg1, arg2 {,argN}
```

Function:

Exchanges the contents of two or more arguments. If more than two arguments are specified, the value of the first one is passed to the second one, and so on, until the value of the last is transferred to the first. Arguments may be of any type (integer, floating or strings), provided that the resulting set of assignments is legal.

NOTE: This command will not work in compiled programs because compilers do not allow the return of values through arguments.

SYSTEM_ADDR, SYSTEM_ADDR()

See also:
BASIC_ADDR, VAR_ADDR

Syntax:

```
address = SYSTEM_ADDR
value = SYSTEM_ADDR( n )
```

Function:

When used without arguments, this function returns the address of system variables, which are normally at $28000 (hex), but could theoretically change in future (!) versions of QL.

The latter form is functionally equivalent to

```
value = PEEK_L(SYSTEM_ADDR+n)
```

and is useful for reading some system variables. For example SYSTEM_ADDR(28) returns the base of the resident procedure area, and SYSTEM_ADDR(32) the address of the last byte of RAM plus one.

TEST_SET()

See also:

BUFF_ADDR

Syntax:

```
value = TEST_SET( address )
```

Function:

Performs a 68000 TAS instruction. The byte at the given address is tested and set in a single indivisible operation, and its former value is returned. This instruction is useful in a multi-tasking environment to implement "semaphores" to arbitrate between different jobs accessing the same resource.

For example, to avoid problems where several jobs have access to the *Mega-Toolbox* buffer located at BUFF_ADDR, simultaneous access of the buffer can be prevented by using the first location of the buffer as a "lock". This can be implemented by including the following code in each compiled program with access to the buffer.

```
REPeat wait: IF TEST_SET( BUFF_ADDR )=0 THEN EXIT wait
    .
    .
REMark Use the buffer as desired, but of course don't
REMark alter the byte at (BUFF_ADDR)
    .
    .
REMark When finished...
POKE BUFF_ADDR, 0 : REMark release the buffer
```

TRIM$()

Syntax:

```
string$ = TRIM$( a$ )
```

Function:

Removes all trailing blanks from a string. When used with REVERSE$() it enables you to remove leading blanks and is also useful for getting round the Superbasic bug which causes blanks to be appended to a sliced string array.

UPPER$()

See also:

LOWER$()

Syntax:

```
string$ = UPPER$( a$ )
```

Function:

Converts a string to upper case.

VAR_ADDR()

See also:

SYSTEM_ADDR, BASIC_ADDR

Syntax:

```
address = VAR_ADDR( variable )
```

Function:

This function returns the address (relative to A6) of a given *variable,* which may be a single variable or expression of any type (integer, string or floating), or even an array or sub-array (in which case the address of its first element is returned). This is very useful when passing strings, floating point numbers or arrays of parameters to a user-defined machine code routine, or to QTRAP instructions with negative values for trapnumber (which forces arguments used by the trap to be relative to A6.

If used from within compiled programs, note that OPTION_USE must be used to alter the way that parameters are passed. Otherwise, a "Not implemented" error will result.

VAR_TYPE()

Syntax:

```
type = VAR_TYPE( variable )
```

Function:

Returns the type of a variable:-

1 means string
2 means real (floating point)
3 means integer

WAIT_R, WAIT_S

Syntax:

```
WAIT_R jobid [,timeout]
WAIT_S jobid [,timeout]
```

Function:

These procedures perform a simple form of synchronization between jobs. WAIT_R suspends the current job until the job whose id is *jobid* gets removed from the system.

WAIT_S suspends the current job until the referenced job is either suspended or removed.

timeout

in both cases sets a limit to the length of time that the calling job will wait, in fiftieths of a second. If *timeout* is omitted, a default value of -1 is used which causes an indefinite wait.

WINDOW_OP

See also:

MRECOL, RECOL_X

Syntax:

```
WINDOW_OP [#n,] andvalue [,xorvalue]
```

Function:

Each word of the screen RAM owned by the window *#n* is AND'ed with *andvalue* and if given, exclusive OR'ed with *xorvalue*.

This command enables you to very quickly change **all** colours in the given window, or to suppress flashing, etc.

The following example shows the values required to leave a window completely unchanged - most other values will perform some change.

```
WINDOW_OP 65535,0
```

WLOAD

See also:

SLIDE_X, WSAVE, WSAVE_C

Syntax:

```
WLOAD [#n,] filename
```

Function:

Loads a window previously stored with a WSAVE or WSAVE_C command.

This works well only if the destination window is the same as the saved window or if the two windows have the same width and their X-positions differ by a multiple of eight.

WSAVE, WSAVE_C

See also:

WLOAD

Syntax:

```
WSAVE #n, filename
WSAVE_C #n, filename
```

Function:

Both commands save the contents of a window in a file.

WSAVE_C attempts to save the contents in a compressed form. Compression not only reduces the volume of storage required, but also reduces the amount of time spent loading and saving to microdrives or discs.

XCUR(), YCUR()

Syntax:

```
    value = XCUR( #n )
    value = YCUR( #n )
```

Function:

These functions return the x and y position of the character cursor.

XDIR

Syntax:

```
    XDIR #n, devicename
```

Function:

Extended directory: shows the length of each file, and its data space if it is an EXECutable program.

XFORMAT

Syntax:

```
    XFORMAT #n, devicename [,sectors]
```

Function:

Formats a medium five times or, if the sector argument is specified, until the format procedure returns a number of good sectors greater than or equal to the given value (it tries a maximum of five times). This is useful for formatting new microdrive cartridges.

XINC(), YINC()

Syntax:

```
    value = XINC( #n )
    value = YINC( #n )
```

Function:

These functions return the current cursor increments.

XMAP(), YMAP()

Syntax:

```
    value = XMAP( #n, x )
    value = YMAP( #n, y )
```

Function:

These functions provide a conversion from graphic coordinates of the given window to the pixel coordinate system.

The graphic coordinates of a window have their origin at its bottom left hand corner of the window, whereas the pixel coordinate origin is at the top left hand corner of the screen. The positive x direction of the pixel coordinate system (which is used to specify the position of a window) is downwards, i.e. opposite to that for graphic coordinates.

XPOS(), YPOS()

Syntax:

```
value = XPOS( #n )
value = YPOS( #n )
```

Function:

These functions return the position of the upper left hand corner of the given window (in pixel coordinates).

XSIZE(), YSIZE()

Syntax:

```
value = XSIZE( #n )
value = YSIZE( #n )
```

Function:

These functions return the dimension of the given window (in pixels).

ZOOM

See also:

HIDE(), RECOLSHOW, SHOW

Syntax:

```
ZOOM [#n,] address, xsize, ysize [,xpos, ypos]
```

Function:

Shows an image (which must not be compressed) stored at *address* but with the ability to alter its dimensions. As for SHOWAT, the stored image is must be word-aligned.

xsize,xsize

specify the number of pixel columns and rows of the resulting image. Hence, ZOOM can be used to reduce an image along one or both directions, or to enlarge it along one direction and reduce along the other etc. These (size) parameters need not be multiples of the stored image dimensions, nor of the *#n* window dimensions, so even fractional factors of reduction or magnification are obtainable.

xpos,ypos

are the coordinates of the upper left-hand corner of the area (in the stored image) which will be zoomed and shown. This makes it possible to zoom the image and show it one piece at a time.

6. General Notes For Programmers

This chapter gives some details about various toolkit features that may be of interest. This includes details of the compression method for stored images, and notes for programmers wishing to use the *Mega-Toolkit* with compiled programs.

6.1. Installation

You may have noticed that the BOOT program supplied passes parameters to the *Mega-Toolbox* when it is first initialised. The statement has the format:-

```
CALL address, fnkey_number, fnkey_length
```

The latter two values can be varied according to your needs because they control allocation of memory for use by the KEY command. By minimising the values of these two parameters, you will reduce the amount of memory reserved for use by the KEY command. The meanings of each parameter is as follows:-

address
: This is the address of the routine within the *Mega-Toolbox* that does the initialisation and should not be changed.

fnkey_number
: This sets the maximum number of function keys that can be programmed using the KEY command. Up to twenty combinations can be defined (using the CTRL, ALT and SHIFT keys in combination with F1 to F5). Set this parameter to the maximum number of keys that your application will require.

fnkey_length
: This defines the amount of memory set aside for storing the strings that will be assigned to each function key. You should specify a value long enough to hold the largest string that you expect to program, up to a maximum of 128 characters.

6.2. Basic Compilers

The *Mega-Toolbox* has been designed to avoid problems when used by compiled programs and has been tested with both *Supercharge* and *Turbo* compilers. Use with these compilers leads to some restrictions described shortly. The *Mega-Toolbox* has not been tested with *Q Liberator*, the main alternative compiler, but this is unlikely to present more problems than the others because it is in general much more tolerant.

Neither *Supercharge* or *Turbo* support extensions which return values through their arguments, i.e. through the BP.LET vectored routine. For this reason, the *Mega-Toolbox* deliberately avoids this facility wherever possible. There are only three exceptions:

```
SWAP, QCALL, QTRAP
```

The QREG() function has been designed to enable values returned by the latter two functions without needing to return the values through their arguments. You have complete control over which method is used to return values from these functions through the OPTION_USE command.

A further point to note is that unlike interpreted programs, compiled programs can be more efficient if functions returning integer values are named with a trailing "%". With this in mind, a second version of the *Mega-Toolbox* has been provided in which the following functions have replaced the existing ones which lack the "%" suffix:

> BTST%, COLOUR%, COLUMNS%, COUNT%, DMODE%, FN_KEY%, FREE_CHAN%, FREE_SECT%, JOB_STAT%, ROL%, ROL_W%, ROWS%, SEARCH%, SEARCH_N%, SIGN%, STICK%, SUBSTR%, TEST_SET%, VAR_TYPE% XCUR%, XINC%, XMAP%, XPOS%, XSIZE%, YCUR%, YINC%, YMAP%, YPOS%, YSIZE%

The integer version is contained in the file "MEGA_BIN%". Examine the BOOT program for details of how to load it.

6.3. HIDE And SHOW Commands

The *Mega-Toolbox* uses the long word at offset $24 inside the Superbasic channel definition block to store the address returned by the latest HIDE/HIDE_C function call on a channel. This makes it possible to provide a default address for the SHOW/SHOW_R commands. Normally this long word is not used, but could cause incompatibilities with other toolkits if they were to use the same location for some other function.

6.4. Image Compression Method

When images are compressed using HIDE_C/WSAVE_C the following simple technique is used.

Where the data contains consecutive words of the same value, the first two are left unchanged, but any further identical words are replaced by a word which indicates how many such occurrences have been replaced. Hence if there were only two identical words in sequence, the third word would be zero. The method is simple and effective, not taking too much time to compress and decode. Most images will show significant compression using this technique, but in cases where the method would actually expand the image, it will be left unaltered.

6.5. Stored Image Header

The following details are provided for use by programmers who wish to manipulate stored images themselves, possibly making use of the following *Mega-Toolbox* commands: MCOPY, MEXCHANGE, MFILL, EXCOPY, and EXFILL.

There is a header consisting of two 16-bit words at the beginning of all images stored either in RAM or in files.

The first word contains the width of the image and must be multiplied by eight to obtain the width in pixels.

The second word contains the height of the image in pixels, and has its most significant bit set only if the image has been compressed. Standard file error

6.6. Using The QL's Alternative Screen

The default (first) QL screen occupies 32K of RAM from $20000 to $27fff. The alternative (second) screen must sit immediately above this, from $28000 to $2ffff, but unfortunately clashes with the QL's system variables which occupy approximately the first 5K of the second screen. This means that the first 6th (5/32) of the second screen can never display anything meaningful, because to do so would mean overwriting the system variables causing the QL to crash.

The location of the system variables means that the first 5K of the second screen is effectively useless. Worse still, the common heap starts to be reserved from immediately above the system variables. Therefore, in order to write to the second screen it is necessary to reserve memory in the heap, preferably before any of the heap has been taken by another program (or toolkit etc.). This is done using the DUAL_SCR command, which reserves memory and returns the start of the second screen. Any heap reserved before the DUAL_SCR command will eat into the memory available and will reduce the useful area of the second screen. Therefore, if you wish to use the second screen, the memory should be reserved as soon as possible after a reset. When experimenting, the following command sequence will show you how much of the second screen can be used:

```
DUAL_SCR
DUAL_COPY 1
```

The first command reserves memory, and the second copies the contents of the second screen to the first, overwriting an area corresponding to that available in the second screen.

Ideally, all but the top 5th of the screen should be overwritten, although you may find that various QL add-ons such as disc interfaces, ROM toolkits etc. reserve memory from the heap, shrinking the size of the second screen.

Once memory has been reserved for the second screen, heap reserved by other programs will not affect it, and the display can be manipulated in a number of ways.

DUAL_COPY
: can be used to copy the contents of the two screens from one to the other. The entire area overlapped by the second screen can be copied, or just that part which lies within a given window.

DISPLAY
: can be used to alter the displayed image to show a mixture of both the first and second screens - in horizontal bands of variable depth. Each band can be taken from either screen, and can be in either high or low resolution screen mode. Parts of the screen can also be blanked entirely enabling drawing operations to be performed invisibly before being made to appear instantaneously.

6.7. File I/O Errors

FILE_LEN, FILE_DSPACE, FREE_SECT, ALCHP_L, RESPR_L, EXPAND_L

Whenever one of the (above) functions that performs a file operation encounters a non standard error code the error code returned by that function will be -7 which corresponds to "File not found". This might occur when using add-on hardware devices that do not return the standard QDOS error codes, and simplifies error trapping by ensuring that only standard error codes can ever be reported.

6.8. Sharing Of Facilities Between Multi-tasking Programs

Mega-Toolbox commands and functions have been designed to be re-entrant wherever possible meaning that multi-tasking programs can all make use of its commands simultaneously without problems. However, in the following cases, it has either not been practical to make the routines re-entrant, or for some other reason it is impractical for two programs to access the same command at the same time:

DISPLAY, DUAL_SCR, KEY, KEY_USE, KEYBOARD, OPTION_USE, PLAY_X, RESET_SCR, STICK_USE and STORE.

7. Distributing Programs With The Mega-Toolbox

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

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

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

8. What To Do If Things Go Wrong

This is a chapter to help sort out problems

BAD OR CHANGED MEDIUM

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

BAD NAME

If when you type a *Mega-Toolbox* command, or run a program which makes use of facilities and obtain a "Bad name" message, this may indicate that for some reason the commands have not been properly linked into Superbasic. If you are typing commands at the keyboard, try again as some software can cause this to happen intermittently. If you still have problems, first try re-booting your QL and verify that the problem is still present.

If you have this problem when using the *Mega-Toolbox* inside a program, make sure that you are using the boot program provided to load and link the commands, and that this is done before your own program is loaded. If you are trying to link the *Mega-Toolbox* from within your own program, you are likely to find that some versions of QL refuse to link in the new commands. Instead, you should modify the boot program provided to load your program after the toolkit, solving the "Bad name" problem and keeping the loading process fully automatic.

IF ALL ELSE FAILS

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

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

Appendix A

Details of Demonstration Programs

File	Notes	Commands Used
arrays_eg	Contains a wide selection of general purpose routines for fast and flexible manipulation of arrays.	VAR_TYP, VAR_ADDR, BASIC_ADDR, MEXCHANGE, EXCOPY, MCOPY, MFILL, MSEARCH_A, PUT$
square_eg	A routine for drawing a square which shows how you can implement optional parameters for SuperBasic routines.	SIGN, SWAP
jobinfo_eg	Shows how to access information in contained in job headers, and how to use CHOOSE$	JOB_ADDR, JOB$, CHOOSE$
chaninfo_eg	Shows how to obtain information about a screen window from its QDOS header and shows the use of bit test and manipulation routines.	CHAN_ADDR, CHAN_ID, CHOOSE$, BTST, ROL, GET, GETMEM$
sysinfo_eg	Shows how to access QDOS system variables including the address of heap, resident procedure area, job headers, display mode etc. Lots of useful features.	SYSTEM_ADDR, CHOOSE$, BTST
bio_eg	Shows how to verify input using several new functions, but is also a useful biorhythm generating program.	SIGN, DAZYS, YCUR
fonts_eg	Shows how to construct your own fonts and add them to basic.	FONT_USE, BIN
fplot_eg	Shows error trapping and how to write self modifying programs - very bad practice of course, but powerful!	ENTER, DISPLAY, KEY_USE, CLEAR_Q
qtrap_eg	Contains several useful procedures showing use of QDOS traps directly from basic. Demonstrates opening channels, files, directories etc. - all with full error trapping in basic.	QTRAP, BUF_ADDR, QPOKE, JOB_ID, CONNECT_C, BUFF_ADDR, CHAN_ID, FREE_CHAN, GETMEM$
excopy_eg	Simple graphic demo, including drawing whilst the screen is turned off and some special effects.	EXCOPY, EXFILL, DISPLAY, CHOOSE$, DISK, HIDE, SCR_ADDR, SHOWAT
box_eg	Simple demonstration of using BOX.	BOX
disk_eg	Simple demonstration of using DISK.	DISK
stick_eg	Shows how to read joystick position and cursor keys.	STICK_READ, STICK, CHOOSE$

Problem Report Form

Please use this form (or a photocopy) to report any problems/suspected bugs or documentations errors associated with this software, and return to:

Software Problem Report, Compware, 57 Repton Drive, Haslington, Crewe, CW1 1SA.

Personal Details
Name:
Address:
Telephone:

Program Details	
Program: QL Mega-Toolbox	Date Purchased:
Program version:	Purchased From:
Manual issue date: December 7, 1987	

QL Configuration	
QL Version:	QDOS version:
QL RAM:	RAM Make/model:
Disc interface (make and model):	
Other add-ons (ROMS, mice etc.):	

Problem Details
Please describe the problem in as much detail as possible. If we are to do anything about it, we must be able to duplicate the fault so please describe all the steps leading up to the problem explicitly, and note exactly, any error messages produced. Use the reverse of this form to describe the problem and continue on additional sheets if necessary.   Please supply a microdrive containing a simple program demonstrating the fault.