BOOT Programs

When our esteemed editor ran into some problems with his BOOT recently (the QL variety, not the footwear variety) a flurry of emails followed which made us realise that this is a subject often taken for granted by experienced users, while the less experienced ones may struggle with the topic, especially if new to the Pointer Environment (PE).

We also realised that there isn't that much by way of documentation out there on this particular subject.

Accordingly, I have donned my Helpline hat and have tried to produce a document out of that particular flurry of emails which should help those who are not as fluent in writing Boot programs for PE and Qpac2.

But first things first, let's quickly establish what we mean by terms like PE and Qpac2.

Pointer Environment, as its name implies, extends the QL's operating system, in particular the CON (console) drivers in such a way that programs can be controlled with a little on-screen pointer (usually an arrow) using either a mouse or the cursor arrow keys on the keyboard. The new console driver also allows you to have overlapping non-destructive windows – if you use CTRL-C to switch between several running programs, the system will ensure that the contents of the on-screen windows are saved and restored as you switch from one program or another, which makes for a much more comfortable system to use than the original command-line QL where everything had to be done manually. All this is handled by the **Pointer interface.**

You may have noticed I used both the term '**Pointer Environment**' and '**Pointer Interface**' above. Many people take them to mean the same thing, whereas in reality they do have separate and distinct meanings. The Pointer Interface is the part which extends the operating system, the technical bit if you like, and the term 'Pointer Environment' usually refers to the user's viewpoint of this extended system.

A third term, **Extended Environment**, refers to a set of three extensions to the machine, including the Pointer Interface, Window Manager and Hotkey System 2

The **Pointer Interface** handles the "interface" between the QL and the user – saving and restoring windows as you switch between programs, mouse routines, on-screen pointers and so on.

The **Window Manager** on the other hand is a set of routines which allow for similar and consistent windows across many types of programs. For example, the Window Manager can control how borders look, how the title strip across the top of a program might look and some standardised colour schemes for programs.

The third part of the system is known as the **Hotkey System 2**. If you have ever used ALTKEYs in Toolkit 2, you'll have a basic idea of what all of these are about. You can define text or strings or commands to be attached to a key. But with hotkeys, in addition to the text, an "action" can be assigned, so that the system would know, for example, to load a given program when you press the keys defined. The Hotkey system also includes something called the Thing list. A "Thing" is basically just a general name for system resources, which can be programs, extensions and just about any other kind of add-on software, hence the vague name of "Things". The Thing list is basically just the computer's list of these added resources to help the QL and you keep track of everything.

Don't worry too much about these little niceties like Window Manager and Things and so on – these things rarely go bump in the night and you can get going with the PE without having to know too much about all these, although it is always helpful to have a little familiarity with the terminology.

QDOS and SMSQ/E

Pointer Environment varies somewhat in terms of how it's implemented on these operating systems. On QDOS systems (including all Sinclair ROM versions plus the Minerva ROM) you have to load three files which provide the necessary extensions to the operating system to allow you to use the PE. The three files are called:

1. **PTR_GEN** – this provides the Pointer Interface.
2. **WMAN** – this adds the Window Manager system
3. **HOT_REXT** – this adds the Hotkey System 2 and associated utilities.

SMSQ/E is the newer version of the QL operating system. The name stands for ‘Single User Multitasking for QL/Extended. The "extended" part of the name refers to the fact that the Extended Environment is built into SMSQ/E - there is no need to load the three files PTR_GEN, WMAN and HOT_REXT as you would need to do for QDOS.

There is an earlier variant of SMSQ/E for the QXL card. This is called SMSQ and it's a version without the 'E' part built in. This is the only variant of the SMSQ system where you do need to install ptr_gen, wman and hot_rext files.

If you do need to get hold of the three files for your computer, you can often get them from commercial pointer driven programs like Qpac2, QD and so on, they are included on the disks of many commercial software packages. You can also download them from my website at http://www.dilwyn.me.uk/pe/index.html or from Wolfgang Lenerz's website at http://www.scp-paulet-lenerz.com/smsqe/add1.html

Memory Requirements

It must be emphasised that PE needs quite a lot of memory! The three files ptr_gen, wman and hot_rext alone can take between 50 and 60 kilobytes of RAM between them, before we even start to think of how much memory the system will need to store copies of the windows of each program running! It quickly becomes apparent that PE simply cannot be used on an unexpanded 128K QL - you realistically need a total of at least 384K of memory, preferably 512K or more to make realistic use of PE.

I remember being at a Quanta workshop in the 1980s where Tony Tebby was about to demonstrate QRAM and he made it clear from the outset: "I've noticed how many people have expanded memory QLs nowadays, so we at Qjump decided to produce QRAM to help you use up all that memory."

512K RAM was a lot in those days.

Qram, Qpac1 and Qpac2

Qram was an application to make use of the Pointer Environment in its original form, back in the 1980s. It was superceded in time by the two newer packages, Qpac1 and Qpac2. The term ‘Qpac’ actually stands for ‘QL Pointer Accessories’, which just means little utilities to use within the Pointer Environment.

Qpac1 was released first and includes a number of small accessory programs such as an Alarm program for setting reminders, a Calendar program, a Calculator, a digital clock program, a Typer program and a Sysmon utility.

Qpac2 on the other hand is a small suite of menus and utilities such as a file manager menu, jobs control menu and a Button Frame where you can park programs while you use other programs. Qpac 2 is neither a replacement for Qpac 1 (the two packages are completely different) and nor is Qpac2 a synonym for Pointer Environment or the Extended Environment. Let's be perfectly clear on that last remark, as people do sometimes confuse the two things. Your Qpac2 disk includes a copy of the Extended Environment files, but also includes a whole lot more. Once you get used to Qpac2 you tend to find it's an indispensable addition to your computer!

Setting Up A Boot File

When using QDOS you need to install the three files **ptr_gen, wman** and **hot_rext**. This is done in a boot program, a short BASIC program which is run as the QL starts up. Boot programs are generally designed to run once, installing every extension you're likely to need, then start any other programs you might need.

A boot program to install pointer environment would consist of a few lines of BASIC to install the extensions in an area called the Resident Procedure space, or Respr space for short. These lines will generally consist of one command to allocate the space needed by each file, then an LBYTES command to actually load the files from disk, then finally a CALL statement to call some machine code within the files to activate them. Here's a simple example.

```
100 REMark the simplest PE boot program
110 base1 = RESPR(20904) : LBYTES flp1_ptr_gen,base1  : CALL base1
120 base2 = RESPR(19880) : LBYTES flp1_wman,base2     : CALL base2
130 base3 = RESPR(12768) : LBYTES flp1_hot_rext,base3 : CALL base3
140 HOT_GO : REMark activate Hotkey System 2
```

Line by line, this program installs the Pointer Interface (ptr_gen) in line 110. The next line loads the Window Manager (wman) and the following line installs the Hotkey System 2. Finally, line 140 issues a HOT_GO command, which activates what is called the Hotkey job, a small program running in the background which waits for hotkeys to be pressed (ALT+another key) and carries out the action associated with that hotkey.

The numbers in brackets after the word RESPR in the listing refer to how much memory is needed to hold that file. The number must be equal to the length of the file in bytes. No harm will be done if the number is bigger, but that would be wasteful of memory. Please note that the numbers refer to the lengths of these files in the latest version of the Pointer Environment (version 2.03 of ptr_gen). The numbers may have to be different for other versions.

For those who have the LRESPR extension on their system - most QL disk interfaces and Toolkit 2 have the extension - you cam simplify the program by replacing the laborious RESPR/LBYTES/CALL commands with a single LRESPR command for each file. LRESPR checks the length of a file, reserves that much memory for them, loads them and calls the machine code at the start of an extensions file to activate it.

```
100 REMark PE boot program using LRESPR
110 LRESPR flp1_ptr_gen
120 LRESPR flp1_wman
130 LRESPR flp1_hot_rext
140 HOT_GO : REMark activate Hotkey System 2
```

Some systems which do not activate the Toolkit 2 extensions may need an extra line containing a TK2_EXT command to enforce its extensions.

```
100 REMark PE boot program using LRESPR
110 TK2_EXT
120 LRESPR flp1_ptr_gen
130 LRESPR flp1_wman
140 LRESPR flp1_hot_rext
150 HOT_GO : REMark activate Hotkey System 2
```

Order Of Loading Extensions

Is there a recognised sequence that extensions should be loaded in? Well, yes there is, up to a point. For QDOS or QXL SMSQ this would generally be:

1. Lightning or Speedscreen (if used)
2. Toolkit 2
3. Pointer environment (ptr_gen, wman, hot_rext in that order)

4. Menu Extension (menu_rext)
5. Other toolkits and extensions
6. Define hotkeys, etc
7. After all extensions loaded, HOT_GO to activate hotkey job
8. Anything else the Boot program needs to do (e.g. set QL clock)
9. Chain in any second program.

Some people might tell you to swap steps 1 and 2 above. This doesn't matter too much. The above is a general guide only, some toolkit packages might tell you that they need to be loaded before or after certain other extensions - this is fine, just follow their instructions.

Boot For SMSQ/E

You would not use Lightning or Speedscreen on SMSQ/E systems, so step 1 would not apply if using SMSQ/E, and SMSQ/E includes the equivalent of the Extended Environment, so step 3 would not be needed either. You probably won't need step 2 either since SBASIC has all the Toolkit 2 extensions already built in. Much simpler, eh?

Boot For Both QDOS And SMSQ/E

Of course, if you are "soft-loading" SMSQ/E on a system which is normally QDOS based, but allows you to run SMSQ/E (e.g. Gold Card, Super Gold Card, Aurora SMSQ/E on QemuLator), you need to be able to load SMSQ/E in your Boot program (or at least give the option of QDOS or SMSQ/E).

SMSQ/E is generally installed with a command such as LRESPR FLP1_SMSQ_REXT in your boot program. The machine then goes through the reset sequence and eventually loads the boot program again. At this point there is the risk that the machine might go into a loop on versions of SMSQ/E which do not check if already installed, so the boot program needs to check if SMSQ/E is already installed. This can conveniently be done with a simple IF VER$="HBA" test in the boot program.

```
100 IF VER$<>"HBA" THEN
110   REMark only offer to install SMSQE if not already present
120   INPUT"1=SMSQE  2=QDOS";os
130   IF os = "2" THEN LRESPR flp1_SMSQ_REXT
140 END IF
150 REMark now install every other toolkit
160 IF VER$ <> "HBA" THEN
170   TK2_EXT : REMark ensure TK2 enabled on QDOS systems
180   REMark install pointer environment on QDOS
190   LRESPR flp1_ptr_gen : REMark in this order...
200   LRESPR flp1_wman
210   LRESPR flp1_hot_rext
220 END IF
230 LRESPR this....
240 LRESPR that....
(and so on)
500 REMark define hotkeys
(and so on)
800 REMark activate hotkeys job
810 HOT_GO
```

This Boot program starts by testing if the machine is currently running in a version of Basic other than SBASIC. SBASIC always has a VER$ version string of "HBA", so if ver$ is not HBA it is reasonable to assume we are running in QDOS. If it finds we are running in QDOS it asks if we want to use SMSQ/E (option 1) or QDOS (option 2). If we enter a 1 in line 120, then line 130 tries to load SMSQ/E. The machine then restarts and since it is now in SMSQ/E, lines 110 to 130 are skipped. We now need a further test (line 160) on whether the machine is currently in QDOS or SMSQ/E. If in

QDOS, lines 170 to 210 issue a TK2_EXT command then install the Extended Environment files ptr_gen, wman and hot_rext. If in SMSQ/E lines 170 to 210 are skipped.

Other toolkits and extensions can then be loaded from line 230 onwards - this part is common to both operating systems (anything only required on one operating system can be separated using appropriate IF VER$= statements as above).

Similarly hotkeys can be defined at line 500 and onward and finally at line 810 we issue a HOT_GO command to make sure that any hotkeys defined earlier in the program are made active.

Further Suggestions

Once you have got this far, you are in a good position to be able to start experimenting with developing your own boot programs. If you have Qpac2, study the examples in the Qpac2 manual which will help further your understanding of this subject.

Don't worry too much about minor complications like Toolkit 2 being built into your disk interface, or things like the AUTOTK2F1 commands of a Super Gold Card. This allows Toolkit 2 to be used on standard QDOS QL ROMs, but you needn't worry about not being able to use the Toolkit 2 if you use SMSQ/E on that system, because SMSQ/E has the equivalent of the Toolkit 2 commands built into SBASIC. If your boot program has a TK2_EXT command, don't worry too much about it having any significant effect on SMSQ/E either. SMSQ/E doesn't need a TK2_EXT command, but it doesn't mind if you use one.

Remember that SBASIC can have an extra complication concerning the use of extensions in what it calls "daughter" SBASIC jobs. SBASIC can have more than one program running at a time, in separate jobs and windows. The main SBASIC (Job 0, the one which SMSQ/E starts into) will install all the extensions you would normally use, and these would all be available to other SBASICs started, apart from any issues with some older extensions which stubbornly refuse to work in anything other than the "mother" SBASIC job 0. The "daughter" SBASICs can load their own extra extensions which are local to that particular program - other SBASICs won't see those extensions.

You can always see which extensions are loaded by issuing the EXTRAS command. This will list all of the extensions to a given channel. It is a handy way of checking if extensions are already present.

John Gilpin showed me a very convenient way of allowing a program to test if an extension name is already present. His short routine sends the output of an EXTRAS command to a file (top marks for using QL device independence!), then running through all the names in that file until either the name is found, or the end of file is found, which means the extension is not there.

```
1000 srch$ = "VIEW" : REMark test if VIEW command installed
1010 OPEN_NEW #3,RAM1_TMP
1020 EXTRAS #3 : CLOSE #3
1030 OPEN_IN #3,RAM1_TMP
1040 found%=0
1050 REPeat loop
1060   IF EOF(#3) THEN EXIT loop
1070   INPUT #3,t$
1080   IF t$==srch$ THEN found%=1 : EXIT loop
1090 END REPeat loop
1100 CLOSE #3 : DELETE RAM1_TMP
1110 IF found% = 1 THEN
1120   PRINT srch$;' present.'
1130 ELSE
1140   PRINT srch$;' not present.'
1150 END IF
```

It is better not to try to use an extension in the same BASIC program as that which installs it. It will probably work OK on more recent Sinclair ROMs, Minerva and SMSQ/E but will almost certainly fail on ROM versions AH and JM at least, because these ROM versions can't cope with using an extension in the same program as that which loads the extension in the same place. Always best to load a second program, e.g. if your boot program loads extensions for a long BASIC game you have written, it is best to keep the boot program short and to the point, just setting up extensions and other essentials, then LRUN (or possibly MRUN) the main program separately. This is called chaining a second program.

What I'm getting at, apart from the specific problem with older ROM versions mentioned, is that it's best to keep your main Boot program short and easy to debug. As our esteemed editor found out a few months ago to his cost, obscure problems in long Boot programs can be very, very difficult to sort out. I have always found it best to have "minimal" boot programs. If your Boot program has to be long, try to split it into logical sections and keep it continuous and in-line as far as possible, without tangles of GOTO commands or complex sets of procedure and function calls. It might not always be possible to split up long programs, but I generally find that I have to just in order to understand some of my rather long past boot programs! If I can't follow them, what chance is there for anyone else? Keep things as simple as possible - the KISS principle (Keep It Simple) - is usually the best approach.