SMSQ Language Dependent Modules

Le Grand Pressigny, FRANCE - Tony Tebby

In order to be able to distribute versions of SMSQ which provide messages in more than one language and to support multiple keyboard layouts, SMSQ uses a uniform "language dependent module" structure.

At the moment, SMSQ supports four types of language dependent module:
1. language preference tables,
2. message tables,
3. keyboard tables,
4. printer translate tables.

The principle underlying the language dependent module structure is that each module is identified by a code: for the main languages supported this is the international dialling code (usually 1, 33, 44 or 49). New modules can be added, but these do not (at present) replace existing modules. For this reason, if you live in the south of France and wish to add messages in Occitan, it is strongly recommended that you identify the messages with a code other than 33 (for example 3300).

Language Module Headers - SMSQ creates a table of language modules. This is just a table of pointers to the individual modules. Clearly this does not impose any particular constraints on the size of a module, but it does require that a module has a header in a known form (defining the type of module and its language code). In addition, each module header includes a link pointer to another module header so that many modules may be added to the system in one call. Finally, to provide the maximum flexibility, the module header is not attached directly to the module, but the last long word of the module header is a relative pointer to the module itself.

The code to link in a list of language modules is quite simple.

```
        moveq   #sms.lldm,d0        key to link in
        lea     lang_mod,a1         pointer to modules to link
        trap    #1                  do SMS call
        rts

lang_mod                            modules start here
```

This code can be written to a file using WPUT (LPUT can be used for greater efficiency, if you wish).

```
WPUT #fch, $7030, $43FA, $0006, $4E41, $4E75
```

Alternatively, the code can be written directly to the computer's memory (useful for testing).

```
POKE_W base, $7030, $43FA, $0006, $4E41, $4E75
```

Language Preference Tables - The language preference tables are the most important and the simplest. A language preference table is simply a language name (usually the international car registration letters) followed by a table of acceptable language numbers in order of decreasing acceptability.

This allows the creation of a new language variant without the need to define all of the tables. Thus for Occitan, the second preferred language would probably be French, and, since there is a complete set of French tables, further preferences would not be needed.

```
occitan_pref
        dc.w    0               it is a preference table
        dc.w    0               always zero
        dc.w    3300            Occitan language number
        dc.w    next-*          relative pointer to next or zero
        dc.l    occ_pref-*      pointer to preference table
```

```
occ_pref
    dc.l    'FOC '          Occitan is a language of France
    dc.w    3300            Occitan is the most acceptable
    dc.w    33              French is next most acceptable
    dc.w    0               . . . and that is all
```

If the preference table is to be written to a file using SBASIC, WPUT and BPUT are the most appropriate routines to use.

```
WPUT #fch, 0, 0, 3300, 0, 0, 4      the header - no next, table follows
BPUT #fch, 'FOC '                   the name: left justified, space filled to 4 chars
WPUT #fch, 3300, 33, 0              the preferred languages
```

If this seems a bit heavy, it is. The system is designed to cope with many more languages than you are ever likely to need and to allow dialects or personal variations to be added without inhibiting access to the standard languages. Once you have linked in this new preference table, you can check whether it is there by printing LANGUAGE$:

```
PRINT LANGUAGE$ (3300)          should print        FOC
```

Message Tables - SMSQ/E uses four message tables itself. The messages are in four groups. It is possible to add new message tables for these four groups for new languages. It is also possible to add new groups of messages for the existing languages and for new languages. Software developers are requested not to treat this too lightly and create new groups frivolously.

At present, the groups are numbered in 4s (message group 0 is the set of old QL standard messages, message group 4 is the set of SBASIC syntax and execution error messages etc.). The error or message code used to access a message is

-(error message number + 32 x error message group number)

The messages in a group may be listed using REPORT

```
FOR mess = 1 to 50: REPORT #n, -(mess + 32 * group)
```

A message table has the standard language dependent module header. The table itself has pointers to the messages which are relative to the start of the table. As there can be no message 0, the zeroth pointer is replaced by the language number. We can add a new message table for group 12 (table of months and days of week) which (fortunately for this example) only has two entries.

```
occitan_ms12
    dc.w    3               it is a message table
    dc.w    12              group 12
    dc.w    3300            Occitan language number
    dc.w    next-*          relative pointer to next or zero
    dc.l    occ_ms12-*      pointer to preference table

occ_ms12
    dc.w    3300            Occitan
    dc.w    occ_mnth-occ_ms12   pointer to first message
    dc.w    occ_dow-occ_ms12    pointer to second message

occ_mnth
    dc.w    36,'IchNi SanGo Ro ShiHa Ku Ju JuiJun'

occ_dow
    dc.w    21,'Ni Ge Ka SuiMo KinDo '
```

Note that the first message must follow the last pointer: the first pointer is, therefore, twice the number of messages in the table, plus 2. It is possible to write this message table to a file using a simple BASIC program as follows, but for the more complex tables with variable length messages a more complex program would be required.

```
WPUT #fch, 3, 12, 3300, 0, 0,      4        the header - no next, table follows
WPUT #fch, 3300, 6, 6+2+36                  the number and two pointers
PUT #fch, 'IchNi SanGo Ro ShiHa Ku Ju JuiJun'
PUT #fch, 'Ni Ge Ka SuiMo KinDo '           the two strings
BPUT #fch, 0                                a pad byte because the string was odd length
```

Once you have linked in this new message table, you can use it by typing the LANG_USE command:

```
LANG_USE 3300        or
LANG_USE FOC
```

Keyboard Tables - The language dependent module that is next most likely to be added is a keyboard table. A keyboard table has the standard language dependent module header. For historical reasons, the module pointer in the header points to an intermediate structure. This intermediate structure has the language code followed by a pair of relative pointers the first of which points to the "normal" keyboard table, the second points to a table of "non-spacing characters". These are keys which when pressed do nothing but modify the next character typed. These keys are usually an accent key which is used to add an accent to the next letter. (Some keyboard drivers may not support non-spacing characters.)

```
occitan_kbd
    dc.w    1               it is a keyboard table
    dc.w    0               no group
    dc.w    3300            Occitan language number
    dc.w    next-*          relative pointer to next or zero
    dc.l    occ_kbd-*       pointer to preference table

occ_kbd
    dc.w    3300            Occitan
    dc.w    occ_ktab-*      pointer to keyboard table
    dc.w    occ_nsid-*      pointer to non-spacing character table
```

If the keyboard table follows immediately after the header and the non-spacing table immediately after that, the header may be written by a simple SBASIC program

```
WPUT #fch, 1, 0, 3300, 0, 0, 4          the header - no next, table follows
WPUT #fch, 3300, 4, 2+512               the number and two pointers (all but QL kbd)
or  WPUT #fch, 1, 0, 3300, 0, 0, 4      the header - no next, table follows
WPUT #fch, 3300, 4, 2+256               the number and two pointers (QL kbd)
```

The size of the keyboard table depends on the keyboard itself. The table is divided into four blocks: normal keystrokes, control keystrokes, shifted keystrokes and shifted control keystrokes. The tables are the characters produced for each of the possible keyboard codes. For the main keyboards, these keyboard codes are as follows.

QL keyboard - 64 entries in each of 4 blocks

```
F1-F5      57    59     60   56   61
Top row    51    27     9    25   62    58    10     63   8     16   13   21   37   45   53
             19      11    17   12   20    14    22     15   18    23   29   32   40
               33     28    35   30   36    38    26     31   34    24   39   47   48
                 (0)       41    3    43    4     44     6    46    7    42   5    (0)
                 (1)  49       52                      54                     50   55   (2)
```

Thus, for a normal QL keyboard layout, the codes for the digit keys are 27, 9, 25 etc. The 27th entry in the keyboard table should be the character '1', the 9th entry '2' etc. (The table starts with the zero'th entry.) *The key codes in brackets are trapped by the driver (Shift, Control and Alt) and the corresponding values in the keyboard tables should be zero.*

Atari ST TT keyboard - 128 entries in each of 4 blocks

F1-F10	59	60		61	62	63		64	65		66	67		68				
Top row	1	2		3	4	5		6	7		8	9		10	11	12	13	41	14
	15		16		17	18	19	20		21		22	23		24	25	26	27		83
	(29)		30		31	32	33	34		35		36	37		38	39	40		28	43
	(42)	96		44		45	46	47		48	49		50	51		52	53	(54)		
			(56)								57						58			

Cursor pad		98			97			Numeric pad		99	100			101			102	
		82		72	71					103	104			105			74	
		75		80	77					106	107			108			78	
										109	110			111				
											112			113			114	

Thus, for a normal ST keyboard layout, the codes for the digit keys are 2, 3, 4 etc. The 2nd entry in the keyboard table should be the character '1', the 3rd entry '2' etc. (The table starts with the zero'th entry.) *The key codes in brackets are trapped by the driver (Shift, Control and Alt) and the corresponding values in the keyboard tables should be zero.*

QXL AT keyboard - 128 entries in each of 4 blocks - The PC models AT and later incorporate an "intelligent" keyboard controller which has three main functions:

1. converting the easy to handle, explicit, AT 102 key keyboard codes into garbled sequences of PC XT keyboard codes (up to 10 keycodes for each keystroke!);

2. losing keystrokes;

3. getting shift keys "stuck down".

The keyboard tables are, therefore, based on the PC XT key codes.

Esc-F10	1		59	60	61	62		63	64		65	66			67		68			
Top row	41	2		3	4		5	6	7		8		9	10		11	12	13		43	14
		15	16		17	18		19	20	21		22		23		24		25	26	27	
		58	30		31	32		33	34		35		36		37		38	39	40		28
	(42)	86		44		45		46	47	48		49		50		51	52		53	(54)	
	(29)				(56)						57							(56)			(29)

Odd pad			55		70	(69)														
Cursor pad			114		103	105			Numeric pad		(69)			133		55		74		
			115		111	113					71			72		73				
											75			76		77		78		
					104						79			80		81				
			107		112	109						82				83		124		
									Numeric pad		(69)			133		55		106		
									Without Num Lock		103			104		105				
											107			108		109		110		
											111			112		113				
												114				115		124		

Thus, for a normal PC keyboard layout, the codes for the digit keys are 2, 3, 4 etc. The 2nd entry in the keyboard table should be the character '1', the 3rd entry '2' etc. (The table starts with the zero'th entry.) *The key codes in brackets are trapped by the driver (Shift, Control and Alt) and the corresponding values in the keyboard tables should be zero.*

Sample Keyboard Table - It is easiest to see the format of the keyboard table if the PC AT 102 key keyboard layout is taken as an example. The first block of 128 characters is for unshifted characters.

```
occ_ktab                ; unshifted keys for UK kbd (in groups of 16)
    dc.b    0,$1b,'1','2','3','4','5','6','7','8','9','0','-','=',$c2,$09
    dc.b    'q','w','e','r','t','y','u','i','o','p',$5b,$5d,$0a,0,'a','s'
    dc.b    'd','f','g','h','j','k','l',';',$27,$9f,0,'#','z','x','c','v'
    dc.b    'b','n','m',',','.','/',0,'*',0,' ',$e0,$e8,$ec,$f0,$f4,$f8
    dc.b    $ea,$ee,$f2,$f6,$fa,0,$f9,'7','8','9','-','4','5','6','+','1'
    dc.b    '2','3','0','.',0,0,'\',0,0,0,0,0,0,0,0,0
    dc.b    0,0,0,0,0,0,0,$d5,$d0,$d4,0,$c0,0,$c8,0,$dd
    dc.b    $d8,$dc,$eb,$ca,0,'/',0,0,0,0,0,0,$0a,0,0,0
```

This block is followed immediately by the block of 128 shifted characters, then the 128 characters which are produced when the control key is held down and finally the 128 characters which are produced when both shift and control keys are held down. As there are only 256 different values that can be stored in a byte and there are 512 total keyboard table entries, there will naturally be a large number of zeros in the tables as well as a certain number of duplicate codes.

Non-Spacing Characters - The non-spacing character table is a little bit odd. It is a 256 byte table which is (nearly) filled with zeros. For any character which can be used as a non-spacing character, the corresponding entry in the table is non-zero. Thus, if the ' is used as a non-spacing character to produce accented characters, the 39th entry in the table is non-zero (' is ASCII code 39). The non-zero value is the offset from the end of the table to the list of modified characters for this non-spacing character.

The table is immediately followed by a variable size table of modifiable and modified characters. This table has entries which are one longer than the number of modifiable characters (each entry is terminated by a zero). The first entry lists the modifiable characters. This is followed by an entries giving the corresponding modified characters for each of the non-spacing characters.

```
occ_nsid
    dc.b    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    dc.b    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    dc.b    0,0,0,0,0,0,0,8,0,0,0,0,0,0,0,0     apostrophe for acute accent
    dc.b    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    dc.b    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    dc.b    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    dc.b    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    dc.b    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    dc.b    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,16    open quote for grave accent
    dc.b    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    dc.b    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    dc.b    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    dc.b    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    dc.b    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    dc.b    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    dc.b    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

    dc.b    'a','e','i','o','u','E',' ',0       modifiable characters (including space)
    dc.b    $8C,$83,$93,$96,$99,$A3,$27,0       acute accented characters (including ')
    dc.b    $8D,$90,$94,$97,$9A,'E',$9F,0       grave accented characters (including £)
```

This table can be written to a file using BPUT commands in an SBASIC program.

```
nsid = FPOS (#fch)                    : REMark - remember where it starts
BPUT #fch, FILL$(CHR$(0),256)         : REMark - fill table with zeros
BPUT #fch\nsid+39, 8                  : REMark - fill in apostrophe
BPUT #fch\nsid+159, 16                : REMark - fill in open quote
BPUT #fch\nsid+256                    : REMark - back to end of table
BPUT #fch, "aeiouE ",0                : REMark - unmodified characters at end
BPUT #fch, "áéíóúÉ'",0                : REMark - then acute accents
BPUT #fch, "àèìòùE`",0                : REMark - and grave accents
```

Once you have linked in this new keyboard table, you can use it by typing the KBD_TABLE command:

```
KBD_TABLE 3300        or
KBD_TABLE FOC
```

Printer Translate Tables - The printer translate tables are directly compatible with the old, not very useful, QL printer translate tables used by the TRA command.

These "language dependent" tables came into existence because someone at Sinclair had the rather strange notion that, in some way, the character ä (for example) should be printed differently depending on the country. I think that an ä is an ä wherever you are.

SMSQ has a standard printer translate that works on any PC compatible printer set to the USA character set (the use of a non-USA character sets tends to make it impossible to print certain characters).

For peculiar printers, however, you can set up your own tables.

A printer translate table has the standard language dependent module header. For historical reasons, the module pointer in the header points to an intermediate structure. This intermediate structure has the language code followed by a pair of pointers relative to the language code. The first of these points to the "byte to byte" translate table, the second points to a table of "byte to three byte" translates".

The byte to byte table has, naturally, 256 single byte entries. The first entry is zero (null stays as null) all other entries are either the translated character or zero. If the entry is zero, the character is translated using the three byte table.

The three byte table is preceded by a zero byte (historic) and starts with the number of three byte sequences (in a byte). This is followed by groups of 4 bytes, the first of which is the QL character, the next three are the characters to be sent to the printer.

The following is a copy of the IBM printer translate table which may be used as a basis for other printers.

```
occitan_tra
        dc.w    2               it is a printer translate table
        dc.w    0
        dc.w    3300            Occitan language number
        dc.w    next-*          relative pointer to next or zero
        dc.l    occ_tra-*       pointer to preference table

occ_tra
        dc.w    3300            Occitan
        dc.w    occ_byte-occ_tra    pointer to byte to byte table
        dc.w    occ_3byte-occ_tra   pointer to three byte table
```

```
occ_byte
        dc.b    $00,$01,$02,$03,$04,$05,$06,$07
        dc.b    $08,$09,$0A,$0B,$0C,$0D,$0E,$0F
        dc.b    $10,$11,$12,$13,$14,$15,$16,$17
        dc.b    $18,$19,$1A,$1B,$1C,$1D,$1E,$1F
        dc.b    $20,$21,$22,$23,$24,$25,$26,$27
        dc.b    $28,$29,$2A,$2B,$2C,$2D,$2E,$2F
        dc.b    $30,$31,$32,$33,$34,$35,$36,$37
        dc.b    $38,$39,$3A,$3B,$3C,$3D,$3E,$3F

        dc.b    $40,$41,$42,$43,$44,$45,$46,$47
        dc.b    $48,$49,$4A,$4B,$4C,$4D,$4E,$4F
        dc.b    $50,$51,$52,$53,$54,$55,$56,$57
        dc.b    $58,$59,$5A,$5B,$5C,$5D,$5E,$5F
        dc.b    $9C,$61,$62,$63,$64,$65,$66,$67
        dc.b    $68,$69,$6A,$6B,$6C,$6D,$6E,$6F
        dc.b    $70,$71,$72,$73,$74,$75,$76,$77
        dc.b    $78,$79,$7A,$7B,$7C,$7D,$7E,$00
        dc.b    $84,$00,$86,$82,$94,$00,$00,$81
        dc.b    $87,$A4,$91,$00,$A0,$85,$83,$89
        dc.b    $8A,$88,$8B,$A1,$8D,$8C,$A2,$95
        dc.b    $93,$A3,$97,$96,$E1,$9B,$9D,$60
        dc.b    $8E,$00,$8F,$90,$99,$00,$00,$9A
        dc.b    $80,$A5,$92,$00,$E0,$EB,$E9,$00
        dc.b    $E6,$E3,$ED,$AD,$A8,$3F,$EC,$00
        dc.b    $AE,$AF,$F8,$F6,$00,$00,$00,$00

        dc.b    $C0,$C1,$C2,$C3,$C4,$C5,$C6,$C7
        dc.b    $C8,$C9,$CA,$CB,$CC,$CD,$CE,$CF
        dc.b    $D0,$D1,$D2,$D3,$D4,$D5,$D6,$D7
        dc.b    $D8,$D9,$DA,$DB,$DC,$DD,$DE,$DF
        dc.b    $B0,$B1,$B2,$B3,$B4,$B5,$B6,$B7
        dc.b    $B8,$B9,$BA,$BB,$BC,$BD,$BE,$BF
        dc.b    $F0,$F1,$F2,$F3,$F4,$F5,$F6,$F7
        dc.b    $F8,$F9,$FA,$FB,$FC,$FD,$FE,$FF

        dc.b    0               ; pad

occ_3byte
        dc.b    15              ; 15 replaces
        dc.l    $A54F087E       ; O bs tilde
        dc.l    $AF5C082E       ; \ bs .
        dc.l    $7F63084F       ; c bs O
        dc.l    $8161087E       ; a bs tilde
        dc.l    $856F087E       ; o bs tilde
        dc.l    $866F082F       ; o bs /
        dc.l    $8B6F6500       ; o e
        dc.l    $A141087E       ; A bs tilde
        dc.l    $A64F082F       ; O bs /
        dc.l    $AB4F4500       ; O E
        dc.l    $B76F0878       ; o bs x
        dc.l    $BC3C082D       ; < bs -
        dc.l    $BD3E082D       ; > bs -
        dc.l    $BE5E0821       ; ^ bs !
        dc.l    $BF760821       ; v bs !
```

SMSQ Language Dependent Modules - (cont'd)

Once you have linked in this new printer translate table, you can use it by typing the TRA command:

TRA 1, 3300 or
TRA 1, FOC

A Complete Language Dependent Extension - This SBASIC program creates a complete language dependent extension with preference, keyboard and message tables. SBASIC procedures are used to set the relative pointers: you can try to decipher them if you wish.

The keyboard table is the standard UK PC (QXL) keyboard with one difference: the F6 to F12 keys are used as non-spacing characters. To avoid conflicts with existing key codes, the ALT cursor key codes are borrowed (the ALT key is handled within the driver and so the only ALT key codes which appear in the table are those for HOME (=ALT SHIFT UP) and END (=ALT SHIFT DOWN)).

The message tables are very slightly modified versions of the English tables: All four standard groups are included to make it easier for you to create your own message tables.