DJToolkit Documentation

Release 1.17.0

Norman Dunbar

Oct 26, 2025

CONTENTS

This document covers only those commands found in DJTOOLKIT versions up to 1.17. These commands have also been incorporated into the Online SuperBASIC Manual which can be found at ReadTheDocs.

Contents:

CHAPTER

ONE

INTRODUCTION

WRITTEN BY NORMAN DUNBAR, 1993-94

This toolkit has been produced at the suggestion of *Dilwyn Jones* in an effort to provide `QLiberator` users with some of the file & memory handling utilities, direct file access (in internal format) & positioning commands that are found in the `Turbo Toolkit` and `Toolkit 2` etc. In addition, there are a few routines not (yet) found in any other toolkit.

All of the procedures and functions in this toolkit can be compiled by `QLiberator`, but one of the functions, `DEV_NAME`, cannot be compiled by `Turbo` or `Supercharge` as it modifies its parameter as well as returning a string.

This toolkit may be supplied as part of a commercial or shareware or public domain program which has been `QLiberator` compiled. It should be supplied *linked* to the object program, not loaded by a `BOOT` program.

To link this toolkit into a `QLiberator` compiled program, use the following compiler directive somewhere near the start of the program to be compiled. The drive name is where the compiler can find the toolkit file:

```
110 REMark $$asmb=FLP2_DJToolkit_BIN,0,12
```

1.1 COPYRIGHT NOTICE AND DISCLAIMER

Note: Yeah, yeah, yeah! Since that was originally written, sometime in the early 1990's, the world has moved on and most of the old QL programs are pretty much in the public domain, free ware or open source. As indeed, this one is!

Norman Dunbar, December 2016

1.2 QUESTIONS ABOUT THE TOOL KIT

1.2.1 WHAT IS A TOOL KIT?

A tool kit is a set of BASIC extensions, new commands and functions which add to the number of "words" understood by the QL's BASIC language. These new extensions greatly add to the power and versatility of SuperBASIC, by adding the facility to perform new actions, or by simplifying a task which is difficult to program at the moment

1.2.2 CAN I USE THE TOOLKIT IN MY OWN PROGRAMS?

Yes, you can use these commands both in interpreted BASIC programs and compiled BASIC programs, but see the note above regarding use with Turbo.

1.2.3 DOES IT WORK WITH OTHER TOOLKITS?

We have tried to ensure that there is no clash, but it is impossible to test it against every other piece of software or hardware. Please let us know if you discover any incompatibilities so that we can try to sort them out.

1.3 BRIEF DESCRIPTION OF THE NEW COMMANDS

ABS_POSITION	Set file position absolute
BYTES_FREE	How much free memory is left, in bytes
CHECK	Test to see if a machine code PROC/FN exists
DEV_NAME	Scan the Directory Device list, returning the next name
DISPLAY_WIDTH	How many bytes are used to hold one screen line
DJ_OPEN	Opens a file, returns error or channel id
DJ_OPEN_IN	Ditto, similar to OPEN_IN
DJ_OPEN_NEW	Creates a file, returns channel id or error
DJ_OPEN_OVER	Overwrites a file, returns error or channel id
DJ_OPEN_DIR	Opens a device directory for access
DJTK_VER$	Return the toolkit version number as a string
FETCH_BYTES	Get some bytes from a channel
FILE_BACKUP	Get the backup date for a specific file
FILE_DATASPACE	Get the file's dataspace
FILE_LENGTH	Get the file's length
FILE_POSITION	Get the current position in the file
FILE_TYPE	Get the file's type
FILE_UPDATE	Get the file's update date
FILLMEM_B	Fill memory with a byte value
FILLMEM_L	Fill memory with a long value
FILLMEM_W	Fill memory with a word value
FLUSH_CHANNEL	Flush the data on a channel to a device
GET_BYTE	Fetch one byte from a channel
GET_FLOAT	Fetch 6 bytes from a channel
GET_LONG	Fetch 4 bytes from a channel
GET_STRING	Fetch a QDOS string from a channel
GET_WORD	Fetch 2 bytes from a channel
KBYTES_FREE	How much free memory is left in Kbytes

continues on next page

Table 1 – continued from previous page

LEVEL2	Test whether level 2 drivers are present on a channel
MAX_CON	Returns the absolute limits of a SCR or CON channel
MAX_DEVS	Counts the number of directory devices. See DEV_NAME
MOVE_MEM	Move memory around
MOVE_POSITION	Set a file position relative to its current one
PEEK_FLOAT	Read 6 bytes from memory into a float variable
PEEK_STRING	Get bytes from memory into a string
POKE_FLOAT	pokes a floating point variable into memory
POKE_STRING	Store the string in memory at a given address
PUT_BYTE	Send 1 byte to a channel
PUT_FLOAT	Send 6 bytes to a channel
PUT_LONG	Send 4 bytes to a channel
PUT_STRING	Send a QDOS string to a channel
PUT_WORD	Send 2 bytes to a channel
QPTR	Is the Pointer Environment available
READ_HEADER	Read the header for a file into a buffer
RELEASE_HEAP	Remove some space allocated with RESERVE_HEAP
RESERVE_HEAP	Get some Common Heap space for a program to use
SCREEN_BASE	Find out where the screen memory starts for a channel
SCREEN_MODE	Returns the current screen mode, 4 or 8
SEARCH_C	Look in memory for a string, case is considered
SEARCH_I	Ditto, but case is ignored
SET_HEADER	Set the header for a file
SET_XINC	Change horizontal spacing between characters
SET_YINC	Change vertical spacing between lines of characters
SYSTEM_VARIABLES	Find out where the system variables are
USE_FONT	Change the fonts used by a channel
WHERE_FONTS	Find the addresses of the two fonts used on a channel

In the following descriptions, all parameters must be supplied as there are no defaults, in addition, when a channel number is being passed, either as a number or as a variable, it must be preceded by a hash (#).

1.4 QDOS ERROR CODES

Many of the above functions return a valid result, such as an address, or a negative error code. The QDOS error codes are listed below for reference.

- -1 Not complete
- -2 Invalid job
- -3 Out of memory
- -4 Out of range
- -5 Buffer overflow
- -6 Channel not open
- -7 Not found
- -8 File already exists
- -9 In use
- -10 End of file

- -11 Drive full
- -12 Bad device name
- -13 Xmit (transmit) error
- -14 Format failed
- -15 Bad parameter
- -16 File error
- -17 Error in expression
- -18 Arithmetic overflow
- -19 Not implemented
- -20 Read only
- -21 Bad line

CHAPTER

TWO

DJTOOLKIT 1.17

2.1 ABS_POSITION

Syntax	ABS_POSITION #channel, position
Location	DJToolkit 1.17

This procedure will set the file pointer to the position given for the file attached to the given channel number. If you attempt to set the position for a screen or some other non-directory device channel, you will get a bad parameter error, as you will if position is negative.

If the position given is 0, the file will be positioned to the start, if the position is a large number which is greater than the current file size, the position will be set to the end of file and no error will occur.

After an ABS_POSITION command, all file accesses will take place at the new position.

EXAMPLE

```
1500 REMark Set position to very end, for appending data
1510 ABS_POSITION #3, 6e6
1520 ...
```

CROSS-REFERENCE

MOVE_POSITION.

2.2 BYTES_FREE

Syntax	memory = BYTESFREE
Location	DJToolkit 1.17

This simple function returns the amount of memory known by the system to be free. The answer is returned in bytes, see also KBYTES_FREE. For the technically minded, the free memory is considered to be that between the addresses held in the system variables SV_FREE and SV_BASIC.

EXAMPLE

```
...
2500 freeMemory = BYTES_FREE
2510 IF freeMemory < 32 * 1024 THEN
2520    REMark Do something here if not enough memory left...
2530 END IF
...
```

CROSS-REFERENCE

KBYTES_FREE.

2.3 CHECK

Syntax	oops = CHECK('name')
Location	DJToolkit 1.17

If name is a currently loaded machine code procedure or function, then the variable oops will be set to 1 otherwise it will be set to 0. This is a handy way to check that an extension command has been loaded before calling it. In a Turbo'd or Supercharged program, the EXEC will fail and a list of missing extensions will be displayed, a QLiberated program will only fail if the extension is actually called.

EXAMPLE

```
1000 DEFine FuNction CheckTK2
1010   REMark Is TK2 present?
1020   RETurn CHECK('WTV')
2030 END DEFine
```

2.4 DEV_NAME

Syntax	device$ = DEV_NAME(address)
Location	DJToolkit 1.17

This function must be called with a floating point variable name as its parameter. The first time this function is called, address *must* hold the value zero, on all other calls, simply pass address *unchanged* back. The purpose of the function is to return a directory device name to the variable device$, an example is worth a thousand explanations.

```
1000 addr = 0
1010 REPeat loop
1020   PRINT "<" & DEV_NAME(addr) & ">"
1030   IF addr = 0 THEN EXIT loop: END IF
1040 END REPeat loop
```

This small example will scan the entire directory device driver list and return one entry from it each time as well as updating the value in 'addr'. The value in addr is the start of the next device driver linkage block and *must not be changed* except by the function DEV_NAME. If you change addr and then call DEV_NAME again, the results will be very unpredictable.

The check for addr being zero is done as this is the value returned when the final device name has been extracted, in this case the function returns an empty string for the device. If the test was made before the call to DEV_NAME, nothing would be printed as addr is zero on entry to the loop.

Please note, every QL has at least one device in the list, the 'MDV' device and some also have a device with no name as you will see if you run the above example (not the last one as it is always an empty string when addr becomes zero).

The above example will only show directory devices, those that can have DIR used on them, or FORMAT etc, such as WIN, RAM, FLP, FDK etc, however, it cannot show the non-directory devices such as SER, PAR (or NUL if you have Lightning), as these are in another list held in the QL.

Note

From version 1.14 of DJToolkit onwards, there is a function that counts the number of directory devices present in the QL. See MAX_DEVS for details.

CROSS-REFERENCE

MAX_DEVS.

2.5 DISPLAY_WIDTH

Syntax	bytes_in_a_line = DISPLAY_WIDTH
Location	DJToolkit 1.17

This function can be used to determine how many bytes of the QL's memory are used to hold the data in one line of pixels on the screen. Note that the value returned has nothing to do with any *window* width, it always refers to the total *screen* display width.

Why include this function I hear you think? If you run an ordinary QL, then the result will probably always be 128 as this is how many bytes are used to hold a line of pixels, however, many people use Atari ST/QLs, QXL etc and these have a number of other screen modes for which 128 bytes is not enough.

This function will return the exact number of bytes required to step from one line of pixels to the next. Never assume that QDOS programs will only ever be run on a QL. What will happen when new Graphics hardware or emulators arrive? This function will still work, assuming that the unit uses standard QDOS channel definition blocks etc.

For the technically minded, the word at offset $64 in the SCR_ or CON_ channel's definition block is returned. This is called SD_LINEL in 'Tebby Speak' and is mentioned in Jochen Merz's *QDOS Reference Manual* and the *QL Technical Manual* by Tony Tebby et al. Andrew Pennel's book, the *QDOS Companion* gets it wrong on page 61, guess which one I used first!

2.6 DJ_OPEN

Syntax	channel = DJ_OPEN('filename')
Location	DJToolkit 1.17

Open an existing file for exclusive use. See DJ_OPEN_DIR below for details and examples.

CROSS-REFERENCE

DJ_OPEN_IN, DJ_OPEN_NEW, DJ_OPEN_OVER, and DJ_OPEN_DIR.

2.7 DJ_OPEN_IN

Syntax	channel = DJ_OPEN_IN('filename')
Location	DJToolkit 1.17

Open an existing file for shared use. The same file can be opened by other applications running at the same time. Provided they have a compatible non-exclusive OPEN mode. See DJ_OPEN_DIR below for details and examples.

CROSS-REFERENCE

DJ_OPEN, DJ_OPEN_NEW, DJ_OPEN_OVER, and DJ_OPEN_DIR.

2.8 DJ_OPEN_NEW

Syntax	channel = DJ_OPEN_NEW('filename')
Location	DJToolkit 1.17

Create a new file for exclusive use. See DJ_OPEN_DIR below for details and examples.

CROSS-REFERENCE

DJ_OPEN, DJ_OPEN_IN, DJ_OPEN_OVER, and DJ_OPEN_DIR.

2.9 DJ_OPEN_OVER

Syntax	channel = DJ_OPEN_OVER('filename')
Location	DJToolkit 1.17

Open existing file but overwrite all the contents. See DJ_OPEN_DIR below for details and examples.

CROSS-REFERENCE

DJ_OPEN, DJ_OPEN_IN, DJ_OPEN_NEW, and DJ_OPEN_DIR.

2.10 DJ_OPEN_DIR

Syntax	channel = DJ_OPEN_DIR('filename')
Location	DJToolkit 1.17

All of these DJ_OPEN functions return the SuperBasic channel number if the channel was opened without any problems, or, a negative error code otherwise. You can use this to check whether the open was successful or not.

The filename must be supplied as a variable name, file$ for example, or in quotes, 'flp1_fred_dat'.

They all work in a similar manner to the normmal SuperBasic OPEN procedures, but, DJ_OPEN_DIR offers a new function not normally found on a standard QL.

DJToolkit Author's Note

I am grateful to Simon N. Goodwin for his timely article in *QL WORLD volume 2, issue 8* (marked Vol 2, issue 7!!!). I had been toying with these routines for a while and was aware of the undocumented QDOS routines to extend the SuperBasic channel table. I was, however, not able to get my routines to work properly. Simon's article was a great help and these functions are based on that article. Thanks Simon.

EXAMPLE

The OPEN routines work as follows:

```
1000 REMark open our file for input
1010 :
1020 chan = DJ_OPEN_IN('filename')
1030 IF chan < 0
1040    PRINT 'OOPS, failed to open "filename", error ' & chan
1050    STOP
1060 END IF
1070 :
1080 REM process data in file here ....
```

DJ_OPEN_DIR is a new function to those in the normal QL range, and it works as follows:

```
1000 REMark read a directory
1010 :
1020 INPUT 'Which device ';dev$
1030 chan = DJ_OPEN_DIR(dev$)
1040 IF chan < 0
1050    PRINT 'Cannot open ' & dev$ & ', error ' & chan
1060    STOP
1070 END IF
1080 :
1090 CLS
1100 REPeat dir_loop
1110   IF EOF(#chan) THEN EXIT dir_loop
```

(continues on next page)

(continued from previous page)

```
1120   a$ = FETCH_BYTES(#chan, 64)
1130   size = CODE(a$(16)):       REMark Size of file name
1140   PRINT a$(17 TO 16 + size): REMark file name
1150 END REPeat dir_loop
1160 :
1170 CLOSE #chan
1180 STOP
```

In this example, no checks are done to ensure that the device actually exists, etc. You could use DEV_NAME to check if it is a legal device. The data being read from a device directory file must always be read in 64 byte chunks as per this example.

Each chunk is a single directory entry which holds a copy of the file header for the appropriate file. Note, that the first 4 bytes of a file header hold the actual length of the file but when read from the directory as above, the value if 64 bytes too high as it includes the length of the file header as part of the length of a file.

The above routine will also print blank lines if a file has been deleted from the directory at some point. Deleted files have a name length of zero.

Note that if you type in a filename instead of a device name, the function will cope. For example, you type in 'flp1_fred' instead of 'flp1_'. You will get a list of the files on 'flp1_' if 'fred' is a file, or even, if 'fred' is not on 'flp1_'. If, however, you have the LEVEL 2 drivers (see LEVEL2 below), and 'fred' is a sub-directory then you will get a listing of the sub-directory as requested.

CROSS-REFERENCE

DJ_OPEN, DJ_OPEN_IN, DJ_OPEN_NEW, and DJ_OPEN_OVER.

2.11 DJTK_VER$

Syntax	v$ = DJTK_VER$
Location	DJToolkit 1.17

This simply sets v$ to be the 4 character string 'n.nn' where this gives the version number of the current toolkit. If you have problems, always quote this number when requesting help.

EXAMPLE

```
PRINT DJTK_VER$
```

2.12 FETCH_BYTES

Syntax	a$ = FETCH_BYTES(#channel, how_many)
Location	DJToolkit 1.17

This function returns the requested number of bytes from the given channel which must have been opened for INPUT or INPUT/OUTPUT. It will work on CON_ channels as well, but no cursor is shown and the characters typed in are not shown on the screen. If there is an ENTER character, or a CHR$(10), it will not signal the end of input. The function will not return until the appropriate number of bytes have been read.

WARNING - JM and AH ROMS will cause a 'Buffer overflow' error if more than 128 bytes are fetched, this is a fault with QDOS and not with DJToolkit. See the demos file, supplied with DJToolkit, for a workaround to this problem.

EXAMPLE

```
LineOfBytes$ = FETCH_BYTES(#4, 256)
```

2.13 FILE_BACKUP

Syntax	bk = FILE_BACKUP(#channel)
Syntax	bk = FILE_BACKUP('filename')
Location	DJToolkit 1.17

This function reads the backup date from the file header and returns it into the variable bk. The parameter can either be a channel number for an open channel, or it can be the filename (in quotes) of a closed file. If the returned value is negative, it is a normal QDOS error code. If the value returned is positve, it can be converted to a string be calling DATE$(bk). In normal use, a files backup date is never set by QDOS, however, users who have WinBack or a similar backup utility program will see proper backup dates if the file has been backed up.

EXAMPLE

```
1000 bk = FILE_BACKUP('flp1_boot')
1010 IF bk <> 0 THEN
1020    PRINT "Flp1_boot was last backed up on " & DATE$(bk)
1030 ELSE
1040    PRINT "Flp1_boot doesn't appear to have been backed up yet."
1050 END IF
```

CROSS-REFERENCE

FILE_DATASPACE, FILE_LENGTH, FILE_TYPE, FILE_UPDATE.

2.14 FILE_DATASPACE

Syntax	ds = FILE_DATASPACE(#channel)
Syntax	ds = FILE_DATASPACE('filename')
Location	DJToolkit 1.17

This function returns the current dataspace requirements for the file opened as #channel or for the file which has the name given, in quotes, as filename. If the file is an EXEC'able file (See FILE_TYPE) then the value returned will be the amount of dataspace that that program requires to run, if the file is not an EXEC'able file, the result is undefined, meaningless and probably zero. If the result is negative, there has been an error and the QDOS error code has been returned.

EXAMPLE

```
1000 ds = FILE_DATASPACE('flp1_WinBack_exe')
1010 IF ds <= 0 THEN
1020    PRINT "WinBack_exe doesn't appear to exist on flp1_, or is not executable."
1030 ELSE
1040    PRINT "WinBack_exe's dataspace is set to " & ds & " bytes."
1050 END IF
```

CROSS-REFERENCE

FILE_BACKUP, FILE_LENGTH, FILE_TYPE, FILE_UPDATE.

2.15 FILE_LENGTH

Syntax	fl = FILE_LENGTH(#channel)
Syntax	fl = FILE_LENGTH('filename')
Location	DJToolkit 1.17

The file length is returned. The file may be open, in which case simply supply the channel number, or closed, supply the filename in quotes. If the returned value is negative, then it is a QDOS error code.

EXAMPLE

```
1000 fl = FILE_LENGTH('flp1_WinBack_exe')
1010 IF fl <= 0 THEN
1020    PRINT "Error checking FILE_LENGTH: " & fl
1030 ELSE
1040    PRINT "WinBack_exe's file size is " & fl & " bytes."
1050 END IF
```

CROSS-REFERENCE

FILE_BACKUP, FILE_DATASPACE, FILE_TYPE, FILE_UPDATE.

2.16 FILE_POSITION

Syntax	where = FILE_POSITION(#channel)
Location	DJToolkit 1.17

This function will tell you exactly where you are in the file that has been opened, to a directory device, as #channel, if the result returned is negative it is a QDOS error code. If the file has just been opened, the result will be zero, if the file is at the very end, the result will be the same as calling FILE_LENGTH(#channel) - 1, files start at byte zero remember.

EXAMPLE

```
1500 DEFine FuNction OPEN_APPEND(f$)
1510   LOCal ch, fp
1515   :
1520   REMark Open a file at the end, ready for additional
1530   REMark data to be appended.
1540   REMark Returns the channel number. (Or error)
1545   :
1550   ch = DJ_OPEN(f$)
1560   IF ch < 0 THEN
1570      PRINT "Error: " & ch & " Opening file: " & f$
1580      RETurn ch
1590   END IF
1595   :
1600   MOVE_POSITION #ch, 6e6
1610   fp = FILE_POSITION(#ch)
1620   IF fp < 0 THEN
1630      PRINT "Error: " & fp & " reading file position on: " & f$
1640      CLOSE #ch
1650      RETurn fp
1660   END IF
1665   :
1670   PRINT "File position set to EOF at: " & fp & " on file: " &f$
1680   RETurn ch
1690 END DEFine
```

CROSS-REFERENCE

ABS_POSITION, MOVE_POSITION.

2.17 FILE_TYPE

Syntax	ft = FILE_TYPE(#channel)
Syntax	ft = FILE_TYPE('filename')
Location	DJToolkit 1.17

This function returns the files type byte. The various types currently known to me are :

- 0 = BASIC, CALL'able machine code, an extensions file or a DATA file.

- 1 = EXEC'able file.
- 2 = SROFF file used by linkers etc, a C68 Library file etc.
- 3 = THOR hard disc directory file. (I think!)
- 4 = A font file in The Painter
- 5 = A pattern file in The Painter
- 6 = A compressed MODE 4 screen in The Painter
- 11 = A compressed MODE 8 screen in The Painter
- 255 = Level 2 driver directory or sub-directory file, Miracle hard disc directory file.

There *may* be others.

EXAMPLE

```
1000 ft = FILE_TYPE('flp1_boot')
1010 IF ft <= 0 THEN
1020    PRINT "Error checking FILE_TYPE: " & ft
1030 ELSE
1040    PRINT "Flp1_boot's file type is " & ft & "."
1050 END IF
```

CROSS-REFERENCE

FILE_BACKUP, FILE_DATASPACE, FILE_LENGTH, FILE_UPDATE.

2.18 FILE_UPDATE

Syntax	fu = FILE_UPDATE(#channel)
Syntax	fu = FILE_UPDATE('filename')
Location	DJToolkit 1.17

This function returns the date that the appropriate file was last updated, either by printing to it, saving it or editing it using an editor etc. This date is set in all known QLs and emulators etc.

EXAMPLE

```
1000 fu = FILE_UPDATE('flp1_boot')
1010 IF fu <> 0 THEN
1020    PRINT "Flp1_boot was last written/saved/updated on " & DATE$(fu)
1030 ELSE
1040    PRINT "Cannot read lates UPDATE date from flp1_boot. Error: " & fu & "."
1050 END IF
```

CROSS-REFERENCE

FILE_DATASPACE, FILE_LENGTH, FILE_TYPE, FILE_TYPE.

2.19 FILLMEM_B

Syntax	FILLMEM_B start_address, how_many, value
Location	DJToolkit 1.17

Fill memory with a byte value. See FILLMEM_L below.

CROSS-REFERENCE

FILLMEM_L, FILLMEM_W.

2.20 FILLMEM_W

Syntax	FILLMEM_W start_address, how_many, value
Location	DJToolkit 1.17

Fill memory with a 16 bit word value . See FILLMEM_L below.

CROSS-REFERENCE

FILLMEM_L, FILLMEM_B.

2.21 FILLMEM_L

Syntax	FILLMEM_L start_address, how_many, value
Location	DJToolkit 1.17

Fill memory with a long (32 bit) value.

EXAMPLE

The screen memory is 32 kilobytes long. To fill it all black, try this:

```
1000 FILLMEM_B SCREEN_BASE(#0), 32 * 1024, 0
```

or this:

```
1010 FILLMEM_W SCREEN_BASE(#0), 16 * 1024, 0
```

or this:

```
1020 FILLMEM_L SCREEN_BASE(#0), 8 * 1024, 0
```

and the screen will change to all black. Note how the second parameter is halved each time? This is because there are half as many words as bytes and half as many longs as words.

The fastest is FILLMEM_L and the slowest is FILLMEM_B. When you use FILLMEM_W or FILLMEM_L you must make sure that the start_address is even or you will get a bad parameter error. FILLMEM_B does not care about its start_address being even or not.

FILLMEM_B truncates the value to the lowest 8 bits, FILLMEM_W to the lowest 16 bits and FILLMEM_L uses the lowest 32 bits of the value. Note that some values may be treated as negatives when PEEK'd back from memory. This is due to the QL treating words and long words as signed numbers.

CROSS-REFERENCE

FILLMEM_B, FILLMEM_W.

2.22 FLUSH_CHANNEL

Syntax	FLUSH_CHANNEL #channel
Location	DJToolkit 1.17

This procedure makes sure that all data written to the given channel number has been 'flushed' out to the appropriate device. This means that if a power cut occurs, then no data will be lost.

EXAMPLE

```
1000 DEFine PROCedure SaveSettings
1010   OPEN_OVER #3, "flp1_settings.cfg"
1020   FOR x = 1 to 100
1030     PRINT #3, Setting$(x), Value$(x)
1040   END FOR x
1050   FLUSH_CHANNEL #3
1060   CLOSE #3
1070 END DEFine
```

2.23 GET_BYTE

Syntax	byte = GET_BYTE(#channel)
Location	DJToolkit 1.17

Reads one character from the file attached to the channel number given and returns it as a value between 0 and 255. This is equivalent to CODE(INKEY$(#channel)).

BEWARE, PUT_BYTE can put negative values to file, for example -1 is put as 255, GET_BYTE will return 255 instead of -1. Any negative numbers returned are always error codes.

EXAMPLE

```
c = GET_BYTE(#3)
```

CROSS-REFERENCE

GET_FLOAT, GET_LONG, GET_STRING, GET_WORD.

2.24 GET_FLOAT

Syntax	float = GET_FLOAT(#channel)
Location	DJToolkit 1.17

Reads 6 bytes from the file and returns them as a floating point value.

BEWARE, if any errors occur, the value returned will be a negative QDOS error code. As GET_FLOAT does return negative values, it is difficult to determine whether that returned value is an error code or not. If the returned value is -10, for example, it could actually mean End Of File, this is about the only error code that can be (relatively) safely tested for.

EXAMPLE

```
fp = GET_FLOAT(#3)
```

CROSS-REFERENCE

GET_BYTE, GET_LONG, GET_STRING, GET_WORD.

2.25 GET_LONG

Syntax	long = GET_LONG(#channel)
Location	DJToolkit 1.17

Read the next 4 bytes from the file and return them as a number between 0 and 2^32 -1 (4,294,967,295 or HEX FFFFFFFF unsigned).

BEWARE, the same problem with negatives & error codes applies here as well as GET_FLOAT.

EXAMPLE

```
lv = GET_LONG(#3)
```

CROSS-REFERENCE

GET_BYTE, GET_FLOAT, GET_STRING, GET_WORD.

2.26 GET_STRING

Syntax	a$ = GET_STRING(#channel)
Location	DJToolkit 1.17

Read the next 2 bytes from the file and assuming them to be a QDOS string's length, read that many characters into a$. The two bytes holding the string's length are NOT returned in a$, only the data bytes.

The subtle difference between this function and FETCH_BYTES is that this one finds out how many bytes to return from the channel given, FETCH_BYTES needs to be told how many to return by the user. GET_STRING is the same as:

```
FETCH_BYTES(#channel, GET_WORD(#channel))
```

WARNING - JM and AH ROMS will give a 'Buffer overflow' error if the length of the returned string is more than 128 bytes. This is a fault in QDOS, not DJToolkit. The demos file, supplied with DJToolkit, has a 'fix' for this problem.

EXAMPLE

```
b$ = GET_STRING(#3)
```

CROSS-REFERENCE

GET_BYTE, GET_FLOAT, GET_LONG, GET_WORD, FETCH_BYTES.

2.27 GET_WORD

Syntax	word = GET_WORD(#channel)
Location	DJToolkit 1.17

The next two bytes are read from the appropriate file and returned as an integer value. This is equivalent to CODE(INKEY$(#channel)) * 256 + CODE(INKEY$(#channel)). See the caution above for GET_BYTE as it applies here as well. Any negative numbers returned will always be an error code.

EXAMPLE

```
w = GET_WORD(#3)
```

CROSS-REFERENCE

GET_BYTE, GET_FLOAT, GET_LONG, GET_STRING.

2.28 KBYTES_FREE

Syntax	memory = KBYTES_FREE
Location	DJToolkit 1.17

The amount of memory considered by QDOS to be free is returned rounded down to the nearest kilo byte. See also BYTES_FREE if you need the answer in bytes. The value in KBYTES_FREE may not be equal to BYTES_FREE/1024 as the value returned by KBYTES_FREE has been rounded down.

EXAMPLE

```
kb_available = KBYTES_FREE
```

CROSS-REFERENCE

BYTES_FREE.

2.29 LEVEL2

Syntax	present = LEVEL2(#channel)
Location	DJToolkit 1.17

If the device that has the given channel opened to it has the level 2 drivers, then present will be set to 1, otherwise it will be set to 0. The level 2 drivers allow such things as sub_directories to be used, when a DIR is done on one of these devices, sub-directories show up as a filename with '->' at the end of the name. Gold Cards and later models of Trump cards have level 2 drivers. Microdrives don't.

EXAMPLE

```
2500 DEFine PROCedure MAKE_DIRECTORY
2510   LOCal d$, t$, l2_ok, ch
2520   INPUT 'Enter drive names :';d$
2530   IF d$(LEN(d$)) <> '_' THEN d$ = d$ & '_': END IF
2540   PRINT 'Please wait, checking ...'
2550   ch = DJ_OPEN_OVER (d$ & CHR$(0) & CHR$(0))
2560   IF ch < 0: PRINT 'Cannot open file on ' & d$ & ', error: ' & ch: RETurn
2570   l2_ok = LEVEL2(#ch)
2580   CLOSE #ch
2590   DELETE d$ & CHR$(0) & CHR$(0)
2600   IF l2_ok
2610     INPUT 'Enter directory name please : ';t$
2620     MAKE_DIR d$ & t$
2630   ELSE
2640     PRINT 'Sorry, no level 2 drivers!'
2650   END IF
2660 END DEFine MAKE_DIRECTORY
```

2.30 MAX_CON

Syntax	error = MAX_CON(#channel%, x%, y%, xo%, yo%)
Location	DJToolkit 1.17

If the given channel is a 'CON_' channel, this function will return a zero in the variable 'error'. The integer variables, 'x%', 'y%', 'xo%' and 'yo%' will be altered by the function, to return the maximum size that the channel can be WINDOW'd to.

'x%' will be set to the maximum width, 'y%' to the maximum depth, 'xo%' and 'yo%' to the minimum x co-ordinate and y co-ordinate respectively.

For the technically minded reader, this function uses the IOP_FLIM routine in the pointer Environment code, if present. If it is not present, you should get the -15 error code returned. (BAD PARAMETER).

EXAMPLE

```
7080 DEFine PROCedure SCREEN_SIZES
7090   LOCal w%,h%,x%,y%,fer
7100   REMark how to work out maximum size of windows using iop.flim
7110   REMark using MAX_CON on primary channel returns screen size
7120   REMark secondaries return maximum sizes within outline where
7130   REMark pointer environment is used.
7140   w% = 512 : REMark width of standard QL screen
7150   h% = 256 : REMark height of standard QL screen
7160   x% = 0
7170   y% = 0
7180   :
7190   fer = MAX_CON(#0,w%,h%,x%,y%) : REMark primary for basic
7200   IF fer < 0 : PRINT #0,'Error ';fer : RETurn
7210   PRINT'#0 : ';w%;',';h%;',';x%;',';y%
7220   :
7230   fer = MAX_CON(#1,w%,h%,x%,y%) : REMark primary for basic
7240   IF fer < 0 : PRINT #0,'Error ';fer : RETurn
7250   PRINT'#1 : ';w%;',';h%;',';x%;',';y%
7260   :
7270   fer = MAX_CON(#2,w%,h%,x%,y%) : REMark primary for basic
7280   IF fer < 0 : PRINT #0,'Error ';fer : RETurn
7290   PRINT'#2 : ';w%;',';h%;',';x%;',';y%
7300 END DEFine SCREEN_SIZES
```

2.31 MAX_DEVS

Syntax	how_many = MAX_DEVS
Location	DJToolkit 1.17

This function returns the number of installed directory device drivers in your QL. It can be used to DIMension a string array to hold the device names as follows:

```
1000 REMark Count directory devices
1010 :
1020 how_many = MAX_DEVS
1030 :
1040 REMark Set up array
1050 :
1060 DIM device$(how_many, 10)
1070 :
1080 REMark Now get device names
1090 addr = 0
1100 FOR devs = 1 to how_many
1110   device$(devs) = DEV_NAME(addr)
1120   IF addr = 0 THEN EXIT devs: END IF
1130 END FOR devs
```

CROSS-REFERENCE

DEV_NAME.

2.32 MOVE_MEM

Syntax	MOVE_MEM destination, length
Location	DJToolkit 1.17

This procedure will copy the appropriate number of bytes from the given source address to the destination address. If there is an overlap in the addresses, then the procedure will notice and take the appropriate action to avoid corrupting the data being moved. Most moves will take place from source to destination, but in the event of an overlap, the move will be from (source + length -1) to (destination + length -1).

This procedure tries to do the moving as fast as possible and checks the addresses passed as parameters to see how it will do this as follows :-

- If both addresses are odd, move one byte, increase the source & destination addresses by 1 and drop in to treat them as if both are even, which they now are!
- If both addresses are even, calculate the number of long word moves (4 bytes at a time) that are to be done and do them. Now calculate how many single bytes need to be moved (zero to 3 only) and do them.
- If one address is odd and the other is even the move can only be done one byte at a time, this is quite a lot slower than if long words can be moved.

The calculations to determine which form of move to be done adds a certain overhead to the function and this can be the slowest part of a memory move that is quite small.

EXAMPLE

```
MOVE_MEM SCREEN_BASE(#0), SaveScreen_Addr, 32 \* 1024
```

2.33 MOVE_POSITION

Syntax	MOVE_POSITION #channel, relative_position
Location	DJToolkit 1.17

This is a similar procedure to ABS_POSITION, but the file pointer is set to a position relative to the current one. The direction given can be positive to move forward in the file, or negative to move backwards. The channel must of course be opened to a file on a directory device. If the position given would take you back to before the start of the file, the position is left at the start, position 0. If the move would take you past the end of file, the file is left at end of file.

After a MOVE_POSITION command, the next access to the given channel, whether read or write, will take place from the new position.

EXAMPLE

```
MOVE_POSITION #3, 0
```

moves the current file pointer on channel 3 to the start of the file.

```
MOVE_POSITION #3, 6e6
```

moves the current file pointer on channel 3 to the end of the file.

CROSS-REFERENCE

ABS_POSITION.

2.34 PEEK_FLOAT

Syntax	value = PEEK_FLOAT(address)
Location	DJToolkit 1.17

This function returns the floating point value represented by the 6 bytes stored at the given address. BEWARE, although this function cannot detect any errors, if the 6 bytes stored at 'address' are not a proper floating point value, the QL can crash. The crash is caused by QDOS and not by PEEK_FLOAT. This function should be used to retrieve values put there by POKE_FLOAT mentioned above.

EXAMPLE

```
1000 addr = RESERVE_HEAP(6)
1010 IF addr < 0 THEN
1020    PRINT "OUT OF MEMORY"
1030    STOP
1040 END IF
1050 POKE_FLOAT addr, PI
1060 myPI = PEEK_FLOAT(addr)
1070 IF myPI <> PI THEN
1080    PRINT "Something went horribly wrong!"
1090    PRINT "PI = " & PI & ", myPI = " & myPI
1100 END IF
```

CROSS-REFERENCE

POKE_STRING, PEEK_STRING, POKE_FLOAT.

2.35 PEEK_STRING

Syntax	a$ = PEEK_STRING(address, length)
Location	DJToolkit 1.17

The characters in memory at the given address are returned to a$. The address may be odd or even as no word for the length is used, the length of the returned string is given by the length parameter.

EXAMPLE The following set of functions return the Toolkit 2 default devices:

```
1000 DEFine FuNction TK2_DATA$
1010   RETurn TK2_DEFAULT$(176)
1020 END DEFine TK2_DATA$
1030 :
1040 DEFine FuNction TK2_PROG$
1050   RETurn TK2_DEFAULT$(172)
1060 END DEFine TK2_PROG$
1070 :
1080 DEFine FuNction TK2_DEST$
1090   RETurn TK2_DEFAULT$(180)
1100 END DEFine TK2_DEST$
1110 :
1120 :
1200 DEFine FuNction TK2_DEFAULT$(offset)
1210   LOCal address
1220   IF offset <> 172 AND offset <> 176 AND offset <> 180 THEN
1230      PRINT "TK2_DEAFULT$: Invalid Offset: " & offset
1240      RETurn ''
1250   END IF
1260   address = PEEK_L (SYSTEM_VARIABLES + offset)
1270   IF address = 0 THEN
1280     RETurn ''
1290   ELSE
1300     REMark this is a pointer to the appropriate TK2 default
1310     RETurn PEEK_STRING(address+2, PEEK_W(address))
1320   END IF
1330 END DEFine TK2_DEFAULT$
```

CROSS-REFERENCE

POKE_STRING, PEEK_FLOAT, POKE_FLOAT.

2.36 POKE_FLOAT

Syntax	POKE_FLOAT address, value
Location	DJToolkit 1.17

This procedure will poke the 6 bytes that the QL uses to represent a floating point variable into memory at the given address. The address can be odd or even as the procedure can cope either way.

EXAMPLE

```
1000 Address = RESERVE_HEAP(6)
1010 IF Address < 0 THEN
1020    PRINT "ERROR " & Address & " Allocating heap space."
1030    STOP
1040 END IF
1050 POKE_FLOAT Address, 666.616
```

CROSS-REFERENCE

POKE_STRING, PEEK_STRING, PEEK_FLOAT.

2.37 POKE_STRING

Syntax	POKE_STRING address, string
Location	DJToolkit 1.17

This procedure simply stores the strings contents at the given address. Only the contents of the string are stored, the 2 bytes defining the length are not stored. The address may be odd or even.

If the second parameter given is a numeric one or simply a number, beware, QDOS will convert it to the format that would be seen if the number was PRINTed before storing it at the address. For example, 1 million would be '1E6' which is arithmetically the same, but characterwise, very different.

EXAMPLE

```
1000 Address = RESERVE_HEAP(60)
1010 IF Address < 0 THEN
1020    PRINT "ERROR " & Address & " Allocating heap space."
1030    STOP
1040 END IF
1050 POKE_STRING Address, "DJToolkit " & DJTK_VERS$
```

CROSS-REFERENCE

PEEK_STRING, PEEK_FLOAT, POKE_FLOAT.

2.38 PUT_BYTE

Syntax	PUT_BYTE #channel, byte
Location	DJToolkit 1.17

The given byte is sent to the channel. If a byte value larger than 255 is given, only the lowest 8 bits of the value are sent. The byte value written to the channel will always be between 0 and 255 even if a negative value is supplied. GET_BYTE returns all values as positive.

EXAMPLE

```
PUT_BYTE #3, 10
```

CROSS-REFERENCE

PUT_FLOAT, PUT_LONG, PUT_STRING, PUT_WORD.

2.39 PUT_FLOAT

Syntax	PUT_FLOAT #channel, byte
Location	DJToolkit 1.17

The given float value is converted to the internal QDOS format for floating point numbers and those 6 bytes are sent to the given channel number. The full range of QL numbers can be sent including all the negative values. GET_FLOAT will return negative values correctly (unless an error occurs).

EXAMPLE

```
PUT_FLOAT #3, PI
```

CROSS-REFERENCE

PUT_BYTE, PUT_LONG, PUT_STRING, PUT_WORD.

2.40 PUT_LONG

Syntax	PUT_LONG #channel, byte
Location	DJToolkit 1.17

The long value given is sent as a sequence of four bytes to the channel. Negative values can be put and these will be returned correctly by GET_LONG unless any errors occur.

EXAMPLE

```
PUT_LONG #3, 1234567890
```

CROSS-REFERENCE

PUT_BYTE, PUT_FLOAT, PUT_STRING, PUT_WORD.

2.41 PUT_STRING

Syntax	PUT_STRING #channel, string
Location	DJToolkit 1.17

The string parameter is sent to the appropriate channel as a two byte word giving the length of the data then the characters of the data. If you send a string of zero length, LET A$ = “” for example, then only two bytes will be written to the file. See POKE_STRING for a description of what will happen if you supply a number or a numeric variable as the second parameter. As with all QL strings, the maximum length of a string is 32kbytes.

EXAMPLE

```
PUT_STRING #3, "This is a string of data"
```

CROSS-REFERENCE

PUT_BYTE, PUT_FLOAT, PUT_LONG, PUT_WORD.

2.42 PUT_WORD

Syntax	PUT_WORD #channel, word
Location	DJToolkit 1.17

The supplied word is written to the appropriate channel as a sequence of two bytes. If the word value supplied is bigger than 65,535 then only the lower 16 bits of the value will be used. Negative values will be returned by GET_WORD as positive.

EXAMPLE

```
PUT_WORD #3, 65535
```

CROSS-REFERENCE

PUT_BYTE, PUT_FLOAT, PUT_LONG, PUT_STRING.

2.43 QPTR

Syntax	PE_Found = QPTR(#channel)
Location	DJToolkit 1.17

This function returns 1 if the Pointer Environment is loaded or 0 if not. The channel must be a SCR_ or CON_ channel, if not, the result will be 0. If a silly value is given then a QDOS error code will be returned instead.

EXAMPLE

```
PRINT QPTR(#0)
```

will print 1 of the PE is loaded or zero otherwise.

2.44 READ_HEADER

Syntax	error = READ_HEADER(#channel, buffer)
Location	DJToolkit 1.17

The file that is opened on the given channel has its header data read into memory starting at the given address (buffer). The buffer address must have been reserved using RESERVE_HEAP, or some similar command.

The buffer must be at least 64 bytes long or unpredictable results will occur. The function will read the header but any memory beyond the end of the buffer will be overwritten if the buffer is too short. After a successful call to this function, the contents of the buffer will be as follows :

Address	Value	Size
Buffer + 0	File length	4 bytes long (see FILE_LENGTH)
Buffer + 4	File access	1 byte long - currently zero
Buffer + 5	File type	1 byte long (see FILE_TYPE)
Buffer + 6	File dataspace	4 bytes long (see FILE_DATASPACE)
Buffer + 10	Unused	4 bytes long
Buffer + 14	Name length	2 bytes long, size of filename
Buffer + 16	Filename	36 bytes long

Directory devices also have the following additional data :

Address	Value	Size
Buffer + 52	Update date	4 bytes long (see FILE_UPDATE)
Buffer + 56	Reference date	4 bytes long - see below
Buffer + 60	Backup date	4 bytes long (see FILE_BACKUP)

Miracle Systems hard disc's users and level 2 users will find the files version number stored as the the 2 bytes starting at buffer + 56, the remaining 2 bytes of the reference date seem to be hex 094A or decimal 2378 which has no apparent meaning, this of course may change at some point!

This function returns an error code if something went wrong while attempting to read the file header or zero if everything went ok. It can be used as a more efficient method of finding out the details for a particular file rather than calling all the various FILE_XXXX functions. Each of these call the READ_HEADER routine.

To extract data, use PEEK for byte values, PEEK_W for the filename length and version number (if level 2 drivers are present, see LEVEL2), or PEEK_L to extract 4 byte data items.

The filename can be extracted from the buffer by something like:

```
f$ = PEEK_STRING(buffer + 16, PEEK_W(buffer + 14)).
```

EXAMPLE The following example allows you to change the current dataspace requirements for an EXECutable file:

```
6445 DEFine PROCedure ALTER_DATASPACE
6450   LOCal base, loop, f$, ft, nv
6455   base = RESERVE_HEAP (64)
6460   IF base < 0 THEN
6465     PRINT "ERROR: " & base & ", reserving heap space."
6470     RETurn
6475   END IF
6480   REPeat loop
6485     INPUT'Enter filename:';f$
6490     IF f$ = '' THEN EXIT loop
6495     ft = FILE_TYPE(f$)
6500     IF ft < 0 THEN
6465       PRINT "ERROR: " & ft & ", reading file type for " & f$ & "."
6510     END IF
6515     IF ft <> 1 THEN
6520       PRINT f$ & 'is not an executable file!'
6525       NEXT loop
6530     END IF
6535     PRINT 'Current dataspace is:'; FILE_DATASPACE(f$)
6540     INPUT 'Enter new value:'; nv
6545     OPEN #3,f$ : fer = READ_HEADER (#3,base)
6550     IF fer < 0 : CLOSE #3 : PRINT "READ_HEADER error: " & fer : NEXT loop
6555     POKE_L base + 6,nv
6560     fer = SET_HEADER(#3,base)
6565     IF fer < 0 : PRINT "SET_HEADER error: " & fer
6570     CLOSE #3
6575   END REPeat loop
6580   RELEASE_HEAP base
6585 END DEFine ALTER_DATASPACE
```

CROSS-REFERENCE

SET_HEADER, FILE_LENGTH, FILE_TYPE, FILE_DATASPACE, FILE_UPDATE, FILE_BACKUP.

2.45 RELEASE_HEAP

Syntax	RELEASE_HEAP address
Location	DJToolkit 1.17

The address given is assumed to be the address of a chunk of common heap as allocated earlier in the program by RESERVE_HEAP. In order to avoid crashing the QL when an invalid address is given, RELEASE_HEAP checks first that there is a flag at address-4 and if so, clears the flag and returns the memory back to the system. If the flag is not there, or if the area has already been released, then a bad parameter error will occur.

It is more efficient to RELEASE_HEAP in the opposite order to that in which it was reserved and will help to avoid heap fragmentation.

CROSS-REFERENCE

See RESERVE_HEAP, below, for an example of use.

2.46 RESERVE_HEAP

Syntax	buffer = RESERVE_HEAP(length)
Location	DJToolkit 1.17

This function obtains a chunk of memory for your program to use, the starting address is returned as the result of the call. Note that the function will ask for 4 bytes more than you require, these are used to store a flag so that calls to READ_HEADER do not crash the system by attempting to deallocate invalid areas of memory. If you call this function, the returned address is the first byte that your program can use.

EXAMPLE

The following example shows how this function can be used to reserve a buffer for READ_HEADER, described elsewhere.

```
1000 buffer = RESERVE_HEAP(64)
1010 IF buffer < 0
1020    PRINT 'ERROR allocating buffer, ' & buffer
1030    STOP
1040 END IF
1050 error = READ_HEADER(#3, buffer)

.....do something with buffer contents here

2040 REMark Finished with buffer
2050 RELEASE_HEAP buffer
```

CROSS-REFERENCE

RELEASE_HEAP, ALCHP, RECHP, ALLOCATE.

2.47 SCREEN_BASE

Syntax	screen = SCREEN_BASE(#channel)
Location	DJToolkit 1.17

This function is handy for Minerva users, who have 2 screens to play with. The function returns the address of the start of the screen memory for the appropriate channel.

If the returned address is negative, consider it to be a QDOS error code. (-6 means channel not open & -15 means not a SCR_ or CON_ channel.)

SCREEN_BASE allows you to write programs that need not make guesses about the whereabouts of the screen memory, or assume that if VER$ gives a certain result, that a Minerva ROM is being used, this may not always be the case. Regardless of the ROM in use, this function will always return the screen address for the given channel.

EXAMPLE

```
PRINT HEX$(SCREEN_BASE(#0), 24)
```

2.48 SCREEN_MODE

Syntax	current_mode = SCREEN_MODE
Location	DJToolkit 1.17

This function can help in your programs where you need to be in a specific mode. If you call this function you can find out if a mode change needs to be made or not. As the MODE call changes the mode for every program running in the QL, use this function before setting the appropriate mode.

The value returned can be 4 or 8 for normal QLs, 2 for Atari ST/QL Extended mode 4 or any other value deemed appropriate by the hardware being used. Never assume that your programs will only be run on a QL!

EXAMPLE

```
1000 REMark Requires MODE 4 for best results so ...
1010 IF SCREEN_MODE <> 4
1020    MODE 4
1030 END IF
1040 :
1050 REMark Rest of program ....
```

CROSS-REFERENCE

MODE.

2.49 SEARCH_C

Syntax	address = SEARCH_C(start, length, what_for$)
Location	DJToolkit 1.17

See SEARCH_I for details.

CROSS-REFERENCE

SEARCH_I.

2.50 SEARCH_I

Syntax	address = SEARCH_I(start, length, what_for$)
Location	DJToolkit 1.17

This function, and SEARCH_C above, search through memory looking for the given string. SEARCH_C searches for an EXACT match whereas SEARCH_I ignores the difference between lower & UPPER case letters.

If the address returned is zero, the string was not found, otherwise it is the address where the first character of what_for$ was found, or negative for any errors that may have occurred.

If the string being searched for is empty ("") then zero will be returned, if the length of the buffer is negative or 0, you will get a 'bad parameter' error (-15). The address is considered to be unsigned, so negative addresses will be considered to be very large positive addresses, this allows for any future enhancements which will allow the QL to use a lot more memory than it does now!

EXAMPLE

```
1000 PRINT SEARCH_C(0, 48 * 1024, 'sinclair')
1010 PRINT SEARCH_I(0, 48 * 1024, 'sinclair')
1020 PRINT
1030 PRINT SEARCH_C(0, 48 * 1024, 'Sinclair')
1040 PRINT SEARCH_I(0, 48 * 1024, 'Sinclair')
```

The above fragment, on my Gold Card JS QL, prints:

```
0
47314

47314
47314
```

Looking into the ROM at that address using

```
PEEK_STRING(47314, 21)
```

gives:

```
Sinclair Research Ltd
```

which is part of the copyright notice that comes up when you switch on your QL. The reason for zero in line 1000 is because the 's' is lower case, case is significant and the ROM has a capital 'S', so the text was not found in the ROM.

CROSS-REFERENCE

SEARCH_C.

2.51 SET_HEADER

Syntax	error = SET_HEADER(#channel, buffer)
Location	DJToolkit 1.17

This function returns the error code that occurred when trying to set the header of the file on the given channel, to the contents of the 64 byte buffer stored at the given address. If the result is zero then you can assume that it worked ok, otherwise the result will be a negative QDOS error code. On normal QLs, the three dates at the end of a file header cannot be set.

EXAMPLE

See the example for READ_HEADER.

CROSS-REFERENCE

READ_HEADER.

2.52 SET_XINC

Syntax	SET_XINC #channel, increment
Location	DJToolkit 1.17

See SET_YINC, below, for details.

2.53 SET_YINC

Syntax	SET_YINC #channel, increment
Location	DJToolkit 1.17

These two functions change the spacing between characters horozontally, SET_XINC, or vertically, SET_YINC. This allows slightly more information to be displayed on the screen. SET_XINC allows adjacent characters on a line of the screen to be positioned closer or further apart as desired. SET_YINC varies the spacing between the current line of characters and the next.

By choosing silly values, you can have a real messy screen, but try experimenting with OVER as well to see what happens. Use of the MODE or CSIZE commands in SuperBasic will overwrite your new values.

EXAMPLE

```
SET_XINC #2, 22
SET_YINC #2, 16
PRINT #2, "This is a line of text"
PRINT #2, "This is another line of text"
PRINT #2, "This is yet another!"
```

CROSS-REFERENCE

SET_XINC.

2.54 SYSTEM_VARIABLES

Syntax	sys_vars = SYSTEM_VARIABLES
Location	DJToolkit 1.17

This function returns the current address of the QL's system variables. For most purposes, this will be hex 28000, decimal 163840, but Minerva users will probably get a different value due to the double screen. *Do not* assume that all QLs, current or future, will have their system variables at a fixed point in memory, this need not be the case.

EXAMPLE

```
PRINT SYSTEM_VARIABLES
```

2.55 USE_FONT

Syntax	USE_FONT #channel, font1_address, font2_address
Location	DJToolkit 1.17

This is a procedure that will allow your programs to use a character set that is different from the standard QL fonts. The following example will suffice as a full description.

EXAMPLE

```
1000 REMark Change the character set for channel #1
1010 :
1020 REMark Reserve space for the font file
1030 size = FILE_LENGTH('flp1_font_file')
1040 IF size < 0
1050    PRINT 'Font file error ' & size
1060    STOP
1070 END IF
```

(continues on next page)

(continued from previous page)

```
1080 :
1090 REMark Reserve space to load font into
1200 font_address = RESERVE_HEAP(size)
1210 IF font_address < 0
1220    PRINT 'Heap error ' & font_address
1230    STOP
1240 END IF
1250 :
1260 REMark Load the font
1270 LBYTES flp1_font_file, font_address
1280 :
1290 REMark Now use the new font
1300 USE_FONT #1, font_address, 0

.......Rest of program

9000 REMark Reset channel #1 fonts
9010 USE_FONT #1, 0, 0
9020 :
9030 REMark Release the storage space
9040 RELEASE_HEAP font_address
```

2.56 WHERE_FONTS

Syntax	address = WHERE_FONTS(#channel, 1_or_2)
Location	DJToolkit 1.17

This function returns a value that corresponds to the address of the fonts in use on the specified channel. The second parameter must be 1 for the first font address or 2 for the second, there are two fonts used on each channel. If the result is negative then it will be a normal QDOS error code. The channel must be a CON_ or a SCR_ channel to avoid errors.

EXAMPLE

The following example will report on the two fonts used in any given channel, and will display the character set defined in that font:

```
4480 DEFine PROCedure REPORT_ON_FONTS (channel)
4485   LOCal address, lowest, number, b
4490   REMark show details of channel's fonts
4495   CLS
4500   FOR a = 1,2
4505     address = WHERE_FONTS(#channel, a)
4510     lowest = PEEK(address)
4515     number = PEEK(address + 1)
4520     PRINT '#'; channel; ' font '; a; ' at address '; address
4525     PRINT 'Lowest character code = '; lowest
4530     PRINT 'Number of characters  = '; number + 1
4535     REMark print all but default characters
```

(continues on next page)

(continued from previous page)

```
4540     PRINT : REMark blank line
4545     FOR b = lowest + 1 TO lowest + number :PRINT CHR$(b);
4550     PRINT \\\ : REMark 2 blank lines
4555   END FOR a
4560 END DEFine REPORT_ON_FONTS
```

CHAPTER

THREE

DJTOOLKIT UPDATES

3.1 UPDATES TO DJTOOLKIT V1.10

A few new commands have been added, such as `QPTR`, the font handling commands and `DISPLAY_WIDTH` to check the display size. All these are now documented in the manual. Norman has, however, given me an embarrassing list of typing errors I made when I transferred the manual to prepare it in Text87, these will be corrected in the next issue of the manual, as they do not affect the accuracy of the manual, only offend those who dislike typos!

Dilwyn Jones

3.2 UPDATES TO DJTOOLKIT V1.11 (18/5/1993)

Despite the fact that the AH and JM ROM presented the `DISPLAY_WIDTH` command with problems, this has now been solved in two ways. Firstly, a demo routine (`DISPLAY_WIDTH_JM`) checks for an offending ROM version and returns a default value to prevent the problem. Secondly, Norman has patched the `DISPLAY_WIDTH` function to include a check for AH and JM ROMs (or rather the versions of QDOS with those versions of BASIC, to be accurate) to prevent the problem.

I have fixed an embarrassing number of faults in the demo files. Nobody actually complained about these, I just noticed them myself. I've also added a few more demo routines such as a fast copier and dates utility - most of the new routines are at the end of the file.

Note that the DEMOS_sav version can give a list of missing extensions when loaded with QLOAD if the QLiberator extensions are not present. For the most part, they will still run OK, since a check is made for their presence (e.g. the `CURSOR_ENABLE` routine. I have also updated the DEMOS_doc documentation file to include details of the new routines.

I made a few changes to the demo routines to take account of the fact that Norman reprogrammed some functions in V1.10 at the suggestion of Ralf Rekoendt of Germany, to return negative error codes rather than stopping with an error message.

The first commercial program using this toolkit has been launched - DJC's CONVERT-PCX graphics conversion utility for clipart ported from the PC (used in conjunction with Discover, plug, plug). CONVERT-PCX costs just £10.00.

Dilwyn Jones

3.3 UPDATES TO DJTOOLKIT V1.12 (15/6/1993)

Due to the fact that I tried to make things a bit quicker in the MOVE_MEM command, I ended up using an algorithm that allowed an easy (!) way to figure out which direction the memory needed to be moved in order to avoid overlap problems. As it turned out, the algorithm was wrong! This caused a slight problem in that some of the first 6 bytes were not moved when moving from an even address to an even address with no overlap, the program did it as if there was an overlap and missed a few bytes out of the move.

MOVE_MEM is now fixed, bigger and for small memory moves, it spends most of its time figuring out how to actually do it. Large memory moves, say saving and restoring screens, should now work correctly and quickly.

Norman Dunbar

3.4 UPDATES TO DJTOOLKIT V1.13 (19/07/1993)

I use a Gold Card for all my work, it is quick and the vast amount of memory allows me to run lots of utility programs together with QPAC 2 etc. So what, I hear you think. Well it seems that a small bug has existed in the FILE_POSITION function which causes a normal 128K QL to crash with a fancy screen display which fills the screen from the bottom to the top - interesting. A Trump Card (old version, no level 2 drivers) just gives up quietly and gives no indication of its troubles. A Gold Card works !!!!!!!

It seems that the system call FS_POSRE (TRAP #3, D0 = $43) actually destroys register A1 which is of course the maths stack pointer. This has now been fixed for All QLs, not just the Gold Card users. Funny, no one has complained about it up until yesterday when Dilwyn Phoned !

Having tested the new version (1.13) on a Trump Card equipped QL, I fired up the trusty Gold Card and traced the execution of FILE_POSITION using QMON 2. Lo and behold, the A1 register is PRESERVED by the system call FS_POSRE (and FS_POSAB?) when running with a Gold Card, mystery solved, but why is it preserved ?

Norman Dunbar

3.5 UPDATES TO VERSION 1.13 PART 2 (22/10/1993)

Dilwyn contacted me to say that a customer was having problems with some of Dilwyn's demo routines. I have had a look at these (Dilwyn has more than enough problems with Page Designer 3 !!!!) and found that most of them were caused by not having enough LOCal statements.

Some of Dilwyn's routines use the same names, but some are ARRAYs and others are not. If FASTCOPY has been called, LOAD_A_FONT refuses to work due to the variable 'fl' being a DIM``med array in ``FASTCOPY (for file length). I have added a few more LOCals to every routine that needs them. Problem now solved.

You should be aware that on some QLs, JS in particular, there is a bug that occurs when a program routine is executed. If the routine (PROC or FN) has a total of 10 or more parameters and locals then the SuperBasic listing gets trashed in a big way. The program will probably fall over with BAD NAME or something.

When testing the amended demo routines, I of course had forgotten about this bug and managed to remove the SuperBasic job from the QL all together (who said it couldn't be done ?) Using the JOBS/RJOB utilities in QPAC 2 did not even show SuperBasic as a job any more !!!!!

Luckily I always (?) save changes before running them, just in case. One quick reset later and all was well again. Enough waffle, hopefully the demos are now ok. I have put a warning in the demos file at the start and REMark``ed out extra ``LOCal lines but there shouldn't be any more name clashes - famous last words.

Norman Dunbar

3.6 UPDATES TO DJTOOLKIT V1.14 (12/06/1994)

At Dilwyn's request, some additional routines have been added to the toolkit. These being some file opening functions that return an error code or the channel id. I also added a couple of extra handy routines of my own, just for fun. The new routines are:

```
POKE_FLOAT address,value (PROC)
PEEK_FLOAT(address)      (FN returning float)
MAX_DEVS                 (FN returning integer)
DJ_OPEN('filename')      (FN returning integer)
DJ_OPEN_IN('filename')   (ditto)
DJ_OPEN_NEW('filename')  (ditto)
DJ_OPEN_OVER('filename') (ditto)
DJ_OPEN_DIR('filename')  (ditto)
MAX_CON(#ch, x,y,xo,yo)  (FN returning int + altered params)
```

The file opening procedures are very similar to Simon Goodwin's recent article in the DIY Toolkit series in QL WORLD magazine (Vol 2, issue 8 which was marked Vol 2 issue 7 just to be confusing). The article was about his routines called `ANYOPEN%`. Simon's article came in very handy as I had known about the ability to extend the SuperBasic channel table, but had not quite figured out how to fill it in afterwards. Thanks Simon.

Norman Dunbar

3.7 UPDATES TO DJTOOLKIT V1.15 (16/06/1994)

So, I thought it was complete, but Dilwyn left a message on my machine, which went something like "what happened to the fill memory commands then ?" - oops, I forgot !

This version, 1.15, now contains the additional procedures

```
FILLMEM_B start_address, how_many, value
FILLMEM_W start_address, how_many, value
FILLMEM_L start_address, how_many, value
```

and that is about what they do !

Norman Dunbar

3.8 UPDATES TO DJTOOLKIT V1.16 (27/02/2013)

Change to `GET_STRING` function so as not to cause End Of File error on SMSQmulator if a null string is the last item fetched from the end of a file.

3.9 UPDATES TO DJTOOLKIT V1.17 (26/10/2025)

A bug in MOVE_MEM failed to copy the final byte, the one at the source address if:

> The source address was less than the destination address; AND The destination address was less than source + size AND The two addresses were both EVEN or both ODD.

This was "fixed" by simply doing a byte by byte move in those circumstances. I will get to fixing it at some point soon! (For my usual definitions of "soon"!)

Norman Dunbar