SMSQmulator manual

Please read this ENTIRELY before you do anything else. I won't reply to questions already answered in this document. Required reading stops after section VII, but I recommend you also read the rest. At the end of this document, there is a small section mentioning the changes that were made to this manual.

There is also a User Guide written by Timothy Swenson (“UserGuide.pdf”) – perhaps you want to read that first.

In the following Table of contents, a CTRL+ mouse click will bring you to the corresponding section.

Content

I - What you get

SMSQmulator is an emulator to run SMSQE in a normal Java environment. It is NOT a QL emulator: it does not (and doesn't even try to) emulate a QL with all of its quirks and idiosyncrasies.

It comes in two parts: a Java program, and the SMSQE operating system. They are part of the same zip file called SMSQmulator.zip.

The zip file also contains a starter “QXL.WIN” type file, called “SMSQmulator.win” and used as device “win1_”. When you start up SMSQmulator for the first time, the emulator boots into that device.

Java 8 or 7?

SMSQmulator is written in Java and thus needs a Java runtime environment. SMSQmulator comes in two different versions, for two different Java versions : Java 7 or 8. Though all versions are functionally identical, you should always try to use the latest one (higher number). A version of SMSQmulator for Java 7shoulld run in aJava 8 environment, but not the other way around.
Note that the Java 6 version is abandoned and no version for JAVA are compiled any more (I've noticed that under Java 6, some Linux machines will NOT play sounds, where they will under Java 7 or 8.)

The source code

You may also download the source codes for all parts of the program(s). The web page states which is which. There is a section of this manual devoted to compiling the source code.

II - For a quick start

If you haven't already done so, unzip the file(s) to wherever you want. For the purposes of this manual, I'll assume that you unzipped the “SMSQmulator.zip” file into a directory called “SMSQmulator”.

Starting the emulator

In the SMSQmulator directory there should be the following files/directories:

- SMSQmulator.jar (the Emulator)
- SMSQE (the SMSQE file for the Emulator)
- this manual in .pdf and .odt formats
- a sub-directory called “lib”.
- Another user guide, by Timothy Swenson (UserGuide.pdf).
- A QXL.WIN type container file called SMSQmulator.win.

Just start the jar file up in that directory (the jar file needs the SMSQE file and the “lib” sub-directory to be located next to it). Normally, you should be able to start the jar file by just double-clicking on it – if it doesn't start, you either don't have (the correct version of) Java installed, or your native system doesn't yet know how to start up a “jar” file. Since this is a generic Java question, please check how, under your system, Java “jar” files can be started. As a general hint, a command line as follows should always start a Java jar file:

```
java -jar entire_path_to_jar_file
```

If, after the first start-up, the program either complains that it can't find the ROM file, or else just shows a black screen without much to see, this means that it couldn't find the SMSQE file. You must now tell it where it can find the SMSQE file. Use the "File->Load SMSQE File" menu item to load the SMSQE file. The SMSQE file is normally called SMSQE and is found in the directory where the SMSQmulator.jar file is located.

Running the emulation

Once you have loaded the SMSQE file, the emulation should start automatically. Sometimes you have to click in the SMSQE window to make it accept keystrokes.

If this is the first time you have used SMSQmulator, it will probably start up in “fast mode”, with a small screen of 512x256 pixels, and 16 bit colour mode.

You can change the size of the SMSQE screen via the Config menu – see the configuration section for more information about this.

You can also change the number of colours used: QL colours, 8bit or 16 bit colours. However, for this change to take effect, you MUST quit the emulation entirely and restart it (resetting isn't enough).

I strongly suggest that, once SMQ/E is running, you set at least one directory (folder) for the WIN, NFA and SFA drives: "Config->set dirs for NFA device" (same for SFA) and also one file for a QXL.WIN file: "Config->set files for WIN device". See the Configuration section of this document for more information on this, and also the section on SFA/NFA and WIN devices.

Monitor mode

“Monitor mode” means that there is a monitor (a bit like QMON) where you can, for example, halt the emulation, trace it step by step, execute a subroutine, examine memory or registers etc...

In “monitor mode”, there are a menu bar and 4 windows:

- The SMSQE screen, which should be pretty obvious.
- Two windows below, you can resize between them. These are windows belonging to the monitor (on the left is window #2, on the right window #1).
- A small window beneath them; the monitor command line.

The monitor command line is used to enter commands into the monitor. Type h (followed by ENTER) in there to get a summary of all the commands the monitor accepts.

If the emulation stops, typing g in there will make it go on. K kills the current monitor thread.

You can get rid of the monitor windows by going to the Config item in the menu bar and un-check the “monitor visible” item. This will get rid of the lower three windows.

Fast mode

In “fast” mode, the emulator doesn't “listen” to the monitor. Hence, the emulation runs about 25-50 % faster. Thus, unless you are debugging, fast mode is the way to go... You switch fast mode on via the “Config” menu item (see the Configuration section below). You should then also make the monitor windows disappear (un-check “monitor visible” in the “Config” menu item). The monitor will NOT be useful in fast mode since the monitor commands are more or less ignored when in fast mode. Attention, switching between “fast” mode and monitor mode may result in SMSQE resetting itself. So save any important documents under SMSQE first.

Restarting the emulation and resetting the “machine”.

The only way to really restart the emulation is by exiting it and restarting the emulation program again. You can exit the emulation by: pressing ALT+F4, using the “exit” item in the “File” menu or typing “JVA_EXIT” from an SBasic command line.

You can reset SMSQE by re-reading a ROM file, or by resetting the emulation (both in the “File” menu). Under some systems, the “4 fingered salute” (CTRL, ALT, SHIFT, TAB all pressed at once) will also reset SMSQmulator. There is also a “RESET” command in any SBasic command line.

III - The WIN device

The WIN device uses QXL.WIN type container files as “hard drives/disks”. Each QXL.WIN file is one disk and, of course, these QXL.WIN files exist somewhere in your native file system. The WIN device allows you to read from and write to, a QXL.WIN type file. You get a device called WIN (but this may be changed, see the Configuration section) with 8 drives, each of which must point to a different QXL.WIN type file somewhere in your native directories. The configuration menu item in the menu bar allows you to choose which QXL.WIN file should be associated with what device (see the Configuration section). You can then access the individual drives as win1_ to win8_, as usual. Normally, accessing a QXL.WIN type file should be faster than accessing files on NFA and SFA devices (see below).

One of the first configuration operation one usually undertakes is to set the name of the main QXL.WIN file to be used as win1_.

Formatting a WIN Drive

WIN Drives can be formatted with the **FORMAT** command. This may take one of two different forms:

FORMAT WINx_NUMBER (e.g. FORMAT win1_50)
or
FORMAT WINx_NUMBER_NAME (e.g. FORMAT win1_50_Sources).

The NUMBER parameter must always be present and indicates the size of the drive, in MiB (1MiB=1024*1024 bytes). This NUMBER may not be lower than 1 or higher than 2000 and if, other than for testing purposes, you really format a 2 GB hard disk for SMSQE, I'd REALLY like what you put on there...).

The optional NAME parameter contains the name of the new drive, which is displayed, for example, when you do a **DIR** of the drive. The name may not be longer than 20 characters (anything beyond that will simply be cut off). If you do not pass the optional NAME parameter, the drive will get the current usage name of the device and the drive number (e.g.: FORMAT win2_50: the name will be WIN2.)

ATTENTION: The device name to be used when formatting is the current usage name of the device. If you named your WIN device “fld”, then you must do a format fld1_...

To be able to format a drive, you must have first of all configured (via the config menu item) the name of a valid **BUT NOT YET EXISTING** native file that will hold the QXL.WIN type file (and, yes, this will make SMSQmulator complain that the file doesn't exist). After the FORMAT, a corresponding file of the desired size will have been created.

As a safety feature, you **cannot** format an already existing file. If you try, this will fail with the error “already exists”. You must delete the corresponding file first from within your native file system.

IV - The NFA and SFA devices

These are two new devices (NFA and SFA) giving you access to the underlying “native” file system of the machine on which you run SMSQmulator and allowing you the save files to/from the native file system. Thus both devices allow you to read/write to native file systems, but the SFA device keeps the QL file header, whereas the NFA device does not. As usual with SMSQE, you get 8 drives for each device, NFA1_ to NFA8_, same with SFA.

The way this works is that you assign to each drive on these devices, the name of a directory in your native file system and all reads/writes for that drive will go to that directory. With the "Config" menu in SMSQmulator, you can tell each drive of these devices to what directory on you native file system it should point. Alternatively, you can also use the SFA_USE and NFA_USE commands (see the Configuration section).

The NFA (Native File Access) device

This enables simple read/write to native directories. The name of the device is NFA (but this may be changed, see the Configuration section), so you have 8 drives named NFA1_ to NFA8_.

If you save files to an NFA device, the QL file header will be lost (the SFA device is able to write SMSQE style file headers, see below...), so you cannot, for example, EXEC a file from an NFA drive. Loading/saving basic programs or LRESPRing files such as qpac2 will work, though.

The NFA driver ** should ** be safe, but do point each drive to somewhere where it can't create havoc with your files — a newly created directory will suffice.

Please note that, depending on your OS, file names may be case sensitive (contrary to usual SMSQE convention) so that NFA1_FILE1 may be different from NFA1_file1. However, it is possible to make the device “semi case independent” via a configuration option: Please see the section on configuration.

Normally the program remembers the names of the NFA drives as they are stored in a file called SMSQmulator.ini in your home/personal directory.

The SFA (SMSQE File Access) device

This is a device that, like NFA, stores files somewhere on you native file system, but with a special file header (invisible to SMSQE). Thus, you can **SEXEC** files to it, and **EXEC** files from it. The name of the device is SFA (but this may be changed, see the Configuration section), so you have 8 drives,. SFA1_ to SFA8_.

The SFA device will only seem to contain those files that do have a corresponding SFA file header, **i.e. only files that were saved through itself or through Q-Emulator**. If there are other, non SMSQE, files in the native directory to which the SFA drive is bound, the SFA device will not “see” them – they won(t show up in a **DIR** command, for example.

A word of caution, though:
It is possible – but NOT RECOMMNDED- to point SFA and NFA devices to the same underlying directory. DON'T DO THIS (see below)! Suppose you have a directory called “Smsqe" somewhere on your native file system, and you point both SFA1_ and NFA1_ to it.
Further suppose, that you have 3 files in this directory:

- file1 – a "normal" file containing whatever you put in there.
- file2 – a file with an SFA file header saved through the SFA drive.
- file3 – another "normal" file.

If you do “**DIR** NFA1_” this will show:
- file1
- file2
- file3

since these are all of the files in the directory.

But If you do “**DIR** SFA1_” this will only show:
- file2

since that is the only file with an SFA style file header in your "Smsqe" directory. So far, that is the expected behaviour since the SFA device only “sees” those files that have a special file header.

However, suppose you now try to create a new file called "file1" on the **SFA** drive, for example by typing “**SAVE** SFA_file1”. You would expect this to succeed. For the SFA drive, this would be valid, since, for it, there is no file called "file1" on the underlying directory, as it only "sees" those with an SMSQE file header. But, of course, on the native file system to where SFA1_points, there already IS a file called file1, only, because it is not an SMSQE style file, SFA1_ doesn't show it to you. If you now try to get SFA1_ to save a new file with exactly that file name, this would means that it would try to overwrite the file called "file1" in your directory – even though you might not want to do that, since, perhaps, you only did a DIR of SFA1_ and thus forgot that there is already a file called "file1" in that native directory.

To avoid this possible loss of files, the SFA device refuses to write over, or delete, files that do not have an SMSQE file header. It will return the error "is in use". Thus, you data is safe.

However, this may produce the confusing situation that you try to write a file, perhaps called file1; to the SFA device, which it refuses, saying that the file is in use – but a DIR of your SFA drive will not show this file to be existing!

I would thus recommend that you create a special directory just for the SFA drives and where only the SFA drives read/write to.

N.B.: the file header for files on an SFA device used to be the same as that for SMSQE files, except that an additional identification long word "SFA0" was tacked onto the front. As of version 1.03 of SMSQmulator, this special header is abandoned and the new header is compatible with that of Q-Emulator, with one difference however: Q-Emulator only saves this header for those files that need it (i.e., to all intents and purposes, EXECutable files only) whereas SMSQmulator saves it for ALL files. SMSQmulator can still read (but not write) files with the old header.

Do not mix the directories for NFA and SFA devices

As mentioned above, temptation may be great to have some NFA and SFA device point to the same underlying native directory. The situation here is clear: **DON'T.** You will cause yourself untold hours of grief. As an example: you save some SBASIC extensions file under SFA. If you LRESPR this file under SFA again, all will be fine. If you LRESPR the same file under NFA, your machine will crash 'due to the presence of the special header).
Or you save some file under NFA and wonder why the SFA drive can't see it, etc... So DON'T. Keep the NFA and SFA directories separate. Always.

Avoid accented characters in NFA and SFA file or directory names

SMSQmulator doesn't handle file names with accented characters in SFA and NFA files/directories very well. You might want to avoid them.

You cannot FORMAT NFA or SFA drives

Well, the title says it all: trying to FORMAT NFA or SFA drives will give an error “not implemented”.

The file names must be valid under the native file system

When you create a file under the NFA and SFA devices, corresponding files are created on the native file systems the respective drives point to. This means, that the file name you choose must be valid file names under the native file system. For example, under SMSQE it is perfectly valid to create a file with a star (*) in its name. Under Windows, this is not a character that can validly be used in a file name. Please only use those names that can actually be used under the native file system.

NFA and SFA drives to not support direct sector accesses

Due to the nature of these drives, direct sector access to the drives (e.g. winx_*d2d) is NOT allowed.

Do not use SFA and NFA devices with native directories containing “_”

The underscore character (“_”) is special in SMSQE and is used as a sub-directory divider amongst others. It is perfectly acceptable to use the underscore in SMSQE file or directory names, SMSQmulator will switch between that and the directory separator your native directory uses without any problems.

However, the names of native directories themselves must NOT have a "_" in them – if they do, the NFA and SFA devices will get hopelessly confused.

Making a directory may result in file name changes

Under SMSQE, when you make a new directory and an existing file has a name that should then belong with that directory, the file is moved into this new directory. SMSQmulator mimics that behaviour in the native directories as well. Suppose you have an SFA device pointing to a directory called "smsqe" somewhere on your native file system. Suppose you have just two files in that directory: a file called `test_doc` and another file imaginatively called `another_file`.
A "**DIR** SFA1_" will show:
`another_file`
`test_doc`
which are the two files on SFA1_.

Now you create a new directory called "`test`" on SFA1_:
MAKE_DIR SFA1_test
A "**DIR** SFA1_" will now show:
`another_file`
`test ->`
which is as it should be.

If you do a **DIR** SFA1_test_ you will see:
`test_doc`
which is as it should be.

The file `test_doc` has been moved into the `test` sub-directory. You will notice that something similar happened on your native file system: a (native) sub-directory called `test` was created in the "smsqe" directory, the file `test_doc` was moved into it, but its native name changed from `test_doc` to `doc`. Under SMSQE, you will still see it called `test_doc`.

V - The FLP device

The FLP device allows you to read, write and “format” QL format floppy disk **images**, NOT floppy disks themselves. A floppy disk image is a file containing an image of a floppy disk. For the time being, the ONLY images supported are those of 720 KiB and 1.44 MiB floppy disks. The length of the images files must be **exactly** either 720 KiB or 1.44 MiB.

Thus you must prepare a floppy disk image before you can access it from within SMSQmulator. How exactly you can prepare such a floppy disk image will depend on your operating system, under Linux the “dd” command will do. Please beware that in some cases, the resulting floppy image file may be read only, in which case the flp drive will be treated as a write protected disk. You may also not even have the necessary permissions to read from the image file, in which case SMSQmulator will complain that it can't even find the disk.
.
The resulting “img” file will contain the floppy disk image. This is the file you can then read via SMSQmulator. (By convention, the resulting image file should have an extension of “.img”, even though this is not absolutely necessary).

The configuration menu item in the menu bar allows you to choose which .img file should be associated with what device. You can then access the individual drives as flp1_ to flp8_, as usual.

As of version 2.00, SMSQmulator also lets you write to the floppy disk image file. You can then later rewrite that modified image file to a true floppy disk if your system has one. Again, exactly how you do that depends on your operating system.

You may also “format” a floppy disk, i.e. create an empty disk image file in QL format. By default, SMSQmulator will format a 720 K disk. You can force it to format to 1.44 MiB in two ways, as is usual under SMSQE:

Set the default density via the FLP_DENSITY command (see below).
Append a star plus a letter at the end of the device name when formatting:

FORMAT “flp1_*D” formats a double density disk (i.e. 720 K).
FORMAT “flp1_myname*H” formats a high density disk (1.44 MiB).called “myname”.

Note that you need the quotes around the name when you use an asterisk (star) in the name of the device to format. The allowed letters are S, D, H and E for Single, Double, High and Extra High density. However, SMSQmulator doesn't handle single density or extras high disk images and will use double density instead if you specify single or extra high density.

The FLP_DENSITY command can be used to set the default density of the image files. This doesn't concern pre-existing images, SMSQmulator will read the density information from within the disk. The FLP_DENSITY command can be followed by one of four letters:

FLP_DENSITY S | D | H | E for Single, Double, High and Extra High density. However, SMSQmulator doesn't handle single density or extras high disk images and will use double density instead if you specify single or extra high density.

Please note: Apart from FLP_DENSITY, FLP_USE, FLP_DRIVE and FLP_DRIVE$ (see below), all the other usual SMSQE FLP_xxxx commands, while they exist and seem to be working, will have no effect.

FLP_DRIVE number,native_name$
This keyword allows you to set a drive to a native image file. `number` in the drive number (1 to 8) and `native_name$` the name of the native file. Remember, some native file systems are case-dependent. You will get an error if the file can't be found or isn't a valid QL disk image file. In that case, if that drive previously held a valid image, that image is closed and the drive no longer points to a valid disk image.

result$=**FLP_DRIVE$ (number)**
This function returns the native name of the drive given as parameter `number` (1 to 8). If the drive isn't assigned, an empty string is returned.

The floppy disk driver expects you to behave in a rational manner:

You could conceivably do something like this:

(go into the config menu and set the drive image for flp1_ to point to a valid QL image disk file)

```
OPEN_NEW #3, "flp1_whatever"
PRINT#3, a_long_string$
```

(now, with that file still open, you go into the config menu and change the drive image for flp1_ to point to another file and then you type:)

```
PRINT#3, an_even_longer_string$
CLOSE#3
```

The result of that will be undetermined – but don't be astonished if the new drive image no longer contains a valid QL Disk image and/or of files are corrupted...!

Of course, changing the disk when no file is open to it should not present any problems.

Likewise, the floppy driver does NOT stop you from using the same image file for two floppy drives. Of course, if you write to both of the drives, everything will get hopelessly confused since one drive doesn't know what the other wrote to the image file!

VI - The MEM device

The MEM device is very close to a WIN device: it also operates on QXL.WIN files, except that the entire QXL.WIN file is loaded into memory at once and kept there until SMSQmulator is terminated. There is no facility to unload it from memory once loaded.
Mem drives are of no real usefulness in normal operations, they are used to provide a "writeable" win drive when SMSQmulator is used as an embedded applet in a website.

Loading a MEM drive

You must indicate, via the "config" menu, which file is to be used for a MEM drive, just as you would do for WIN drives. It is also possible to pass a filename as a URL and not as a normal filename. For local files, this means that you should add "file:/" before the name once you have selected it – SMSQmulator does not do that automatically for you.
Once a MEM drive is loaded, it behaves like a WIN drive, with the following exceptions:
1 – The maximum allowed size for a MEM drive is 500 MiB. QXL.WIN drives larger than that are simply not loaded. Depending on the memory capacity of your machine, the size may even be less.
2 – MEM drives cannot be formatted.
3 – Any changes made to the content of the MEM drive are lost if the entire QXL.WIN file isn't written back to the disk.

Writing the drive back to the disk.

SMSQmulator does NOT do that for you automatically. To write the file back to the same file it was loaded from, use the **MEMD_WRITE** S-Basic procedure.
MEMD_WRITE drive_number
where drive_number is the drive to be written back (1 … 8).
This writes the entire QXL.WIN file back to the original file it was loaded from, overwriting that file in the process. There is **no** confirmation request for this.

Using MEM drives slows loading of SMSQmulator

If you configure SMSQmulator to use a MEM drive (and unless you disable it), it tries to load the entire file into memory when it is started. This may considerably slow down the start time for SMSQmulator.
Operations on a MEM device won't be faster than on a WIN device.

VII - Getting the bigger picture

On modern big screens, an SMSQE type screen can seem small and forlorn – and difficult to read.
SMSQmulator can make its screen look bigger (magnified) if you wish. However, be advised that this will generally make the SMSQE screen look not very nice., since the pixels are just scaled to look bigger.
Also, the window used by SMSQmulator will be twice the size (in each direction), so make sure that the new size will fit on your screen.
You can get this effect by two methods:

- Re size SMSQmulator's window. This will make the fonts bigger or smaller. However, they may look distorted, since internally, SMSQE will keep its original resolution.
- A slightly better effect can be achieved with the “Double the window size” item in the “Config” menu. Here, the SMSQE “pixels” will be doubled in size in each direction. This doesn't necessarily make the SMSQE display look that much nicer, but it will at least be to scale exactly. This same config item then allows you to switch back to the previous size.

It is recommended that you let the monitor windows NOT be visible when changing sizes like that.

VIII - Faster floating point routines

Taking a leaf out of QPC's book, I have (optionally) replaced some SMSQE floating point routines by faster ones, based on Java “doubles”. You should see a 2-4 fold speed increase with most routines involving floating point numbers.
The operations that have been sped up are: addition, subtraction, multiplication, division, SIN, COS, TAN, COT, squaring, doubling, halving, square root and reciprocal (1/x).
Prior to version 2.08, SMSQmulator came preconfigured to use the original SMSQE routines, it now comes preconfigured to use the faster routines.
You can still switch between the faster routines and the original SMSQE ones. To do that you can either configure the SMSQE file (see the section “Configuration with the standard (menu) config program”) or use the following new SBasic keywords:
JVAQLFP turns the old SMSQE routines on
JVAIEFP turns the new routines on (default unless configured otherwise).

IX - The TCP/SCK devices

As of version 2.21 SMSQmulator has devices that allow opening TCP and SCK channels, in a manner compatible to the uQLx documentation. This may allow you to connect to the Internet, provided the host machine can do it. If you manage to get Lynx running under SMSQ/E, this will work.
There is no provision for a UDP connection.
Beware, SMSQmulator only implements some of the TRAP#3 calls provided for by the uQLx documentation, namely trap#3 with D0 =: 1 to 7,50,51,53,58,59,5b,5e,62,7b,7c. (numbers are in hex). If your software needs more, please tell me.

X - Configuration

There are several ways to configure SMSQmulator: through the “Config” menu item in the main menu bar, through the standard SMSQE config programs (preferably MenuConfig) and through some basic keywords.

Please note that you should configure SMSQmulator with the config menu item, while you should configure the SMSQE file with the MenuConfig program.

The SMSQmulator.ini file

SMSQmulator stores the options you configure through the config menu item in a file called “SMSQmulator.ini”. This may be located in your home or personal directory, or in the directory the program was started from.

SMSQmulator will first look for the “SMSQmulator.ini” file in the directory it was started from, and it if finds it there, will use that file. If not, it will look for this file in your home/personal directory and use that or possibly create the file there if it doesn't exist yet.

The “SMSQmulator.ini” file is a simple text file, you can edit it with a simple text editor if you want, the info in it must be in the shape: option = value.

The options are re-read every time the emulation is run (not every time the machine is reset!).

The “Config” menu item

The “Config” menu item in the menu bar allows you to set a certain number of options. Please note that options 2 – 20 below are tucked away in the “Devices” sub-menu.

1. Setting the language

This allows you to set the language used by the emulator, but NOT the language used by SMSQE. Please use the standard LANG_USE command for this.

2. Setting the directories for the NFA drives

Each NFA “drive” (NFA1_ to NFA8_) points to a directory on your native file system. Here you can choose what directory to use for each of the 8 drives. You can also use the **NFA_USE** command for this (see the Basic keywords section).

3. Setting the case of file names for the NFA device

Depending on your OS, file names may be case sensitive (contrary to usual SMSQE convention) so that, for example, “NFA1_FILE1” may be different from “NFA1_file1”. This is the case, for example, in Linux, (but not in windows, where file names are case independent). Case dependent file names may pose a problem for SMSQE, since sometimes, some programs may change the case of a file name but still expect to open the same file. As an example, the GST macro assembler will open files in upper case : if you have a file called “nfa1_test_asm”, the assembler will try to open “NFA1_TEST_ASM”. Under a Linux system this will fail, since there is no file by that exact name.

Here you can configure how SMSQmulator deals with that: It can,:

- not change the file names at all (good, for example, on windows but should not be used on file systems where case matters);
- set all file names to lower case;
- set all file names to upper case.

If you choose one of the latter two options, any file name passed to the NFA device will have its case converted. So, for example, if you set to convert file names to lower case, and type “SAVE NFA1_FILE1” it will actually be saved as NFA1_file1. In that case, when you copy files from outside SMSQmulator to the directory used as NFA1_, make sure that these files have the correct (upper or lower cased) name.

However, again, there is a "problem“: Suppose that NFA1_ points to a directory called “smsqe” and you have two files on drive NFA1_ called “FILE1” and “file2”. Also suppose that you configured the NFA device to change the file name case to all lower case. If you try to open “NFA1_file2”, you will have no problem, SMSQmulator will try to open the file “file2” in the directory “smsqe”. Now suppose you try to open the file FILE2. Since you configured the device to change the file names to lower case, SMSQmulator will try to open a file called “file2” (not “FILE2”) in the directory “smsqe” - and it will fail. However, SMSQmulator normally hides file names that don't correspond to the file case you have configured.

So, a little bit of discipline is in order: If you are using SMSQmulator on a system where file name cases matter, point the devices to a directory where all file names are in the same case and DO set the configuration to change the file name cases to that (lower or upper) case, whatever suits you.

4. Setting the usage name of the NFA device

Normally, the NFA device is called NFA, and you have drives NFA1_ to NFA8_ on it. You can change the name of the NFA device to a new usage name. For example, if you set it to “WIN”, then the device will no longer be called NFA but will be called WIN and you have drives win1_ to win8_.

The new name must be exactly three letters (no numbers!) long, or an empty string. Anything else will cause the new name to be rejected. An empty string will reset the device's name to its original name.

If you set a usage name, the device will no longer be accessible through its original name, but only through its usage name. Thus, the device will then no longer respond to “NFA”: rid NFA1_ would result in it not being found.

You can also use the **NFA_USE** command for this (see the Basic keywords section).

Please note that if you call the device WIN and you have a file called “boot” on win1_, SMSQE will attempt to boot from that file.

5. Disabling the NFA device

If you never use this device, you might disable it entirely. The device will then not be linked into SMSQE at all. This will diminish the number of devices in the system, perhaps alleviating the menu extensions problem (see below, the section on “Quirks”).

6. Setting the directories for the SFA drives

Each SFA “drive” (SFA1_ to SFA8_) points to a directory on your native file system. Here you can choose what directory to use for each of the 8 drives. You can also use the **SFA_USE** command for this (see the Basic keywords section).

Please note that if you set the directory names through the config item, this may only take effect the next time you reboot the machine. Setting it through the basic keyword is immediate, but will be forgotten the next time you reboot the machine (the info is NOT stored in the .ini file when using the basic keyword).

7. Setting the case of file names for the SFA device

The same reasoning as for the NFA device applies. See point 3 above.

8. Setting the usage name of the SFA device

Normally, the NFA device is called SFA, and you have drives SFA1_ to SFA8_ on it. You can change the name of the SFA device to a new usage name. For example, if you set it to “WIN”, then the device will no longer be called SFA but will be called WIN and you have drives win1_ to win8_.

The new name must be exactly three letters (no numbers!) long, or an empty string. Anything else will cause the new name to be rejected. An empty string will reset the device's name to its original name.

If you set a usage name, the device will no longer be accessible through its original name, but only through its usage name. Thus, the device will then no longer respond to “SFA”: DIR SFA1_ would result in it not being found.

You can also use the **SFA_USE** command for this (see the Basic keywords section).

Please note that if you call the device WIN and you have a file called “boot” on win1_, SMSQE will attempt to boot from that file.

9. Disabling the SFA device

If you never use this device, you might disable it entirely. The device will then not be linked into SMSQE at all. This will diminish the number of devices in the system, perhaps alleviating the menu extensions problem (see below, the section on “Quirks”).

10. Setting the files for WIN drives

The WIN device uses QXL.WIN type container files as “disks”. Each QXL.WIN file is one disk and, of course, these QXL.WIN files exist somewhere in your native file system. Here you can set the names of the QXL.WIN type files for the WIN device. You can also use the **WIN_USE** or **WIN_DRIVE** keywords for this (see the Basic keywords section).

11. Setting the usage name of the WIN device

Normally, the WIN device is called WIN, and you have drives win1_ to win8_ on it. You can change the name of the WIN device to a new usage name. For example, if you set it to “QXL”, then the device will no longer be called WIN but will be called QXL and you have drives qxl1_ to qxl8_.

The new name must be exactly three letters (no numbers!) long, or an empty string. Anything else will cause the new name to be rejected. An empty string will reset the device's name to its original name.

If you set a usage name, the device will no longer be accessible through its original name, but only through its usage name. Thus, the device will then no longer respond to “WIN”: DIR win1_ would result in it not being found.

You can also use the **WIN_USE** command for this (see the Basic keywords section).

Please note that if you call the device WIN and you have a file called “boot” on win1_, SMSQE will attempt to boot from that file.

12. Disabling the WIN device

If you never use this device, you might disable it entirely. The device will then not be linked into SMSQE at all. This will diminish the number of devices in the system, perhaps alleviating the menu extensions problem (see below, the section on “Quirks”).

13. Ignoring Qxl.win file lock errors

SMSQmulator uses “file locking” to make sure that no two instances of the program can access a same qxl.win file at the same time. You can see that happening when you launch two instances of SMSQmulator one after the other: the last one launched will not be able to access the qxl.win file(s), and that SMSQmulator warns you that it cannot access the file since it is already opened by another program.

Apparently, on some systems (e.g. a Mac), when a qxl.win file that lies on a remote drive is accessed over the network, the file refuses to be “locked”. Normally in such a case, SMSQmulator generates the error and does not use this qxl.win file. If you configure SMSQmulator to ignore that error, it will try to use a qxl.win file even if it could not acquire a file lock. Please be aware that if two instances of SMSQmulator (or other programs) try to write to the same qxl.win file, then havoc will ensue, with totally unpredictable, but certainly disastrous, results. **YOU HAVE BEEN WARNED**.

14. Make unlockable files read only

To alleviate the problem of two programs writing to an unlockable qxl.win file, you may decide that any file that is unlockable becomes read-only. So, if two instances of SMSQmulator try to access the same qxl.win file, the first one will have normal read-write access, the second one read only.

Please note :

1 - This option will only be used and acted upon if you opted to ignore Qxl.win file lock errors (it is even greyed out until you opt for that).
2 – If any drive becomes read-only under this scheme, you may get a warning that the drive is read only. Remember that you may switch these warnings on or off.
3 – It is preferable to restart the emulation whenever you change this option.

15. Setting the files for FLP drives

Here you can set the names of the .img files for the FLP device. You can also use the **FLP_DRIVE** keyword for this (see the Basic keywords section).

16. Setting the usage name of the FLP device

Normally, the FLP device is called FLP, and you have drives flp1_ to flp8_ on it. You can change the name of the FLP device to a new usage name. For example, if you set it to “MDV”, then the device will no longer be called FLP but will be called MDV and you have drives mdv1_ to mdv8_ .

The new name must be exactly three letters (no numbers!) long, or an empty string. Anything else will cause the new name to be rejected. An empty string will reset the device's name to its original name.

If you set a usage name, the device will no longer be accessible through its original name, but only through its usage name. Thus, the device will then no longer respond to “FLP”: DIR flp1_ would result in it not being found.

You can also use the **FLP_USE** command for this (see the Basic keywords section).

Please note that if you call the device WIN and you have a file called “boot” on win1_, SMSQE will attempt to boot from that file.

17. Disabling the FLP device

If you never use this device, you might disable it entirely. The device will then not be linked into SMSQE at all. This will diminish the number of devices in the system, perhaps alleviating the menu extensions problem (see below, the section on “Quirks”).

18. Setting the files for MEM drives

This is similar to setting file names for WIN drives. However, you may also pass a file name as a URL and not as a normal file name. For local files, this means that you should add “file:/” before the name once you have selected it – SMSQmulator does not do that automatically for you.

19. Setting the usage name of the MEM device

This is similar to setting the usage name for the WIN device.

20. Disabling the MEM device

If you never use this device, you might disable it entirely. The device will then not be linked into SMSQE at all. This will diminish the number of devices in the system, perhaps alleviating the menu extensions problem (see below, the section on “Quirks”).

As a default, the MEM device is disabled.

21. Setting the screen size

Here you can set the screen size. Please set this as a combined number such as 1024x512, i.e. a number, followed by the letter ”x” followed by another number.

Beware: The screen size is reset immediately, which will cause a reset of the SMSQE machine!

If you set the screen size to 512x256 and the colour mode to QL colours, then SMSQmulator makes a QL compatible screen at $20000.

If you use a screen with QL colours, then the X size of the screen must be a multiple of 8. SMSQmulator doesn't complain if you set a screen size that is not a multiple of 8, but will then silently make the screen bigger to the next multiple of 8.

The minimum screen sizes are of 512 for the X size and 256 for the Y size. SMSQmulator will not complain if you set a screen size that is smaller in any dimension (or both), but will then silently make the screen bigger, to at least 512 x 256.

22. Setting the colour mode

SMSQmulator allows three colour modes:

- normal QL colours (i.e. modes 4 and 8 – but no flash in mode 8)
- 8 bit colours, Aurora compatible
- 16 bit colours (aka high colours).

Screen redraws etc. should be faster in 16 bit mode. Apart from trying out old games, there should not be any reason not to use the high colour modes.

For SMSQmulator versions before 2.17, the change in colour mode ONLY takes effect the next time you restart the emulation entirely. For versions 2.17 and beyond, the change takes effect immediately but causes a reboot of the machine.

23. Setting (very) slightly brighter colours

If this is set, the colours displayed in 8 and 16 bit modes ****may**** look a little bit brighter, depending on your eyes and monitor. This setting has NO effect in QL compatible mode.

24. Double window size

Here, you can determine whether the SMSQE “pixels” should be displayed at double their usual size. This doesn't necessarily make the SMSQE display look that much nicer, but it will at least be to scale exactly. This config item then allows you to switch back to the previous size. See the section “Getting the bigger picture” for more information.

25. Screen update rate

The screen update rate is the rate at which the screen is updated (redrawn), expressed in milliseconds. The lower the figure, the more often the screen is redrawn. The normal rate is 50 which means that the screen is redrawn every 50 milliseconds (= every 20th of a second). This is done via a thread that is independent of the main emulation thread, at least on modern machines with multicore processors. On these machines, the screen update rate should remain at that value. On slow machines and especially on single core machines, you might want to set this to a higher value, so that the screen will get redrawn less often. This could be useful, notably to speed up the boot process : set it to a high value at the start of the boot process, and a normal value at the end.

The value is restricted to the range 0 … 65535. If you enter a lower value, the program will silently set the value to 0. If you enter a higher value, it will automatically be set to 65535.

26. Allow QL screen emulation

This sets whether QL screen emulation will be possible or not. After changing the state of this item, you must restart SMSQmulator entirely, else it is not taken into account. Please see the section on QL Screen emulation.

27. Window mode

The Emulator's window (which shows the emulated screen) can be shown either as a window on your desktop, or in full screen mode, where it takes up the entire screen.

Please note: Any changes you make via this menu will only take effect if you leave the application and restart it again.

More precisely, there are three display modes :

1 – Window.
SMSQmulator's window is just a normal window on your desktop. This was the way the window was displayed up to version 2.17 of SMSQmulator, and still is the default today.

2 – Full screen.
The window takes over the full screen. There is no bar on top of the window, except for SMSQmulator's own menu bar. You probably also want to hide this menu bar when using this mode. The emulated screen will be stretched to fill the entire monitor screen – but the emulated resolution will not change. This will most probably mean that the display is not very nice as it will be distorted in one direction. You can get around this by :

- setting the emulated resolution to the same resolution as your monitor. In this case, you should make sure that you have not configured SMSQmulator to « double the window size ». Switch this off before you choose this mode. SMSQmulator can optionally display a warning if this is still switched on;

- setting the emulated resolution to half the resolution of your monitor (in each direction) and then switching the « double the window size » config item on,

- using the special full screen mode.

3 – Special full screen.
Here again, the window will be in full screen mode, i.e. covering the entire screen, with this difference : Here, SMSQmulator will not honour the screen resolution you have configured for the emulated screen size, but will automatically set the emulated screen's resolution to the resolution of your monitor.

In this mode, SMSQmulator will, however, honour the «double the window size » configuration item. If this is activated, SMSQmulator will automatically set the emulated screen's resolution to half the resolution of your monitor (in each direction) and then display this at full screen size, effectively making each QL pixel "bigger". This means that you'll have a nicely proportioned full sized screen.

28. Setting the monitor used in full screen mode

By definition, in full screen mode, SMSQmulator's window will occupy the entire screen. In a multi-monitor environment, where the user has more than one display connected to the computer, the user must decide which display the program is to use. This configuration item lets you choose between the different displays.

Please note: this configuration item is only visible if you are in a multi-monitor environment.

29. Setting the sound volume

You can set the volume of sounds SMSQmulator can produce. The range of values is 0 to 100, with 100 being the loudest and 0 being no sound at all. Any value that exceeds this range is silently adjusted to fit.

30. SSSS Frequency

Normally, the frequency of the SMSQ/E Sampled Sound System is 20 KHz. However, on some systems, this frequency will not be allowed, so SMSQmulator can also use a frequency of 22.05 KHz, which most systems will allow. This is also the default value, but it means that the sound may be played a bit faster than it should be. With this configuration item, you can change between both frequencies. The change takes effect next time you start SMSQmulator.

As of v. 2.21, SMSQmulator will resample the sound when the frequency is set to 22.05 KHz, so that it no longer plays faster.

31. Setting the memory size

Ranges can be from 1 to 240 MB. If you use high colours, the minimum must be 8 MB, if you set less, the machine will still use 8 MB! Note that, whilst the screen itself does NOT take up any of the main memory you configure here (except when in 512x256, QL colours mode), the windows themselves, when other windows overlap them, still take a lot of memory for their save areas.

Unless there is a specific reason not to, I'd use at least 32 MB.

32. Monitor Visible

Use this to make the monitor windows visible or not.

33. Fast mode

In “fast” mode, the emulation runs about 50 % faster. However, the monitor will NOT be useful in fast mode, since no breakpoints etc. can usefully be set, as the monitor commands are more or less ignored when in fast mode. However, unless you are debugging, fast mode is the way to go...

34. Use less CPU time when idle

When this is checked, SMSQmulator tries to determine when the machine is idle and, if it finds this to be the case, will use much less CPU time. As soon as the machine needs full speed again (i.e. does something other than just blink a cursor), CPU use by SMSQmulator should ramp up again.

Any change in this option only takes effect once you reset SMSQmulator.

35. (Cursor blink pause)

** Please note: this is redundant since the introduction of the above configuration option ("Use less CPU time when idle") and is abandoned as of version 2.17 of SMSQmulator. **

[Normally, the emulation should always run as fast as possible, so as to make SMSQE as fast as possible. However, this will mean the your CPU, or at least one core of your CPU if you have a multicore machine, will always be at 100% system charge since SMSQmulator will have it run flat out.

Under some circumstances, this might not be what you wish – the entire computer might get sluggish. For this reason, SMSQmulator can suspend itself for a small time whenever the cursor is about to blink. In general, a blinking cursor means that the emulated machine is just sitting there twiddling its figurative thumbs. Slowing down the emulation at that point should have no negative impact on performance. However, if you're doing something that needs performance, take care that there isn't another window with a blinking cursor open (or set the blink value to 0).

The value you enter is a number of milliseconds. 100-200 seems to work for me, depending on the machine I'm on.]

36. Suspend execution when the window is minimised

When you check this box, the emulation will suspend itself when SMSQmulator's window is minimised. This means that it will no longer use up system resources. The emulation will resume when you reopen the window.

37. Action taken after a JVA_POPUP event

After receiving a JVA_POPUP event, SMSQmulator can either re-open its window, or make its taskbar entry blink. Here you can set which of these two actions it should perform.

38. Pause after mouse click

On some systems, when clicking and releasing the mouse (especially when tapping a laptop mousepad), both the click and the release of the mouse button may happen so fast that SMSQmulator doesn't register it. If this happens (i.e. if a mouse click doesn't seem to do anything), try setting this parameter to something like 50.

39. Time correction

Normally SMSQmulator should adjust for different time zones and Daylight Saving Times (DST).

If it doesn't, you can configure the number of seconds (+ or -) to be added to the time here, so that SMSQmulator's and your operating system's clocks are in sync. Remember, one hour is 3600 seconds.

Note, if you have to make use of this facility, It may happen that the dates of some (normally not all) files on an NFA/SFA device do not correspond to those you would see from within a native file manager (the difference shouldn't be more than the time added for DST). There is nothing to be done about that.

40. Hide the menu bar

This will hide the very menu bar in which you can get at this config item. To get the menu bar back, type the following SBasic command: JVAMBAR.

41. Warnings

From time to time, SMSQmulator may generate some warnings, for example, if you try to use a non existing file as a WIN drive. In this section of the configuration menu, you can switch individual warnings on and off:

→ *Warn if a WIN drive doesn't comply with the standard*
It has been pointed out to me that some (older versions of some) utilities that allow the creation of WIN.WIN type files create files that do not really comply with the WIN.WIN standard. Whilst SMSQmulator tries to cope with these files as best it can, it can warn you if this happens. I would suggest that if you have such a non-compliant QXL.WIN file, you do not use it any further, but copy all files to a compliant one.

→ *Warn if a WIN drive can't be opened because the native file doesn't exist*
When you configure SMSQmulator to use certain files as QXL.WIN disks, SMSQmulator will check for you that the file exists, and complain if it doesn't. You can shut this complaint off here. Note that in some cases (notably when you want to format a QXL.WIN file, you will **have to** configure a non existing file as a QXL.WIN file, since SMSQmulator refuses to format an already existing file.

→ *Warn when a WIN drive is full*
When you try to copy files to, or otherwise fill up, a WIN drive, you will, of course, get a “drive full” message from SMSQE. Before that, SMSQmulator can pop up a warning message stating that this will happen. This is useless in most cases, but sometimes useful to me. Here you can switch this message off.

→ *Warn if a WIN drive is read-only*
A win container file might, for some reason, be read only. SMSQmulator will warn you if that is the case – here you can switch this warning on or off.

→ *Warn on sound problems*
SMSQmulator will normally pop up a warning window when the sound can't be initialised, or there is no sound volume control. You can suppress this warning here.

→ *Warn if an FLP drive is read-only*
An FLP image file might, for some reason, be read only. SMSQmulator will warn you if that is the case – here you can switch this warning on or off.

→ *Warn if an FLP image file can't be opened*
When you configure SMSQmulator to use certain files as FLP image files I t will check for you that the files exist and can be opened, and complain if it doesn't. You can shut this complaint off here.

Configuration with some basic keywords

Keywords such as NFA_USE also make some configurations possible. See below in the section called "Basic Keywords".

Configuration with the standard (menu) config program

The SMSQE file can and should be configured with the standard config program, (generally MenuConfig from Jochen Merz). The configuration items should be self-explanatory.

You can configure the following categories:

- SMSQ
- WMAN
- WMAN system colours
- Hotkey system II

These are the usual SMSQ configuration items. One item of note in the SMSQ section is the item "**Use IEEE floating point routines**". If you answer "yes", SMSQE will use the faster floating point routines – see the section on Faster floating point routines.

There are also other categories:

- NFA Drives
- WIN Drives
- SFA Drives
- FLP Drives.

  Please be advised that, for the time being, **configuring the items in these latter four categories will not result in any change**. The values you configure will be overridden by those in the ini file, or by SMSQmulator's default values.

XI - Basic keywords

There are some new Basic keywords that come with SMSQmulator's version of SMSQE. This section only describes keywords which are peculiar to SMSQmulator, or work differently from what might be expected. The order in which they are listed here is simply chronological: by going to the end of this section you can find the latest new keywords.

NFA_USE

This takes two forms which do very different things:

NFA_USE *name$:* this sets the devices usage name
and
NFA_USE drive, directory*$:* this sets the directory a drive refers to.

1. The first form sets the usage name of the entire device

Please read the “Configuration” section about the usage name for a device.

Unchanged name:
DIR win1_ → not found (there is no win device)
DIR NFA1 _
(files list….)
NFA_USE “win”
DIR NFA1_ → not found
DIR win1_
(files list...)

Please note:

1 – if you call the device WIN and you have a file called “boot” on win1_, SMSQE will attempt to boot from that file.

2 – If you set the usage name through the config item, this will only take effect the next time you reboot the machine. Setting it through the basic keyword is immediate, but will be forgotten the next time you reboot the machine (the info is NOT stored in the .ini file when using the basic keyword).

2. The second form sets the native directory to be used for a drive

DIR NFA1_ → not found
NFA_USE 1,”/home/whatever/smsqe” (or **NFA_USE** 1,”c:\smsqe”)
DIR NFA1_
(files list….)

Since SMSQmulator version.2.21, setting the file names, either through the config item or through the basic keyword is immediate, and the setting is stored in the .ini file.

WARNING:
Moreover, please make sure that you have closed all files on the drive before changing the directory name for it.

For example:
DO NOT do the following:

NFA_USE 1,"/home/whatever/smsqe"

t%=**FOP_NEW** ("nfa1_testfile") (opens /home/whatever/smsqe/testfile)
….
(#t% isn't closed)
NFA_USE 1,"/home/another directory" (where does #t% point to now?)

t% will most likely still point to the same file as it was when it was originally opened. But this is not guaranteed.

Always close all files on a drive if you make it point to another directory. File corruption may occur if you don't.

NFA_USE$

This function returns the name of the directory (if any) currently set for a drive.

NFA_USE$ *drive* where drive is the drive to query (1 … 8). Example:
PRINT NFA_USE$(2) might print "c:\users\qxl.win"

SFA_USE and SFA_USE$

Please refer to the NFA_USE section above. The commands are the same (except for SFA_ instead of NFA_), and apply to the SFA device, not the NFA device.

WIN_USE

This takes two forms which do very different things:

WIN_USE *name$:* this sets the devices usage name
and
WIN_USE drive, file*$:* this sets the QXL.WIN file a drive will refer to.

1. The first form sets the usage name of the entire device

Please read the "Configuration" section about the usage name for a device.

Stating with an unchanged name, the following commands hve the following effect:
DIR win1_ → (files list)
WIN_USE “QXL”
DIR win1_ → not found (there is no win device)
DIR qxl1 _ → (files list)

WIN_USE “win”
DIR qxl1_ → not found
DIR win1_ → (files list.)

Please note:

1 – if you call the device WIN and you have a file called “boot” on win1_, SMSQE will attempt to boot from that file.

2 – If you set the usage name through the config item, this will only take effect the next time you reboot the machine. Setting it through the basic keyword is immediate, but will be forgotten the next time you reboot the machine (the info is NOT stored in the .ini file when using the basic keyword).

2. The second form sets the native directory to be used for a drive

DIR win1_ → not found
WIN_USE 1,”/home/whatever/smsqe/QXL.WIN” (or **WIN_USE** 1,”c:\smsqe\QXL.WIN”)
DIR win1_ → (files list….)

Since SMSQmulator version.2.21, setting the file names, either through the config item or through the basic keyword is immediate, and the setting is stored in the .ini file.

Please read the warning under the NFA_USE command. It also applies, *mutatis mutandis*, to this command.

FLP_USE

Please refer to the WIN_USE section above. This command is the same, but applies to the FLP device, not the WIN device, and the files in question are .img files, not QXL.WIN type files.

The effect of the various xxx_USE keywords on each other

As seen above, the xxx_USE keywords, e.g. WIN_USE “WIN”, will make a device respond to a name other than its original name. After a WIN_USE “WIN”, the WIN device will only respond to being called WIN.

However, what happens if you use, say, FLP_USE “win” followed by WIN_USE “win”? With the first command you tell the FLP device to call itself win and with the second, you tell the WIN device to call itself win. Which of them will ... win?

There is a hierarchy built into SMSQmulator and a higher device will always trump over a lower device. This hierarchy is as follows:

low FLP
RAM
NFA
SFA
MEM
high WIN

So a WIN_USE will trump over all others, a MEM_USE will trump any others except for WIN etc…

FLP_DENSITY

Sets the “density” of floppy disks. Please refer to the FLP driver section above.

FLP_DRIVE

Sets the name of the native image file to be used for a floppy drive. Please refer to the FLP driver section above.

FLP_DRIVE$

Gets the name of the native image file to be used for a floppy drive. Please refer to the FLP driver section above.

FLP_xxxx

These keywords will probably have no effect. Please refer to the FLP driver section above.

DISP_TYPE and other DISP_xxxx commands

The (standard SMSQE) function DISP_TYPE returns the type of display/colour mode you are using: 0 for QL colour mode or 32 for 16 bit mode.
None of the other SMSQE DSIP_XXXX commands have any function under SMSQmulator – they will just do nothing, but won’t return an error.

KBD_TABLE

This standard SMSQE keyword has no effect under SMSQmulator. The keyboard will reflect the keyboard configured for Java.

SOUNDFILE

Use: **SOUNDFILE** "file_name"[,rep%]

Loads and plays this sound file. Please note that the file is not loaded into memory all at once. Hence, it seems desirable to load it from a fast device (e.g. a hard disk) to avoid the music being broken up. The quotes are necessary unless it is a string variable.

As of version 2.20, the rep% parameter means that, when the end of the file is reached, the file will be replayed again as often as indicated by the rep% parameter. So if the parameter is 1, it will be replayed once again, which means that it will have been played twice in total.

SOUNDFILE2

Use: **SOUNDFILE2** "file_name"[,rep%]

Does just about the same as SOUNDFILE, but the sound is played through another job created just for this. This means that the command comes back immediately after the sound playing job has been set up – it allows your program to continue whilst the sound is being played.

The sound playing job is owned by the job issuing the SOUNDFILE2 command, so if you remove that job, the sound playing job will be removed, too.

As of version 2.20, the rep% parameter means that, when the end of the file is reached, the file will be replayed again as often as indicated by the rep% parameter. So if the parameter is 1, the sound will be replayed once again, which means that it will have been played twice in total.

SOUNDFILE3

Use: **SOUNDFILE3** "file_name"[,rep%]

Is the same as SOUNDFILE2, but the job playing the sound is totally independent of the one issuing the command.

As of version 2.20, the rep% parameter means that, when the end of the file is reached, the file will be replayed again as often as indicated by the rep% parameter. So if the parameter is 1, the sound will be replayed once again, which means that it will have been played twice in total.

For all of these jobs, please note that the sound may continue to play for a few seconds after the soundjob is killed, as there is an internal buffer. Use KILLSOUND (below) to stop it.

KILLSOUND

Kills (stops) the sound.

Please note that this also removes the first job called "SOUNDFILE JOB" that it can find. NORMALLY, this should be the only sound playing job that runs in the machine. It is indeed not advisable to have several jobs all trying to make sounds in the machine, since the sound will be totally intermingled and garbled!

So having multiple sound sources is not a good idea…

JVASGET

Gets the content of the clipboard into the SCRAP.
See the section called “The System clipboard and the Menu Extensions' SCRAP thing”.

JVASPUT

Puts the content of the SCRAP into the clipboard.
See the section called “The System clipboard and the Menu Extensions' SCRAP thing”.

JVAMBAR

This makes sure that the menu bar of the main Java application is visible again, after you have hidden it with the corresponding config menu item in that menu bar.

JVAVOL

With this command, you can set the volume of sounds SMSQmulator can produce. The range of values is 0 to 100, with 100 being the loudest and 0 being no sound at all. Any value that exceeds this range is silently adjusted to fit.

JTMRSET

Sets (starts) a timer. The timer normally has a precision of 1 millisecond.

JTMRGET

This is a function which gets the number of milliseconds elapsed since the timer was last set. This is returned by Java as an unsigned long word (but SMSQE will convert that into a signed long word), so make sure that the timer period fits in such a long word. As a guideline, anything under 590 hours will be OK.

MACHINE

The standard SMSQE MACHINE keyword will tell you on what kind of machine SMSQE is running: 30 is QPC, 10 is a GoldCard, 12 a Super GoldCard, 17 a Q60 etc.

For SMSQmulator this value is 20.

WM_MOVEMODE

The WM_MOVEMODE is a standard SMSQE keyword that determines how the system behaves when you move SMSQE windows around:
The keyword is followed by a parameter from 0 to 3:

0 – This is the old behaviour, you move the little icon around.
1 – You move the outline of a box around.
2 – You move the window itself around.
3 – You move the window itself around and the window is made semi-transparent..

Option 3 is new and only available on newer versions of SMSQE. SMSQE comes with option 0 pre-set.

WM_MOVEALPHA

This determines how much the window to move around will be transparent. The keyword takes one parameter: a number between 1 and 255. : 0 means the window is totally transparent (you won't see anything of the move operation!), 255 means that it is totally opaque (same as move mode 3). The preset value is 128. Some prefer 180.

MEMD_WRITE

Please see the section on the MEM device for explanations about this procedure.

JVAQLFP

Switch back to using the normal SMSQE floating point routines See the section on Faster floating point routines.

JVAIEFP

Use faster floating point routines. See the section on Faster floating point routines.

JVA_SCRUPDT

Use this command to set the screen update rate:

JVA_SCRUPDT interval
where interval is the interval in milliseconds: every *interval* milliseconds the screen will get redrawn. See Screen update rate for more explanations. Interval is limited to values from 0 to 65535, but is read in as a word, so for values higher than 32768, you need to calculate the negative value that would correspond to. Any value exceeding a limit will be set to the closest limit.

JVA_MBAR_STATUS

This function returns the status of the menu bar : 0 if the bar is invisible, 1 if the menu bar is visible, and -1 if, for some reason, it was not possible to get the status of the menu bar.

JVA_EXIT

This leaves the emulation entirely, may be useful in full screen mode.

JVA_QLSCREMU

Switch QL screen emulation on/off. Please see the section on QL screen emulation.

JVA_VER$

This function returns the version number of SMSQmulator as a string.
version = **JVA_VER$**
PRINT JVA_VER$ might yield “2.20”.

JVA_NETNAME$

This function returns the current network name, or the current name, of the computer SMSQmulator is running on. It is possible that under some circumstances/machines this cannot be found - you will then get the string “Unknown name”.

JVA_WINDOWTITLE

Changes the title of the window, which normally is “SMSQmulator v. xxxx”.
Example:
name$=”QL”
JVA_WINDOWTITLE name$

WIN_DRIVE

This is the same as rhe second form of the WIN_USE command. It sets the native file used for the qxl.win container.
WIN_DRIVE drive,native_name.
Example:
DIR win1_ → not found
WIN_DRIVE 1,”/home/whatever/smsqe/QXL.WIN”
(or **WIN_DRIVE** 1,”c:\smsqe\QXL.WIN”)
DIR win1_ → (files list….)

WIN_USE$, WIN_DRIVE$

These two functions are the same. They return the name of a native file used as the qxl.win container.
Example: if you have defined the file “/home/you/qxl.win” and win drive 1, then
PRINT WIN_DRIVE$(1) will print “/home/you/qxl.win”.

Since SMSQmulator version.2.21, setting the file names, either through the config item or through the basic keyword is immediate, and the setting is stored in the .ini file.

JVA_POPUP

This keyword, which takes no parameters, causes a minimised SMSQmulator window to pop open again, or to blink. See the section on Popping up the window.

Alphabetical index of SBasic keywords

XII - Beeps and sounds

Generally speaking, sound isn't SMSQmulator's forte. Under some (Linux) systems, SMSQmulator for Java 6 just doesn't produce any sound. For a better sound experience on these systems, try to use SMSQmulator for Java 7. SMSQmulator can produce a warning window if it finds that a sound problem may exist. You can configure whether this warning should pop up or not (see the section on configuration, warnings).

1. Beep

SMSQmulator tries to emulate the QL beep sounds. This is far from perfect. The sound will often be too clean. Moreover, the "`wrap`", "`random`" and "`fuzziness`" parameters to the **BEEP** command are simply ignored.

Also, the BEEP behaviour may vary from machine to machine, OS to OS. On some machine/OS combinations, a beep of a duration of less than 6667 (e.g. BEEP 6666,10) will not produce any sound, whereas BEEP 6667,10 will.

2. SSSS

SMSQmulator also tries to emulate the SSSS, as found in QPC & the Q40/Q60 systems. Unfortunately, there is a bug in the Java SourceDataLine component, which means that this emulation is not perfect.

This bug, unfortunately, particularly raises its ugly head when using the original versions of Simon Goodwin's SOUND driver. I stress that this isn't due to Simon Goodwin's driver, but it is due to the way the Java sound queue is implemented: often the last sound(s) played will be repeated indefinitely.

Instead of using Simon Goodwin's SOUND driver, use the one provided directly by SMSQmulator (see just below, the SOUND device) where no additional file needs to be loaded.

If you use the SSSS you might want to use the sound keywords provided herewith, then the risk of garbled sound is minimised (see above the SOUND.xxx keywords in the basic section).

Please have a look at the configuration for the SSSS frequency.

3. The SOUND device

SMSQmulator incorporates (as of version 1.23) its own SOUND device. No additional file needs to be loaded and DO NOT load Simon Goodwin's SOUND driver.
The SOUND device is patterned on Simon Goodwin's SOUND driver, and behaves just like it, with some exceptions:

- Only one channel may be open to it at one time.

- The only operation allowed is to send it multiple bytes.

This still caters for the most frequent use case, i.e. copy a _ub file to the device, such as: COPY <filename>,SOUND2.

As with the original SOUND device, this one has 9 different types of files it can accept, (SOUND1 to SOUND9). Copying to “SOUND” (without number at the end) will default to SOUND1. The types are:

1 – 20 khz mono (default)
2 – 20 khz stereo
3 – 10 khz mono, with averaging
4 – 10 khz stereo, with averaging
5 – 40 khz mono, with averaging
6 – 40 khz stereo, with averaging
7 – 40 khz mono, every second byte is skipped
8 – 40 khz stereo, every second byte pair is skipped
9 – 40 Khz mono send alternate bytes left/right

XIII - The System clipboard and the Menu Extensions' SCRAP thing

If you have loaded the Scrap thing which is part of Jochen Merz Software's Menu Extensions, then you can get **text data** (nothing else) from your native OS's clipboard into the SCRAP thing and vice-versa.
This is done via simple Basic commands:
JVASGET gets the content of the clipboard into the SCRAP. This overwrites any text content that is already in the SCRAP.
JVASPUT puts the content of the SCRAP into the clipboard. This overwrites any text content that is already in the clipboard.

Creating a simple Hotkey to get the clipboard into the SCRAP

Create a basic program which you will call, say “win1_basic_cbstuff”, with the following content:
10 **JVASGET**
You can then create a hotkey as follows (in this example, the hotkey will be ALT+x keystroke):
ERT HOT_THING (“x”,'sbasic';'lrun win1_basic_cbstuff'&CHR$(10))
Now, every time you press the ALT+x hotkey, the content of the system clipboard will be copied into the SCRAP.

Creating a simple Hotkey to get the clipboard into the HOTKEY Stuffer Buffer via the SCRAP

You could, for example, do the following to get the content of the clipboard into the Hotkey stuffer buffer (from where you can retrieve it via ALT + Space, or F12 under windows):

Create a basic program which you will call, say “win1_basic_cbstuff”, with the following content:

```
10 JVASGET
20 a$=SCRAP_GET$(0)
30 HOT_STUFF a$
40 QUIT
```

Now create a hotkey as follows (in this example, the hotkey will be ALT+x keystroke):
ERT HOT_THING (“x”,'sbasic' ; 'lrun win1_basic_cbstuff'&CHR$(10))
Now, every time you press the ALT+x hotkey, the content of the system clipboard will be copied into the stuffer buffer

If you're, really adventurous, you could even add this line to the basic program:
35 **HOT_DO**(“ “) : rem this is a space between quotes
Now, every time you hit ALT+x, the content of the clipboard will be copied directly into the underlying program.

Of course, the **JVASPUT** command can be used in an analogous manner.

Remember, however, that the size of the Hotkey stuffer buffer is limited, so don't be astonished if not all of a large text is copied into the stuffer buffer.

XIV - QL Screen emulation

As of version 2.20, SMSQmulator can also emulate a basic QL screen when using a higher screen. This means that SMSQmulator will copy bytes written to the QL screen (from $20000 to $28000) to the display that it is actually showing.

Whenever the display SMSQmulator currently is using is a QL colour type screen (i.e. mode 4 or 8), the bytes are simply copied. If it is a higher colour screen, then colour conversion is made.

To use screen emulation, you must first of all configure it through the configurations menu, there is a new configuration item “Allow QL screen emulation”. Set the tick box before this to allow the screen emulation. You must then restart SMSQmulator entirely. You now have a version of SMSQmulator that is ready to accept QL screen emulation, and you can switch this on and off.

Screen emulation is then switched on and off as follows:

JVA_QLSCREMU *mode*

where *mode* can be :

0 – screen copying is switched off
4 – the QL screen is supposed to be in QL mode 4.
8 – the QL screen is supposed to be in QL mode 8.
(-1- automatic mode, uses the mode set by the most recent MODE command. NOT IMPLEMENTED YET)

If the screen SMSQmulator currently is using is a QL colour type screen, the mode parameter, whilst still being compulsory, is ignored – the QL screen will be copied in the mode the screen is in.

Note : Using this facility will have a definite impact on performance, especially on slower machines. This is why you have to configure it and restart the emulator: it then uses another CPU core emulation, which is slower than the normal one.

XV - Popping up the window

Ad of version 2.21, SMSQmulator contains a new keyword, JVA_POPUP. When used, this keyword will cause a “popup event”. This is only useful when the entire window of SMSQmulator has been iconified (i.e. minimised to the taskbar). Upon receiving such an event, and if the window is iconified, SMSQmulator will either re-open the window to its normal state, or blink the taskbar entry, depending on the configuration option you have chosen.

This could be useful, for example if you launch a long-ish calculation and want to be notified when that’s done.

JVA_POPUP has no effect if the window is not iconified.

Moreover, for this to work, the emulation obviously must not be suspended when the window is minimised.

XVI - Key mappings

Some keys on the PC keyboard don't have corresponding keys on the QL keyboard. I've tried to map the keys intuitively, but some might require explaining:

Key	Resulting QL keystroke
F6 – F10	SHIFT-F1 to SHIFT-F5
Break	CTRL-SPACE
Scroll lock	CTRL-F5
Home	ALT-Left
End	ALT-Right
Page up	SHIFT-Up
Page down	SHIFT-Down
F11	not used
F12	same as ALT-SPACE (on some systems ALT-SPACE is trapped by the OS/window manager).

XVII - What to do when things don't work: error reporting

It is always possible that programs won't work on SMSQmulator. Please report any such programs to me.

1. Please note that SMSQmulator emulates a 68000 processor, not a 68010/20/30/40/60 processor.

That means that programs specifically written for higher processors will not work under SMSQmulator – this is the case of programs marked, for example, as “for the Q40/Q60”. George Gwilt's Gwass also falls under this category.

2. Old Games

Many old games will not work. Most of them seem to make assumptions about the machine they run on, especially the size of the memory, the placement of the system variables etc... SMSQmulator is an SMSQE machine, not a QL Emulator. The best bet to try old games is to put the emulator into a state as close to the QL as possible: use QL colours and a screen of 512 x 256 pixels. I'd be interested to hear about games that DO run. Please also see the section on emulating the QL Screen.

3. Error reporting

Of course, I'd like all errors to be reported to me. For this to be of any use, however, try to reproduce the error on a "naked" machine, as much as possible.

(---*If you're interested, here is why:*

SMSQE is a pretty stable OS. I've only made very few changes, which normally are well tested.

This means that an error will most probably not come from SMSQE itself, but from my own Java code including the 68008 emulation which gets an instruction wrong,... If it is the latter, this means that I have to spend hours (literally) tracing the code instruction by instruction to find out which instruction was in error (as an indication: Who would have thought that, if the keyboard history told me "buffer full", this was due to the ROXR instruction being wrong???...).

So, the more easily the error can be reproduced, the less code I'll have to trace through…--)

XVIII - Compiling SMSQmulator

Compiling the SMSQE part

The source code to SMSQE comes in a file called smsqesrc.zip (or smsqesrcXXX.zip where XXX is a version number) and can be downloaded from the SMSQE website (www.wlenerz.com/smsqe). This is actually a small QXL.WIN type file and will decompress to a file called smsqesrc.win or smsqesrcXXX.win.

You can compile and build SMSQE from that. For this, proceed as follows:

Bind that QXL.WIN file into your system as winWhatever_, and have dev8_ point to it (i.e. **DEV_USE** 8,winWhatever_): All of the sources presume that they are on dev8_.

Make sure that you have QMAC and the corresponding linker in the PROG_USE directory. Once you've done that, **EXEC** the following program:

EXEC dev8_extras_exe_SMSQEMake

Once this program is running:
select "generic"
select "Java"
select "all"
select "make"

and hit OK.

Wait until everything is compiled.

The resulting file will be called “dev8_smsq_java_java”. This is the SMSQE file SMSQmulator needs. Copy it to somewhere where SMSQmulator can get at it.

(NB. You can also recompile the SMSQEmake program. For that you need Qliberator and also **LRESPR** the following extensions:

- dev8_extras_cline_bin
- dev8_extras_exe_source_outptr_bin
- dev8_extras_exe_source_SMSQEMake_bin

Then compile the program in dev8_extras_source_SMSQEMake_bas with Qliberator.

Building the Java part

All Java related source files are in two zip files: SMSQmulatorsrc.zip (or SMSQmulatorXXXsrc.zip where XXX is a version number) and Inifile.zip.

Each zip file, when de-zipped, creates a directory with the same name as the zip file (minus the “.zip” extension). Each zip file contains the source code (“src” sub-directory) and, for the “Inifile” object, also the “executable” code (“dist” directory).

The source code is in “Netbeans” format. Should you use other IDEs, such as Eclipse, you must adapt the project to your IDE. More modern versions of SMSQmulator no longer provide the “nbproject” folder but you can create that easily by creating a new project called SMSQmulator.

You have to make sure that the SMSQmulator project has the Inifile.jar and AppleJavaextensions.jar on its path. The "lib" sub-directory in any of the SMSQmulator executable versions already contains these jar files. Attention: Depending on your IDE, if you recompile the project, the jar files might disappear from the "dist" sub-directory, so save them to a safe place before you recompile the project. If you rebuild the project from source, you'll have to point your IDE to the jar files for inclusion (e.g. as a library in Netbeans) in your project.

There is also a Javadoc for the different SMSQmulator classes in the "doc" sub-directory.

XIX - Known Bugs, quirks & desirable enhancements

This version contains certain known bugs and quirks:

Bugs/things that need be done – if possible

1. Imperfect keyboard emulation:

On a mac:
If you're an Apple user, the ALT key is replaced by the CMD key. This is a feature, not a bug. CMD + Q doesn't work. This is not a feature.

Under Linux:
Different desktop environments / window managers will trap certain key combinations, before they even get to my application. There isn't much I can do about that, since the keystrokes never reach my application. Under some window managers, you may be able to switch off the desktop environment / window manager snatching the keys.

For example:
KDE traps the CTRL-F1 to CTRL-F4 keys, as they switch between the different virtual desktops. You can get rid of that by invoking the system settings menu (e.g. type “systemsettings” from a terminal), then go to “Shortcuts and gestures”, in there navigate to “Global Keyboard Shortcuts” and choose “Kwin” from the KDE component dropdown menu. Look through the list until you find the CTRL-F1 etc. keystrokes and modify them.

Under (L)Ubuntu 14.04, Ibus traps the CTRL-SPACE keystroke. To remove this shortcut, run "ibus-setup" in a terminal. In the window that opens under “keyboard shortcuts” choose the “next key” to be space and delete the space – control shortcut. (Remember that you can also use the pause/break key for CTRL-SPACE, see above Key mappings).

Quirks

Menu Extensions not showing all devices
It has been noticed that some of the menu extensions can only show 6 file device driver devices. Since SMSQmulator already comes with 7 (WIN, FLP, RAM, MEM, DEV, NFA and SFA), one of these is not going to show up. This is a design feature of the menu extensions and not peculiar to SMSQmulator. Remember that you can disable some devices (WIN, MEM, NFA, SFA, FLP).

Menu Extensions showing strange sub-directory names

If you have more than 6 file devices and use the directory select menu extension, you will notice that some sub directories of the 7th device may be prefixed with the "_" character. This is a problem in the menu extensions application window items hit routine (which adds a "_" to all directory names but not if the files lie on the 7th or more device) and not peculiar to SMSQmulator.

Directory select Menu Extension may crash your machine

If you have more than 7 file devices and use the directory select menu extension, you will crash your machine (sysmon will start howling). This is a problem in the menu extensions and not peculiar to SMSQmulator.

Caps Lock indicators may not always work

On some systems caps lock indicators may not work correctly, due to some Java bug.

Time is relative

SMSQmulator has a nominal 50 Hz clock, i.e. a "hardware" interrupt is nominally generated every 20 milliseconds. However, please be advised that this is only an average: there is no guarantee that a clock tick will actually happen exactly every 20 ms. There can be a delay between 2 clocks ticks, which is then "caught up" by a smaller delay between the next clock tick(s). However, on an average over, say 80ms, there should be 4 clock ticks.

No Address Error

Normally, if you try to read/write a word or a long word from an odd address, you should get an address error. SMSQmulator does not throw an address error in that case, the read/write operation takes place at the address-1.

Don't change the jar file extension

The main SMSQmulator file is called SMSQmulator.jar. You may, of course, change that name to anything you like, but please keep the filename extension (".jar"). Under some circumstances, SMSQmulator may not work correctly if you don't.

Desirable enhancements

PAR, SER drivers (probably never gonna happen).
TCP more calls, UDP (maybe).

XX - Credits, licence and copyright info

Credits

This project only was possible because I stood on the shoulders of giants.

Credits must go to:

- Tony Headford for his M68K emulator

and of course to

- All of you who wrote SMSQE! You know who you are.

I'd also like to thank Timothy Swenson for his tireless efforts in hunting (and finding!!!) so many bugs.

Marcel Kilgus has pointed out numerous bugs (and ways to fix them), including in the emulation core – thanks.

Thanks also to

- Tobias Fröschle for bug hunting and sharing code for the Mac.
- Bob Spelten for bug hunting.
- Urs König for bug hunting and spellchecking.
- David Westbury for also finding quite some bugs.
- Giorgio Garabello for the italian translation.
- Marcos Cruz for bug hunting and spanish text fixing.

Copyright and Licences

- *Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.*
- *Neither the name of SMSQmulator nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.*
-

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDERS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
”

SMSQE is copyright © its respective copyright holders and is published under the SMSQE licence, see the “SMSQE licence.txt” file in the “lib” sub-directory.

M68000K is copyright © Tony Headford and is published under Tony Headford's licence, see the “M68000K licence.txt” file in the “lib” sub-directory.

The AppleJavaExtensions are copyright © Apple, Inc and are published under the licence found in the “Apple Java Extensions licence.txt” file in the “lib” sub-directory.

The SpringUtilities are copyright © Oracle and are published under the licence found in the “Oracle licence.txt” file in the “lib” sub-directory.

XXI - Versions of this manual

From time to time this manual gets updated. The (first) version to which each updated edition of this manual applies, is found at the bottom of each page. If a new version of SMSQmulator comes out which doesn't need a new manual, the version of the manual is NOT changed.

As of v. 2.00, here is a small recount of what has changed in this manual. This will most probably also point to new features introduced with that version of SMSQmulator:

2.21 JVA_VER$, JVA_NETNAME$, JVA_WINDOWTITLE descriptions added. IP Devices should work with most basic TCP/SCK calls, WIN_DRIVE and WIN_DRIVE$ added. WIN_USE$, NFA_USE$ and SFA_USE$ explanations added. Setting the file/directory names of devices, either through the config item or through the basic keyword is immediate, and the setting is stored in the .ini file. (This has already been the case since some versions ago). Unlockable files may be made read-only. Alphabetical index for the SMSQmulator Sbasic keywords. SSSS may use resampling. Popping up the window via JVA_POPUP explained.

2.20 QL screen emulation and associated configuration item descriptions added. SSSS frequency configuration : may be set to one of two values, SOUNDFILE parameter changed.

2.18 Different window modes can be set, JVA_EXIT introduced, multi-monitor config item, compiling updated, restarting and resetting updated (including new “Exit” item in the “File” menu), which java version updated, more hyperlinks throughout.

2.17 SMSQmulator starts in 16 bit colour mode if nothing configured, setting the colour mode takes effect immediately and reboots, cursor blink pause abandoned.

2.16 JVA_SCRUPDT and JVA_MBAR_STATUS introduced.

2.15 Don't change the jar file extension warning.

2.14 No address error explained.

2.13 Warning when using NFA_USE and SFA_USE.

2.08 Using the new floating point routines is now standard.

2.07 Small corrections, cosmetic changes. Added a section on faster floating point routines.

2.05 Pause after mouse click introduced.

2.04 New configuration item: ignore QWL.WIN lock file error

2.03 Key mappings explained; key problems explained under bug and quirks.

2.02 Warnings if flp drive is read only / can't be opened, section on floppy disks

2.01 Section on floppy disks – you are expected to behave rationally.