SMSQ/E FOR THE Q68

Contents

Introduction

This manual explains the SMSQ/E particulars as they pertain to the Q68. It does not explain SMSQ/E in general, there is a good manual for SMSQ/E on Dilwyn Jones' site at http://www.dilwyn.me.uk/docs/smsqegd2/index.html.

1 SDHC cards

The Q68 uses SDHC memory cards for mass storage. These must be SD**HC** cards – simple SD cards are not compatible. There are two card sockets into which you can insert the cards. The left card socket is socket 1, the one on the right is socket 2 (like mdv1_ and mdv2_). From here onward, the card inserted into the left socket will be called **card1**, the card inserted into the right socket will be called **card2.**

For the cards to be useful, they must be partitioned and the **first** primary partition must be formatted in FAT32 format (this cannot be done on the Q68). The different files the Q68 needs must be put into that partition, which should be possible from any machine running an OS that can read/write SDHC cards (Linux, Windows, macOs): just copy the files to the card.

The Q68 always tries to start up from card1, by loading the operating system from that card. Once SMSQ/E is loaded, it will follow its own usual boot process, normally running the boot file found on win1_.

1.1 Using container, OS and other files on the card

Under SMSQ/E, the Q68 uses qxl.win type container files as the main mass storage devices. These files must lie in the first primary partition on the SDHC card which must be formatted with a FAT32 file system. Moreover, these container files must be located within the first 16 directory entries of this FAT32 formatted partition. This is also true for the file containing SMSQ/E itself.

Special precautions must be taken when writing the container and OS file(s) themselves to the card. Indeed, SMSQ/E expects a container file not to be fragmented in the FAT32 file structure. It assumes that, once it has found the beginning of a container file, the rest of that container file lies in contiguous sectors on the card. This is also true for the SMSQ/E binary files (named "Q68_RAM.SYS" or “Q68_SMSQ.WIN”) itself. Thus the files on the cards must not be fragmented.

The best way to achieve this is to make sure that, before writing the SMSQ/E binary file and the container files, the card is freshly formatted. Then write each container (or other) file, one after the other, immediately after formatting the card.

Hence, you should dedicate a card solely for the purpose of using it with the Q68.

Do not drag and drop several files onto the card at once. **Do not** delete files from the card – always format it. It is **very much** recommended that you read the section Avoiding fragmentation to make sure you treat the card as you should.

1.1.1 Naming scheme

The file name of a container or OS file MUST be in "8.3" format, i.e. a name of 1 to 8 characters, possibly followed by a decimal point and a three letter extension. Missing letters are filled up with spaces. The name and extension must be in upper case and the extension, if present, must be separated from the name by a period (".").

Please only use plain ASCII characters for the name and no accented characters, i.e. the letters A-Z and numbers 0-9.

In all commands or configuration items where you must give or configure a name, SMSQ/E tries to help you as much as possible. Names are automatically converted into upper case and correctly formatted, so that “qlwa.win” would automatically be converted to “QLWA   .WIN”. However, a “_” is not converted to a “.” .

See also the default names of OS and container files.

1.2 Initialising a card : CARD_INIT

With the Q68, before you can use drives on an SDHC card, the card must be initialised (it would actually be more accurate to say that the card reader a card is in must be initialised, i.e. put in a state where it will read a card). Card1 is automatically initialised at boot time. By design, card2 is not initialised at boot time, though this will depend on you configuration options. If it is not initialised, you have to initialise it yourself. You can do this with the supplied **CARD_INIT** command. The card itself is not touched by this command (it is not formatted, written to or anything).

Syntax:
CARD_INIT *card_number*

where *card_number* is the card to be initialised (1 or 2).

Example:

CARD_INIT 2.

Initialisation will fail with the error “medium check failed” if no card is in the drive.

1.3 Swapping cards

As a general rule, cards may be swapped in and out, even when the system is running - but this is not a recommended practice. However, if you insist on doing this, you must be aware of a few rules:

1 – Do not remove a card when there are files still open to a drive and certainly not whilst the machine is reading/writing to a card. If you remove a card whilst there are still files open or files being written, data loss WILL (not “may”) occur. Note that you will NOT be able to write the missing data to the card even if you reinsert it immediately after having

removed it from the socket. Some device drivers have a special keyword telling you whether it is safe to remove a card from its socket (see, e.g. WIN_SAFE).

2 – When a card is removed from its socket, the card reader in that socket becomes uninitialised. Before using the new card that you just inserted, you must initialise it, as described above. This is true whether you insert a new card or re-insert the old one that was just removed.

2 Win drives on SDHC cards

For the Q68, SMSQ/E uses “qxl.win” type container files as the main mass storage devices. These files must comply with the rules set out above.

The corresponding SMSQ/E device is called "WIN" and, potentially, you may have up to 8 different drives for this device, called "win1_" to "win8_".

Each WIN drive can point to one container file lying indiscriminately on SDHC card one or two. For each WIN drive, you must set the name of the container file, and the number of the card on which this file is to be found. You may do so by configuring this with the standard configuration program. You can also set the names of the container files at any time with the **WIN_DRIVE** command. Menuconfig is the best choice here.

2.1 Safety precaution

Do not point two different WIN drives to the same container file on the same card. For the time being, the system doesn't stop you from doing so, but data loss and file corruption WIN (not "can") occur as a result!

If in doubt, use the **WIN_DRIVE$** function to consult the list of container files already assigned to a drive.

2.2 Basic commands for WIN drives

The basic commands related to WIN drives are as follows:

2.2.1 WIN_DRIVE

assigns a container file on a card to become a drive.

Syntax:

WIN_DRIVE *drive, card, file_name$*

where:

- *drive* is the WIN drive number (1... 8) to be assigned.

- *card* is the SDHC card on which the file can be found (1 or 2).

- *file_name$* is the name of the container file. The file name MUST be in "8.3" format, i.e. a name of one to 8 characters, a decimal point and an extension of up to three letters. The extension, if present, must be separated from the name by a period (".").

Please note that this command does not check that the file is actually present and readable on the card.

The **WIN_DRIVE** command has an intended side-effect:

> If the command is applied to a card that isn't present (any more), all channels to all drives that would have corresponded to files on that card are closed, and the drive definition blocks for such drives are removed.
>
> This is a protection against a card being ripped out of the drive, a new one inserted and a write operation to the new card being made. That way, at least, no old information will be written to the new card.

Example:

WIN_DRIVE 2,6,"QXL.WIN"

make the file "QXL.WIN" on card 2 into WIN drive 6.

2.2.2 WIN_DRIVE$

This function returns the name of a container file corresponding to a drive, and the card on which this file should be found.

Syntax:

name$=**WIN_DRIVE$**(*drive*)

where

- *drive* is the drive to be questioned (1 to 8)

The function returns, as a string, the name of the container file assigned to the drive passed as parameter, as well as the number of the card this file is on (1 or 2), separated from the name by a comma. Please note that this does not tell you whether such a file actually exists on the card.

Example:

PRINT **WIN_DRIVE$**(6) might return "QL_WIN6.WIN,2"

this means that the drive win6_ is assigned to a file called "QL_WIN6.WIN" which is to be found on the card in socket 2.

2.2.3 WIN_SAFE

This command checks whether it is safe to remove a card as far as the WIN driver is concerned.

Syntax:

WIN_SAFE *card*

where *card* is the SDHC card on which the file can be found (1 or 2). If this command returns without error, then, as far as the WIN driver is concerned, the card may be safely removed. If not, it will return the error "is in use". Be patient then and retry a few seconds later.

Example:

```
WIN_SAFE 1
```

checks whether card 1 may be safely removed.

2.2.4 WIN_CHECK

Checks whether a WIN container file on the card is indeed in contiguous sectors on the card.

Syntax:

WIN_CHECK *drive*

where *drive* is the container file containing the WIN drive in question. If the command does not return an error, the container file corresponding to the drive is OK.

2.2.5 WIN_FORMAT

Please see below in the section "Formatting a drive".

2.2.6 WIN_USE and WIN_WP

These are the standard SMSQ/E keywords and work as expected.

2.2.7 WIN_START, WIN_STOP, WIN_REMV, WIN_SLUG

These typical SMSQ/E commands are not needed and do not exist.

2.3 Formatting a drive.

Formatting a drive means that an **EXISTING** file on a card is made into a qxl.win type container file. This is achieved with the usual FORMAT command:

FORMAT *winX_name*

where, as usual, "*X*" is the drive number and "*name*" the formatted name of the drive. This may be up to 10 characters long. Anything longer will be truncated to 10 letters. Please note that this is NOT the name of the container file on the card, just the name that will be displayed when doing a **DIR** on the drive.

Formatting a drive will irretrievably erase some of the content of the file on the card and make it into a qxl.win container file.

As a security precaution, before issuing the FORMAT command, you must issue the **WIN_FORMAT** command:

WIN_FORMAT *drive, yes_or_no*

where "*drive*" is the drive to be formatted, and "*yes_or_no*" is 1 for allowing the drive to be formatted and 0 for refusing to do so. Please note that this overrides any write protection you may have set for this drive with the WIN_WP command.

Trying to format the drive without issuing the **WIN_FORMAT** command first will fail.

Moreover the **FORMAT** command itself **MUST be issued from the main SBasic job**, i.e. job 0. This is because the **FORMAT** command will invite you, as an additional precaution, to enter two random letters into channel#0 of job 0. Failing to enter the letters will make the machine seem to hang. Entering different letters will make the **FORMAT** command fail. Even though the letters to be entered are displayed in upper case, you may enter them in lower case. If you issue the **FORMAT** command from another job than job 0, the machine may seem to have crashed - so don’t.

Example:
WIN_DRIVE 3,1,”example.win”
   set drive 3 to correspond to an existing file called EXAMPLE.WIN on card1.
WIN_FORMAT 3
   prepare to allow formatting
FORMAT “win3_WINDRIVE 3”
   format the file to become a qxl.win container.

You can only format an existing file. You cannot create a new container file on the card with the **FORMAT** command (you can do this with the **CARD_CREATE** command).

3 The FAT device

The FAT device is a FAT16 driver for the Q68 (and possibly other SMSQ/E systems). Its purpose is to allow you to exchange files easily between the Q68 (qxl.win type) container files and the outside world. It can read/write a FAT16 formatted partition on the SDHC card. This means that, in addition to the standard FAT32 formatted partition that an SDHC card must have to be recognized by SMSQE, there must be another partition, formatted in FAT16. If you only have one SDHC card, the FAT16 partition must lie AFTER the main FAT32 partition.

Please note that, by default, SMSQ/E may be configured NOT to load this driver at all. If you want to use this device, you might configure SMSQ/E first to use it.

3.1 Principle

You just use the FAT device like any other device. There are potentially 8 drives you may use (FAT1_ to FAT8_). You must however configure SMSQ/E to use the device at all and set the device particulars, either by configuration or with the FAT_DRIVE command.

3.2 Limitations

The FAT driver comes with a few caveats.

1 – The FAT driver can only handle standard DOS names (i.e. 8.3 names). None of the extended name schemes invented for FAT16 is used by the FAT driver. The presence of extended names doesn't harm the FAT driver's operation, it just won't use them – and when writing back to the partition, it might destroy the extended names (but not the normal ones nor the data!).

2 – The FAT driver can only handle FAT16 partitions formatted with a cluster size of 8 sectors per cluster and a sector size of 512 bytes.

Under Linux, this can be achieved with the command:
mkdosfs -F 16 -s 8 /(your device)

3 – The maximum partition size the FAT driver can handle is limited to 255 MiB.

A ready-made file containing a 1.5 GiB FAT32 partition and a 250 MiB FAT16 partition can be downloaded from "www.wlenerz.com/Q68/empty_image _file.zip". This is a compressed empty file under 3MiB in size which will be expanded to 1.8 GiB when decompressed.

Under Linux, you can use the "dd" command to copy that to an SDHC card, which will give you the necessary partition scheme to start using the SHDC card with the Q68. Under windows, you might use the win32diskimager freeware https://sourceforge.net/ projects/ win32diskimager/.

4 – Each FAT16 partition must be a primary partition, not be contained in an extended one (remember there can only be 4 primary partitions on an SDHC card. The first one must be the FAT32 partition, any of the others could be an extended one which the Q68 will not use).

5 – Formatting a FAT drive is not possible.

3.3 Configuration

In the "Fat drives" section of the configuration (see below) you can configure on what card and what partition the FAT drives should be.

3.4 Basic keywords

There are several keywords to be used with the FAT drive:

3.4.1 FAT_USE

You may use the usual FAT_USE keyword to set the usage name of the FAT device: **FAT_USE** “flp” will result in the device being called FLP instead of FAT.

3.4.2 FAT_DRIVE

With this command you can set, for any FAT drive, the card and partition it is to be found on.

Syntax:
FAT_DRIVE *drive, card, partition*
where
    drive is the drive to be set (1 to 8)
    card is the SDHC card it is on (1 or 2)
    partition is the primary FAT16 partition (1 – 4)

3.4.3 FAT_DRIVE$

This function will return the card and partition for which a FAT drive is configured.

Syntax:
result = **FAT_DRIVE$**(*drive*)
where *drive* is the drive number (1 to 8).

This will return a string formatted thus:
Card: <card_number>, Partition: <partition_number>

Example:
PRINT FAT_DRIVE$(1)

might return "Card: 1, Partition: 2".

If the drive isn't configured for any card, the card number will be "N" (for "None"). If the drive isn't configured for any partition, the partition number will be 0.

3.4.4 FAT_WP

Sets/removes a software write protection on the drive, just like the usual WIN_WP.

4 The QUB device

This device allows you to the read the first (!) partition of a container image file formatted the Qubide way. Hence, each drive corresponds to one container file on the card, This is just like the WIN drive. The purpose is mainly for you to be able to get data off the Qubide drive and onto a proper WIN drive. You should not operate a Qubide type drive as your main storage system, use the win drives for that..

So basically, this device behaves just like the WIN device, except that it uses different container files.

Please note that, by default, SMSQ/E may be configured NOT to load this driver at all. If you want to use this device, you might configure SMSQ/E first.

The device is called QUB and there are 8 drives. Like for the WIN device, you must indicate for each drive the name of the container image file and the card it is on. Again, sensible names have been preconfigured.

There are the following basic commands: **QUB_USE, QUB_WP, QUB_DRIVE** and **QUB_DRIVE$** which behave in a similar way to the WIN_xxx commands.

5 OS and Container filenames

There are several types of files that lie on an SDHC card during normal use of the Q68 with SMSQ/E. All of these must adhere to the 8.3 naming scheme set out above.

- The SMSQ/E file itself. This may come in two types :
  - (i) as a "naked" file, containing just SMSQ/E itself. In this case, the file **MUST** necessarily be called "**Q68_RAM.SYS**".
  - (ii) as a qxl.win type container file. In this case, the file **MUST** necessarily be called "**Q68_SMSQ.WIN**". This allows a much easier configuration of SMSQ/E, see below the section on configuration. If you use a container file for the OS, this must contain the file called "Q68_SMSQ" in the root directory, and this file MUST be the very first item in the root directory, even before any sub-directory. When delivered, the Q68 comes with a small (1 MiB) file called "Q68_SMSQ.WIN", which adheres to this rule. It contains some other software to allow you to configure SMSQ/E (see the configuration section). Or course, you may create your own container file(s) of any reasonable size and use it for the QS. Just remember that it must be called "Q68_SMSQ.WIN" and that the file called "Q68_SMSQ" must be the very first item in the root directory. Perhaps it is better to have a dedicated small container file just for the OS – this will make upgrading it much easier.
    Despite the fact that the "Q68_SMSQ.WIN" is a qxl.win type container file, there is no obligation to assign it to any of the win drives – the boot process is independent of win drives assignment. By default, however, SMSQ/E is set up to use this file as win8_.

  One of these two files must be present on card1, else SMSQ/E will not be booted. It is not possible to change the names of these two files - if you do, they will not be recognized as SMSQ/E files. If both are present, SMSQ/E will be loaded from the container file.

- The WIN device container files. These should be called **QLWA**x**.WIN** where x can be any number between 0 and 9999, or be omitted. It is recommended, but not mandatory, that you stay with this naming scheme. Since the names of the container files are configurable, you may basically call them whatever you like, provided you adhere to the 8.3 naming rules. SMSQ/E comes pre-configured with what the designers of the Q68 think are sensible names and values. You can configure these names with the usual configuration program, Menuconfig is the best choice here. You can also use the WIN_DRIVE command.

- The QUB device container files. These should be called **QL_BDI**x**.BIN** where x can be any number between 0 and 99. Again, it is recommended but not mandatory that you stay with this naming scheme. However, since the names of the container files are configurable, you may basically call them whatever you like, provided you adhere to the 8.3 naming rules. There again, a sensible default naming scheme has been devised:

- QL_BDI.BIN on card1 for qub1_
- QL_BDI.BIN on card2 for qub2_
- QL_BDI3.BIN and QL_BDI4.BIN on card1 for qub3_ and qub4_.
- QL_BDI5.BIN to QL_BDI8.BIN on card2 for qub5_ to qub8_.

Please remember that the qub device should not be your main storage system for the Q68.

6 Setting the screen modes

The Q68 has several different screen modes presenting different screen sizes and colours. Please note that the more colours are displayed and the higher the resolution, the slower the Q68 will become.

6.1 DISP_MODE

You may set the Q68 screen modes with the **DISP_MODE** command:

DISP_MODE *mode*

where mode can take the following values:

0 = QL 8 colour mode
the standard 512 x 256 pixels mode in 8 colours. In this mode you can also set mode 4, with the usual MODE keyword. This is then equivalent to setting DISP_MODE 1.

1 = QL 4 colour mode
the standard 512 x 256 pixels in 4 colours mode. In this mode you can also set mode 8, with the usual MODE keyword. This is then equivalent to setting DISP_MODE 0.

2 = Small 16 bit mode
512 x 256 pixels in mode 33 (16 bits per pixel).

3 = Large 16 bit mode
1024 x 512 pixels in mode 33 (16 bits per pixel). Please note that this mode will slow down the Q68, you should not use this mode when doing something time-critical.

4 = Large QL Mode 4
1024 x 768 pixels in QL 4 colours mode (there is no mode 8 in this display mode).

5 = Aurora compatible 8 bit colours
1024 x 768 pixels in Aurora 256 colours mode. This allows you to have a big screen with nicer colours while still being reasonably fast (but slower than the QL modes).

6 = Medium 16 bit mode
512 x 384 pixels in mode 33 (16 bits per pixel).

7 = Very large 16 bit mode
1024 x 768 pixels in mode 33 (16 bits per pixel). Please note that this mode will severely slow down the Q68, you should not use this mode when doing something time-critical.

6.2 DISP_xxxx

With the exception of the **DISP_TYPE** function, which returns the standard SMSQ/E values, the other DISP_xxxx keywords, whilst they exist, have no effect on the Q68.

7 Configuring SMSQ/E for the Q68

Configuring SMSQ/E for the Q68 should be made via a standard configuration program. “MenuConfig” is certainly the way to go here.

If you opt to have SMSQ/E in a qxl.win container file called “Q68_SMSQ.WIN”, you can directly configure the file “Q68_SMSQ” in there. Thus, you do not have to leave the Q68 to configure its operating system. Please note, however, that once you have configured the Q68_SMSQ file to your liking, you MUST switch the Q68 off and on again. Simply resetting the machine (with the RESET command or otherwise) is not enough, as the OS is then not re-read from the card.

To make things easier, the Q68_SMSQ.WIN file delivered with the Q68 contains all files necessary for configuring SMSQ/E. As standard, this file is allocated to win8_. If you run win8_boot, it will LRESPR the necessary Menu extensions and launch MenuConfig. Please note that these two programs are © Jochen Merz.

The Q68 configuration block is divided into several sections, as follows:

7.1 SMSQ/E for the Q68

In this block, you can configure several standard SMSQ/E facilities (keyboard/message languages and so on). There should be no need to go into more details for them here, except perhaps to mention that the new CTRL behaviour enables you to keep CTRL+C pressed and the jobs will continuously cycle through.

7.2 Q68

This section contains some Q68 specific configuration items:

7.2.1 Initial display mode

Here you can set the display mode the Q68 should have when booting SMSQ/E. You may choose between the following:

Normal QL Mode 4
the standard 512 x 256 pixels in 4 colours mode. In this mode you can also set mode 8, with the usual MODE keyword.

Large QL Mode 4
1024 x 768 pixels in QL 4 colours mode (no mode 8 possible).

Aurora compatible 256 colours mode
1024 x 768 pixels in Aurora 8 bit colours mode.

Small 16 bit mode
512 x 256 pixels in mode 33 (16 bits per pixel).

Medium 16 bit mode
512 x 384 pixels in mode 33 (16 bits per pixel).

Large 16 bit mode
1024 x 512 pixels in mode 33 (16 bits per pixel). Please note that this mode will severely slow down the Q68, you should not use this mode when doing something time-critical.

Very large 16 bit mode
1024 x 768 pixels in mode 33 (16 bits per pixel). Please note that this mode will severely slow down the Q68, you should not use this mode when doing something time-critical.

7.2.2 Initialise card 2 when booting

This allows you to determine whether card2 should be initialised automatically during the boot process. Note that this will only work if there is actually a card in the socket to be initialised. You may choose between the following options:

- Never initialise the card when booting. This means that you will have to initialise it later yourself.
- Always initialise the card when booting. The driver will attempt to initialise card 2. Note that this will fail if no card is inserted at boot time. The driver does NOT tell you that the initialisation has failed in such a case.
- Only initialise the card during booting if a WIN drive is configured to be located there. This is the default option.

7.2.3 Boot from

Here you may set from what device SMSQ/E will attempt to load a boot file. You may choose between WIN drives 1 to 8, FAT drive 1, or no drive at all. Please note that if you choose to boot from FAT1, then the FAT device will be called FLP (i.e. you'll be booting from FLP1_). Using the command "**FAT_USE**" in your boot file will let the device be called FAT again.

7.2.4 Switch LED off when SMSQ/E is set up

The Q68 has a configurable LED which is switched on when SMSQ/E is loaded. Here you can determine whether it should stay on or not once SMSQ/E is fully set up.

7.3 Configuring the WIN drives

The following items may be configured for the WIN drives:

7.3.1 Filenames for win1_ to win8_

Each win drive corresponds to one container file on an SDHC card inserted into one of the two sockets. Here you may enter the file names for the container file for each of the 8 drives. Each name must respect the file name rules set out above – you won't be able to enter a name that does not comply with the 8.3 FAT32 standard, but you may give the name in lower case, it will be converted to upper case later by the system.

Please also make sure to check the warning given above.

7.3.2 Card assignment

Giving the file name of the container file for a win drive is not enough. The system must also know on what card this container file should be. Here you tell the system, for each drive, whether the corresponding container file should be on card1, card2, or on no card at all.

7.4 Configuring the FAT device

The following items may be configured for the FAT device:

7.4.1 Is the device to be loaded at all?

By default, the device is not linked into the system. If you do need it, set this configuration item to “yes”. In not, neither the device, nor its control thing nor the basic keywords associated with it are loaded.

7.4.2 On what card is the partition for each drive?

For a FAT drive to work, one (or both) SDHC card(s) must have at least one FAT16 formatted partition. You must thus tell the system, for each drive, on what card it must search for its partition.

7.4.3 What is the partition on that card for each drive?

The system also needs to know which partition on the card it is to look for. The partition needs to be a primary partition, not an extended partition. Since there can be only 4 primary partitions on an SDHC card to be used by the Q68, you can choose partitions 1-4. Note that in general, at least card1 will have a FAT32 partition as its first partition.

7.5 Configuring the QUB device and drives

The following items may be configured for the QUB device and drives:

7.5.1 Is the device to be loaded at all?

By default, the device is NOT linked into the system. If you do need it, set this configuration item to “yes”. In not, neither the device, nor its control thing, nor the basic keywords associated with it are loaded.

7.5.2 Filenames for qub1_ to qub8_

Each win drive corresponds to one container file on an SDHC card inserted into one of the two sockets. Here you may enter the file names for the container file for each of the 8 drives. Each name must respect the file name rules set out above – you won't be able to enter a name that does not comply with the 8.3 FAT32 standard, but you may give the name in lower case, it will be converted to upper case later by the system.

Please also make sure to check the warning given above.

7.5.3 Card assignment

Giving the file name of the container file for a QUB drive is not enough. The system must also know on what card this container file should be. Here you tell the system, for each drive, whether the corresponding container file should be on card1, card2, or on no card at all (in which case, the drive doesn’t exist).

8 Additional keywords and facilities

8.1 Sound

The Q68 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.

On the other hand, the Q68 uses a more modern sound system than the QL. Tis eables it to play stereo files with a surprisingly good sound. To make use of this, SMSQ/E for the Q68 uses the SSSS (SMSQ/E Sampled Sound System). This allows you to play “_ub” files.

To make this easier, some new keywords exist:

8.1.1 SOUNDFILE

Loads and plays a sound file through the SMSQ/E Sampled Sound System.

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

where:

- “***file_name***” is the file to be played. This 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 ram disk) to avoid the music being broken up. The quotes around the name are necessary unless it is a string variable.

- 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 be played twice in total (once + 1 repetition). Please use only positive values from 1 to 32766.

8.1.2 SOUNDFILE2

Syntax: **SOUNDFILE** *"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.

The parameters have the same meaning as for SOUNDFILE.

8.1.3 SOUNDFILE3

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

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

The parameters have the same meaning as for SOUNDFILE.

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.

8.1.4 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 playing all at once is not a good idea…

8.2 Access to fast memory

The Q68 has some “fast” memory, which can be accessed faster than the normal memory. There is only a very limited amount of fast memory available, some of which is already used by SMSQ/E.

Two keywords exist to get the amount of fast memory still available, and to reserve some space in it. Note: once reserved, the space cannot be given back (just like RESPR on a normal QL)!

8.2.1 FREE_FMEM

This function returns the amount, in bytes, of memory that is still **FREE** in the **F**ast **MEM**ory area. It does NOT free any memory in that area.

Syntax: *result* = **FREE_FMEM**

where *result* will be the amount of fast memory that is still free. In a freshly booted system this should be around 10 KiB.

8.2.2 ALFM

This function **AL**locates **F**ast **M**emory and returns the address of the allocated space in the fast memeory area.

Syntax: *address* = **ALFM** (*size*).

where:

- *size* is the amount of memory to allocate, in bytes. If you try to reserve more than is available, then the function returns with an out of memory error and no memory will have been reserved.
- *address* is the start address of the memory allocated if the call was successful. You may then use up to *size* bytes in the memory area starting at *address*. Note that the system does NOT stop you from using more than that – but sooner or later (and probably sooner rather than later) you WILL crash the system doing so.

8.3 Slug

The SLUG command can slow the machine down, which might be useful for some games.

Syntax: **SLUG** *how_much*

where how much is a value between 0 (no slug) and 255 (slowed down to a crawl).

8.4 Limited direct access to the card or the FAT32 file system

There are a few very limited (in number and scope) commands that allow some direct access to a card or the FAT32 file system of the first partition on a card. All of the related commands start with CARD_ as they relate to a card and not to any SMSQ/E device.
WARNING : please read this section carefully if you want to make use of these comands. Only use them if you know what you are doing!

8.4.1 CARD_INIT

Please see the section on initialising a card. This command is safe to use for everyone.

8.4.2 CARD_DIR$

This function shows the first 16 entries in the FAT32 root directory of the first partition on a card.

Syntax: result$ = **CARD_DIR$**(*card*) where *card* is the card to question (1 or 2).

On return, result$ will contain a large string with the 8.3 formatted names of the 16 entries, separated between them by a linefeed (CHR$(10)). These names may also be shown as "-- Empty --" which shows that the corresponding entry in the FAT32 root directory is empty, or "-Long name-" which shows that this entry does not point to a file but to a long name. The latter is NOT considered by SMSQ/E to be an empty entry in the directory.

8.4.3 CARD_RENF

This allows you to **REN**ame a **F**ile already existing in the first 16 entries in the FAT32 root directory of the first partition on a card.

Syntax: **CARD_RENF** *card,"old name","new name"*
where

- *card* is the card in question (1 or 2).
- "*old name*" is the name of the existing file to be renamed. It is preferred but not required that this be within quotes.
- "new *name*" is the new name of that file. It is preferred but not required that this be within quotes. If the new name already exists, the command fails with that error (see below, note 2).

Both names must comply with the "8.3" naming scheme.

8.4.4 CARD_CREATE

Allows creation of a file in the FAT32 file system of the first partition on a card. Remember, this partition must be a FAT32 partition. There must be a an empty slot for this file within the first 16 entries in the root directory of this partition (which you can check with **CARD_DIR$**).

Syntax : **CARD_CREATE** *card, size, file_name$*

where

- *card* is the card on which the file is to be created (1 or 2).
- *size* is the size, in MiB, of the file to be created. For the time being, **this cannot be more than 16 GiB in most cases** (see below, note 1).
- *file_name* is the name of the file to be created**. This must obey the "8.3 rule". It is recommended (but not necessary) that the name be within quotes.

The content of the file thus created is not "nulled". It may contain random bytes.

Error returns

Due to the complexity of this command, there are numerous possible error returns each having a different meaning:

name	**number**	**meaning**
err.drfl	(= -1)	drive full - there is no space for a file of this size within contiguous sectors on the card.
err.imem	(= -3)	insufficient memory for operation (I'd like to know how you managed that).
err.orng	(= -4)	the projected size of the is file too big, or 0 or negative.
err.bffl	(= -5)	buffer full - there is no space in the first 16 directory entries for a new file.
err.fex	(= -8)	already exists - a file with this name already exists in the first 16 directory entries for a new file (see below note 2).
err.inam	(= -12)	Invalid file name (not an 8.3 name).
err.ipar	(= -15)	wrong card number (not 1 or 2).
err.mchk	(= -16)	medium check failed because card wasn't readable (perhaps absent / not initialised) or this isn't a valid FAT32 partition

If everything goes alright, the command comes back without any error.

Example:

CARD_CREATE 1,200,"test.win". This will create a file called "TEST.WIN" on card 1, with a size of 200 MiB.

Finally, this command, combined with others, allows you to create a new qxl.win container on the card. To do this, you should proceed as follows:

```
10 CARD_CREATE 1,200,"test.win"     : REM create empty file, 200MiB in size
20 WIN_DRIVE 4,1,"test.win"         : REM point win4_ to it
30 WIN_FORMAT 4                     : REM allow formatting of win4_
40 FORMAT win4_your_name            : REM format win4_
```

(of course, you can use any other win drive for this, not only win4_).

Remember that the steps in lines 30 and 40 MUST be made from job 0, the main SBasic job, not from a daughter job!

Notes:

(1) The actual maximum size depends on the "cluster size" of the FaA32 file system on your card. As an indication:

Cluster size of	max file size
2	just under 8 GiB
4	just under 16 GiB
8	just under 32 GiB

(2) Please be careful when creating or renaming a file. SMSQ/E only checks for an existing file in the first 16 directory entries. If you have filled your card with enough files so that there are more than 16 entries (with some empty slot in the first 16 entries), then the risk exists that you may create a file with the name of an existing file.

9 Index of new or Q68 specific keywords

10 Avoiding fragmentation

SMSQ/E for the Q68 expects that all container files (i.e. files for WIN drives, and also for QUB drives) lie in contiguous sectors on the SDHC card. If this is not the case, the file is said to be “fragmented”. Fragmented files are deadly on the Q68 under SMSQ/E: SMSQ/E assumes that, once it has found the beginning of a container file, the rest of that container file lies in contiguous sectors on the card, and it will cheerfully write into those contiguous sectors which it deems still belong to that file. If the file is fragmented, these contiguous sectors may not belong to it but to another file, **which will thus be irretrievably corrupted**.

This is also true for the SMSQ/E binary file (named "QL_RAM.BIN") itself.

So, special precautions must be taken when writing the container and OS file(s) themselves to the card. The best and recommended way to achieve this is to make sure that, before writing the SMSQ/E binary file and the container files, the card is freshly formatted. Then, one after the other, write each container file to the card immediately after formatting the card.

Hence, you should dedicate a card solely for the purpose of using it with the Q68.

Note : practise has shown that in most cases it may not be necessary to reformat the card. You could also delete every single file on the card before copying new files onto it. Under no circumstances, however, should you only delete files selectively: this may leave “holes” in the file allocation table and this lead to fragmented files (see below). However, the recommendation still is to format the card and not just to delete all files from it.

When copying several files to the freshly formatted card, make sure that the copy process of each file is finished before you start that for the next file. If not, it may happen that the two copy processes write concurrently to the card, which could mean that the sectors for the two files interleave. Depending on the operating system you use (linux, windows, mac os) if you drag several files to the card at once, several concurrent copy processes might be started which might lead to file fragmentation. So, to avoid this, just drag the files to copy one after the other.

Moreover, never just delete a single file -be it a container file or any other file- from the card, but always format it (or at least delete ALL files from the card), and then write the files to the card again: If you delete a single file from the card and later write another, bigger, container file to the card, it is possible that part of this container file will lie in the sectors previously occupied by the deleted file, and the rest in previously unoccupied sectors. This file would then be fragmented and not lie in contiguous sectors on the card: a recipe for a disaster.

If a container file becomes fragmented, you WILL experience data loss, and other files on the card might also be irrecoverably damaged!

For some file systems, a special command exists to check whether a container file is fragmented or safe to use (see, e.g. the WIN_CHECK command).

W. Lenerz
November 2017