WHY YOU SHOULDN'T PEEK AND POKE FONT ADDRESSES!

My Fun With Fonts articles in QL Today sought to discourage you from using PEEK and POKE statements to read font addresses and set new ones, but didn't really explain why. I'm often asked "what is the correct POKE" to set a new character set, but in truth it's the wrong question to ask, since data structures can and do move around a lot on the QL – never the same place twice is perhaps putting it too strongly, but there's an element of truth in that statement.

On a QL, data structures such as system variables and channel definition blocks may be in different places in memory. It can be hard to find them reliably from BASIC on modern QL systems – the operating system makes it easy from machine code, because to track them you need to follow pointers to blocks of data. What may have looked like a fixed position in memory on an unexpanded BBQL (Black Box QL) may move because of expanded memory, presence of disk interfaces which need some memory to themselves, use of non-standard screens such as Minerva dual screen mode, or enhanced resolution or high colour modes on emulators and SMSQ/E systems, and whether or not the pointer environment was present. If you were brought up on a diet of PEEKing and POKEing everything in sight on a ZX81 or Spectrum it may be tempting to do the same on a QL. But you shouldn't.

In very early programs running on unexpanded BBQL systems, you may come across BASIC programs which find the addresses of fonts by using something like LET font1 = PEEK_L(167722):LET font2=PEEK_L(167726). The font addresses hppened to be held at "fixed" addresses on the unexpanded BBQL. Later on, as hardware and software developed, the data moved to a different address. But those early programs often didn't get updated and caused many and repeated "why don't this work?" remarks! Such programs often poke new address values into those locations when defining a new character set. But if we dealing with anything other than an unexpanded BBQL, the 1677xx values may in fact point to totally random areas of memory. Not only might the program just not work (if you are lucky), the pokes may overwrite something important in memory and crash the QL.

The way to fix these programs where the redefined character set or new fonts or "user defined graphics" is to replace the POKE_L statements with extensions such as CHAR_USE from Toolkit 2, or any of the many free font setting extensions available. The first thing to do is work out which channel's font is being changed (you can usually do that by looking at PRINT statements which display the characters (e.g. if you see a PRINT #2,CHR$(128) printing a redefined CHR$ 128 from the second font it is using window channel #2 to display the character from #2's second or upper character set, so the font redefined is attached to #2. Therefore, if the code for redfining fonts is something like:

```
base = RESPR(font_length):LBYTES "new_font",base:POKE_L 123456,base
```

You'd replace it with something like:

```
base = RESPR(font_length):LBYTES "new_font",base:CHAR_USE #2,0,base
```

Each window on the QL can have its own character set, in fact two character sets (the first is the normal printable ones with codes up to 127 – the copyright symbol – and the second with the accented and special characters with higher codes.

To find the start addresses of these fonts, we need to understand a bit about the data structures.

The addresses of the fonts are stored in something called a channel definition block. These store details of sizes of windows, borders and attributes like colours and font addresses. Channel definition blocks are effectively sets of lists of information. Somewhere else in memory there is a pointer to these blocks. And somewhere else (in the system variables) there's another pointer to the list of pointers to those! Now you can start to see why it'd be damn hard to write a POKE that'll reliably work for all possible systems!

There are two factors which make it even harder:

1.BASIC channel numbers are not necessarily the same as operating system channel numbers. In practice when you first start BASIC and as long as you don't open and close window channels, BASIC channel #0 #1 and #2 will correspond to the first three operating system channel numbers. This won't necessarily be true for higher channel numbers or window channels opened in a Minerva multibasic or one of SBASIC's daughter jobs. To keep things simple, I'll assume we are using the usual #0 #1 and #2 as when Job 0 BASIC (the one which opens up when the QL is started or reset) is first started. For other windows, you'd need to find the channel ID and marry it up to the corresponding operating system channel.

2. Pointer environment (PE) is present, whether it be SMSQ/E or PTR_GEN etc with QDOS. The structure of the channel definition blocks for windows change when PE is present. The blocks then become "extended channel blocks" which have 48 bytes (or $30) extra bytes, which cause further movement of all addresses.

Oh dear, this is getting to sound hopelessly complicated before I even start to explain how we move on from a simple PEEK_L / POKE_L to something which works a bit more reliably across other systems.

Firstly, we need a fairly reliable way to look into the system variables, given that those can be in different addresses on different machines. On a standard BBQL with a Sinclair ROM those system variables will be at the address 163840 (just above the 32K video screen). But if we are using the second screen on a Minerva system or a high resolution display or high colour mode on a more modern emulator or QL compatible computer, the system variables may be shoved elsewhere if the screen needs more memory than the 32K on a BQL, for example.

Luckily, on systems where the system variables may move (Minerva and SMSQ/E), there is an extension to the VER$ function which makes it possible to locate the system variables.

```
sv = 163840 : REM default on a Sinclair QDOS ROM
v$ = VER$
IF v$ = 'JSL1' OR v$ = 'HBA' THEN sv = VER$(-2)
```

The first line assumes this is runnng on a Sinclair QDOS ROM such as version AH, JM, JS or MG where the system variables really do live at that address. The secnd line tests which version of BASIC we're running on, in case it's Minerva (version "JSL1") or SBASIC on SMSQ/E systems (version "HBA"), in which case we can use VER$(-2) to check where exactly the ssytem variables may have moved to on those systems. The IF statement then protects Sinclair ROMs, which don't know what VER$(-2) is, from stopping with an error if they try to execute VER$(-2). In other words, it only gets used on Minerva or SMSQ/E.

Incidentally, if you're wondering why assign VER$ to a variable rather than use it direct in the IF clause, like IF VER$ = 'JSL1' OR VER$ = 'HBA' … , if you read the ROM Bugs articles by people like

Simon Goodwin and Mark Knight, you'll see that there is a little issue with VER$ when used in that way which can cause errors elsewhere in a program by virtue of leaving stray bytes on a stack in certain circumstances when used in expressions.

So now 'sv' contains the address of the start of the system variables.

This isn't foolproof – it works on current systems, but only those identified by the JSL1 or HBA identifiers. It's not inconceivable that there may be future variants of the operating system not coevered by this example!

Now that we can find the system variables, we need to find the channels tables. There's a long word 120 bytes into the system variables which points to the base of the channel table. This is actually operating system channels, not BASIC ones, but luckily for channels 0, 1 and 2 of the main BASIC these happen to be the same, and to avoid overcomplication we won't go into that – it would involved working out which operating system channel ID corresponded to which BASIC channel number, and that's taking things too far at this stage. But you'd need to be aware of it when compiling font programs or using a daughter SBASIC jobor a Minerva multi-basic job.

The table pointed to is a list of long words which in turn point to a channel definition block for each channel.

```
LET chbase = PEEK_L(sv+120)
```

chbase points to the address of the start of the definition block for the first channel, which usually correspnds to BASIC's channel #0 (unless prevously closed and reopened, in which case it might not be the same as operating system channel 0). As this is a long word per channel, the pointer to the definition block for channel number 'chan' is fetched like this:

```
LET chhdr = PEEK_L(chbase + 4*chan)
```

So chhdr now points us to the address of the start of the header for our channel number.

At this stage, things start to get a little bit more complicated if pointer environment is present, since there is additional information in each definition block. In fact it consists of an extra 48 bytes to take account of.

If there is no pointer environment, the address of font numbers 1 and 2 for the channel is obtained by:

```
LET font1 = PEEK_L(chhdr+42)
LET font2 = PEEK_L(chhdr+46)
```

But if pointer environment is present, we need to add 48 bytes ($30 in hexadecimal) to the offset from the start of the block:

```
LET font1 = PEEK_L(chhdr+48+42)
LET font2 = PEEK_L(chhdr+48+46)
```

As there is no simple way of checking for the presence of pointer environment from BASIC without extensions, we start to get stuck on the notion of a standard set of PEEKs and POKEs which work uniformly across all systems. Which is why experienced programmers tell you to avoid them and use proper extensions such as CHAR_USE to tell the system where the new replacement fonts may be found:

CHAR_USE #channel,font1_address,font2_address

If either font1_address or font2_address have a value of zero, it just means use the font built into the ROM. The default font.

So, having gone through the steps of the process to understand how this all ties together and works, here is a way (which only works on systems without pointer environment) to check the address of the existing font being used for channels 0, 1 or 2.

```
LET font1_addr = PEEK_L(42+(PEEK_L(PEEK_L(sv+120)+(4*channel_number))))
LET font2_addr = PEEK_L(46+(PEEK_L(PEEK_L(sv+120)+(4*channel_number))))
```

If PE is present, you need to add 48 to the 42 and 46 in both expressions:

```
LET font1_addr = PEEK_L(48+42+(PEEK_L(PEEK_L(sv+120)+(4*channel_number))))
LET font2_addr = PEEK_L(48+46+(PEEK_L(PEEK_L(sv+120)+(4*channel_number))))
```

Even then, this only works for those three initial channels,and even then only if they have never been closed and reopened, or opened as those channel numbers in SBASIC daughter jobs or Minerva Multibasics! It then becomes a matter of finding the channel ID for the BASIC channel and matching that up with the perating system channel numbers to find the correct channel definition block, which adds more than a little complexity to an already complex example.

It's also possible in theory (though rare in practice) that the data blocks, including the system variables, might move in between BASIC commands if the operating system decides to do so, e.g. on some systems changing screen resolutions or colour depths so that the screen memory needs to be bigger might shove the system variables further up in memory! Yet more reasons to avoid PEEK and POK when it comes to fonts.

POKEing new font addresses is essentially the same operation, locating where the font addresses are stored with calculations like the above and using POKE_L to insert the new addresses.

For example, suppose we have loaded a new fount or character set into the common heap to an address we store in the variable ‘new_font’, and wish to assign it as font number 2 for a channel:

```
new_font_address = ALCHP(size of new font in bytes)
LBYTES newfont$,base
LET old_font_address = font2_addr
POKE_L 46+(PEEK_L(PEEK_L(sv+120)+(4*channel_number))),new_font_address
```

And to reset it later to the previous font:

```
POKE_L 46+(PEEK_L(PEEK_L(sv+120)+(4*channel_number))),old_font_address
```

(remember to add the 48 if pointer environment is present of course).

Finally, I’ve rearranged things into a few functions which let you locate the start of fonts using the techniques described above, and use the same technique to use POKE_L to set a new font for a given window channel’s font 1 or font 2. Remember it only works for BASIC channels 0 to 2 and you should alter the value of the variable pe to 48 if the system has pointer environment, or 0 if not. The function Sd_Font returns the address to use to PEEK_L or POKE_L the address of the font – sd_font is the name the QDOS and SMSQ/E technical manuals use to refer to the fonts entry in the Channel Definition Blocks. This is about as close as you'll get to working POKEs with fonts and even this is to totally discouraged in favour of using proper extensions such as CHAR_USE.

```
100 CLS
110 FOR ch = 0,1,2
120   PRINT Font_Address(#ch,1)
130   PRINT Font_Address(#ch,2)
140 END FOR ch
150 STOP
160 :
170 DEFine FuNction Sd_Font(chan,fnt1or2)
180   LOCal v$,sv,pe,Sd_Font
190   REMark get address is Channel Definition Block of font pointer
200   REMark ONLY WORKS FOR CHANNELS 0-2, and MUST adjust "pe"
210   REMark for whether pointer environment is present or not
220   :
230   sv = 163840 : REMark default on Sinclair ROM
240   v$ = VER$ : REMark check version
250   IF v$ = 'JSL1' OR v$ = 'HBA' THEN sv = VER$(-2)
260   :
270   pe = 48 : REMark 48=pointer environment present, 0=not present
280   :
290   RETurn PEEK_L(PEEK_L(sv+120)+4*chan)+pe+42+(4*(fnt1or2 = 2))
300 END DEFine Sd_Font
310 :
320 DEFine FuNction Font_Address (channel,font1or2)
330   RETurn PEEK_L(Sd_Font(channel,font1or2))
340 END DEFine Font_Address
350 :
360 DEFine PROCedure Set_New_Font (channel,font1or2,new_font_address)
370   POKE_L Sd_Font(channel,font1or2),new_font_address
380 END DEFine Set_New_Font
```

Hopefully, you can now see from the complexity that using an extenson like CHAR_USE is far, far easier! And understand that even these convoluted PEEK_L and POKE_L expressions should not be used with new fonts. Don't use these routines, they are only there to explain why coding like this should not be used on QL-type machines.

I'll now go and take a lie down in a darkened room after all that.