CONFIG

Standard Configuration Information Specification

(Please note: most of this text stems from Tony Tabby and Jochen Merz).

Many programs have the facility to configure themselves to set default working parameters. More usually the configuration is done by a separate program which modifies the working program file. Each program will have a different configuration program, and often different versions of the same program will have different configuration programs too. All this makes things very difficult for users.

It is proposed that a standard configuration system is used on all new programs and all new releases of existing programs. If this is done, a single configuration program can be used on any application software file even even when several application files are concatenated. A program that adheres to this standard must include one or several configuration blocks.

The advantages of this approach are obvious. There are two disadvantages. The first is that each program has to carry with it all the configuration information: this will make it larger. The second is that there is no simple means for doing this with compiled BASIC programs. The first will not usually be a problem as it seems unlikely that a 32k program would have more than about 20 configurable items and their associated descriptions, this would add at most 3% to the pro-gram size. The second can be overcome with a little will.

There are two parts to this system: the first is a standard for the format of a configurable file, the second is a program to process files. There can be any number of programs to process files, from any number of suppliers. If the standards for the configurable file are adhered to, then any supplier's configuration program can be used on any (other) supplier's software. Most configuration programs will assume that the configuration block in an application program is correct.

For the time being, there is only one program which handles all currently existing configuration levels: *MenuConfig* by Jochen Merz Software.

Indeed, there are two configuration levels, level 01 and level 02. Level 02 is an extension of level 01. Today, there seems to be no need to create just a level 01 configuration block for any new software.

Table of Contents

1 Configuration Level 01

1.1 Configuration block

The configuration block structure for level 01 contains the following information:

Configuration ID
Configuration level
Software name
Software version
List of
    Type of item (string, integer etc..) (byte)
    Item Selection keystroke (byte)
    Pointer to item
    Pointer to item pre-processing routine
    Pointer to item post-processing routine
    Pointer to description of item
    Pointer to attributes of item (item type dependent)
End word (value -1)

It was initially envisaged that, as time goes on, additional types of item could be added. This seems unlikely now (with the exception of Level 02) and it would mean that new versions of the configuration program(s) would be required. These new versions would, of course, be able to configure all lower level configurable files. But, if an old configuration program were used, and the level specified in the configuration block were greater than the level supported by the configuration program, it would have to give up gracefully.

1.2 Configuration ID and configuration level

The **configuration ID** is word aligned and consists of the eight characters "<<QCFX>>", this is followed by two **ASCII** characters giving the **configuration level** (minimum "01"). For the time being, only MenuConfig can handle level 02.

1.3 Software name and version

The **software name** is a standard string and is followed by a word aligned **version** identification in a standard string (e.g. "1.13a", preceded by a length word, of course). The word aligned list of items follows.

1.4 Items

Items are ordered in a list terminated by the word -1.

1.4.1 Types of item

The item type is one byte. Levels 01 and 02 support 7 types of item. These are: string, character, code selection, code, byte, word and long word. Application specific types of item can be processed by treating them as strings which are handled entirely by an application supplied routine (see the Item Pre-Processing and Item Post-Processing Routines).

1.4.1.1 String (type=0)

The form of a configurable string is a word giving the maximum string length, followed by a standard string. There should be enough room within the application program for the maximum length string plus one character for a terminator. There is a single word of attributes with bits set to determine special characteristics.

bit 0	set to strip spaces
bit 8	set if string is filename
bit 9	set if string is directory
bits 8 and 9	if both are set, string is an extension

```
cfs.sspc equ  %0000000000000001  string strip spaces
cfs.file equ  %0000000100000000  string is filename
cfs.dir  equ  %0000001000000000  string is directory
cfs.ext  equ  %0000001100000000  string is extension
```

At present, the features corresponding to bytes 8 and 9 are supported by MenuConfig, and ignored by the old “config” program.

1.4.1.2 Character (type=2)

A character is a single byte, if it is a control character, it will be written out as a two character string (e.g. ^A = $01). There is a single word of attributes with bits set to determine the possible characters allowed.

bit 0	non printable characters
bit 1	digits
bit 2	lower case letters
bit 3	upper case letters
bit 4	other printable characters
bit 5	-
bit 6	cursor characters
bit 7	-
bit 8	control chars + $40, translated to control chars

Bit 8 is, of course, mutually exclusive with bits 0 to 7, although this is not checked. The configuration block in an application program must be correct.

1.4.1.3 Code (type=4)

A code is a single byte which may take a small number of values. The attributes is a list of codes giving a byte with the value, a byte with the selection keystroke and a standard string. The list is terminated with an end word (value -1). There are two forms. In the first, the selection keystrokes are set to zero. In this case, when a code is selected, the value will step through all possible values. This is best suited to items which can only have two or three possible codes. Otherwise the user may select any one of the possible codes, either from a list (interactive configuration programs) or from a pull down menu (menu driven configuration programs).

1.4.1.4 Selection (type=6)

A selection is in the same form as a code, but instead of a byte being set to the selected value, the value is treated as an index to a list of status bytes. When one is selected, it is set to wsi.slct ($80), the previous selection (if different) is set to wsi..avbl (zero). If any status bytes are unavailable (set to wsi.unav=$10), then they will be ignored. The first status byte in the list must not be unavailable.

1.4.1.5 Values (types 8, 10, 12)

Largely self explanatory. The values are byte, word or long word. Their attributes are the minimum and maximum values. All values are treated as unsigned.

1.4.2 Item Selection Keystroke

The item selection keystroke is an uppercased keystroke which will select the item in the menu (of a menu driven configuration program). It is set in a byte immediately following the item type byte. The action of selecting the item will depend on the item type. For a code or select item a pull-down window may be opened to enable the user to select the appropriate code. For character item, a single keystroke will be expected. For all other types of item, the item will be made available for editing. For interactive configuration programs, the selection keystroke has no meaning.

1.4.3 Pointer to Item

This points to the actual configuration item itself, i.e. the part that is changed when an item is configured. The pointer to item, and all other pointers in the configuration block, are relative addresses stored in a word (e.g. `dc.w item-*`).

1.4.4 Pointer to Item Pre-Processing Routine

It is possible to provide a pre-processing routine within the main program which will be called before an item is presented for changing.

This will be when the item is selected in a menu configuration program, or before the prompt is written in an interactive configuration program. If there is no pre-processing routine, the pointer should be zero.

The amount of pre-processing that application program can do is not limited. It could just set ranges, or it could do the complete configuration operation itself, including pulling down windows.

Pre-processing Routine

Call parameters		**Return parameters**	
		D0	item set / error
		D1+	scratch
D7	0 / Window Manager vector	D7	scratch
A0	pointer to item	A0	scratch
A1	pointer to description	A1	(new) ptr to description
A2	pointer to attributes	A2	(new) ptr to attributes
A3	pointer to 4 kbyte space	A3	scratch
		A4+	scratch

Error returns: set as D0

- >0 item set, do not prompt or change
- =0 OK
- <0 error

The space pointed to by A3 is not used by the configuration program and can be used by the application code. Initially it is clear. The application code may use up to 512 bytes of stack.

If D0 (and the status) is returned <0, then the Configuration program will write out an error message and stop.

If D7 is not 0 on entry (which could happen with an interactive configuration program not running under the Pointer Environment), then it should contain the PE's Window Manager vector.

1.4.5 Pointer to Item Post-Processing Routine

It is possible to provide a “post-processing” routine within the main program, to which this pointer then points. Note that post-processing” is pretty much a misnomer as this routine (if it exists) will be called (at least by MenuConfig) for **every** item **before** configuration starts, and also for **every** item (not only for the just configured one) after any item is changed. It can be used to set limits or other dependencies.

Post-processing Routine

Call parameters		**Return parameters**	
		D0	item set / error
D1.b	set this item just changed	D1.b	item status (avbl/unav)
		D2+	scratch
D7	0 / Window Manager vector	D7	scratch
A0	pointer to item	A0	scratch
A1	pointer to description	A1	(new) ptr to description
A2	pointer to attributes	A2	(new) ptr to attributes
A3	pointer to 4 kbyte space	A3	scratch
		A4+	scratch

Error returns: set as D0

- >0 bit 0 item reset
  - bit 1 description reset
  - bit 2 attributes reset
- =0 OK
- <0 error

The space pointed to by A3 is not used by the configuration program and can be used by the application code. Initially it is clear. The application code may use up to 512 bytes of stack. If an item description is changed, it should occupy the same number of lines as the original.

The returned values for D1 are WSI.AVBL ($00) if the item can be changed or WSI.UNAV ($10) if the item is not available for changing.

If D0 and the status are <0, A1 and A2 and the item status will not be updated, the error message will be written out, no further post-processing routines will be called, and the item just set will be re-presented.

If D7 is not 0 on entry (which could happen with an interactive configuration program not running under the Pointer Environment), then it should contain the PE’s Window Manager vector.

A post-processing routine can also be used to set up initial descriptions and attributes.

If there is no post-processing routine, the pointer should be zero.

1.4.6 Description of Item

The description of an item is in the form of a string. Each description can have several lines, separated by newline characters. Each line should be no longer than 64 characters, except the last line must allow space for the longest item. Interactive programs may append a list of states or selections to the description.

1.4.7 Pointer to attributes

The attributes are item dependent. See item types for descriptions of the attributes.

Note that the attributes are not modified when configuring the program. This means that you can re-use the same attributes for different items provided they require the same attributes. For example, if the configuration block contains several items that are simple yes or no questions, you can use the same attributes for all of these items.

1.4.8 End of list marker

Once every item is set out as per above, the item list is terminated by a word -1.

2 Configuration Level 02

2.1 Extended configuration block

Re-configuring software you already had in previous versions is a very boring thing. Most of the time, all you do is set the old settings in the new file. This has to be made automatic. Therefore, the item structure was expanded in level 02 to make room for an Item ID. Consequently, the configuration block structure for level 02 consists of the following information:

(Optional cutoff flag) ← NEW!!!
Configuration ID
Configuration level
Software name
Software version
List of
    Item ID (long) ← NEW!!!
    Type of item (string, integer etc..) (byte) ← See below
    Item Selection keystroke (byte)
    Pointer to item
    Pointer to item pre-processing routine
    Pointer to item post-processing routine
    Pointer to description of item
    Pointer to attributes of item (item type dependent)
End word (value -1)

As you can see, there is an optional cutoff flag and an additional (not optional) Item ID for each item, compared to level 01. Unless otherwise stated below, the rest remain identical to level 01.

2.2 The "<<QCFC>>" cutoff flag

If a configuration block contains the special flag "**<<QCFC>>**" BEFORE the "**<<QCFX>>**" configuration block ID flag, then MenuConfig offers the user the choice to save a configured version without the configuration description texts, to reduce the required file size to the minimum (as the configuration texts are not required any more after configuration). Of course, a file treated this way can no longer be configured afterwards.

Basically what happens is that when the configured file is saved back to the disk, the file is cut off right before the configuration description texts.

This means that programmers should take care that the configuration items come BEFORE the configuration texts, otherwise they will be cut away too. So make sure that the configuration texts are always the last section in your file!!!

2.3 Item ID

The Item ID is one long word. The ID should be unique for every item. There may be global ID names, which could be used by many programs (like the colourway setting), there can be unique "registered" ID names (which are preferred) and there may be "unregistered" local ID names. Global ID names should start with an underscore, unique ID names should start with a letter. For unregistered local IDs, the top byte of the ID has to be 0.

The global IDs are:

_COL Main Colourway	Byte range -1, 0 to 3 (extended to 0-7 for the 4 palettes).
_COS Sub-Window Colourway	(same)
_COB Button Colourway	(same)

To avoid multiple name conflicts, I attempt to maintain a list of all IDs. If you wish to register for one or more ID names, please email me at my usual address (“wolf” - the usual at sign- “wlenerz.com”). A list of currently known IDs is maintained at www.wlenerz.com/smsqe, go to the section on additional information and data.

2.4 An additional item type?

At some stage, it was considered that a new item type should be added. To quote the original documentation:

“ *It became obvious in MenuConfig, that a new item type "nothing" or "all" is required, which does not do anything automatic but calling the pre/post- processing routines. This is useful for proving own menus without having to mess around with unwanted texts. In addition, more information is required to be passed to these pre/post-processing routines. We think, at the moment, of the following scheme:*

A3, which points to a 4kBytes space, is negative indexed and provides the following information:

$0000 4k base of workspace passed to pre/post-processing routine
-$0004 long MenuConfig's version
-$0008 long primary channel ID
-$000c long pointer to working definition
-$0010 2 word primary window x/y size
-$0014 2 word primary window x/y origin
-$0018 2 word work area x/y size
-$001c 2 word work area x/y origin
-$001d byte text info window number in working def
-$001e byte work info window number in working def
-$0022 long window manager vector
-$0026 long pointer to filename of the file being configured
-$002a long pointer to buffer containing file being configured
-$002e long pointer to buffer of default directory
-$0032 long pointer to buffer of output device
-$0040 long colourway “

It is unclear to me whether this item type was ever seen in the wild, I presume that this stayed at the project stage.

3 Examples

Here are two examples of configuration blocks, the first one done entirely manually, the second using the macros in the SMSQ/E sources.

3.1 An example of a “normally” coded assembler configuration block, level 02.

```
config block
        dc.w    '<<QCFC>>'          ; cutoff flag
        dc.w    '<<QCFX>>'          ; normal header
        dc.w    '02'                ; level
        dc.w    qc1-*-2             ; length of name
        dc.b    'Colours'           ; name
qc1
        dc.w    4                   ; length word, 4 bytes for vers. string
        dc.l    '1.00'              ; version string

; item: get default colours from menu extensions?
        dc.l    'ExFT'              ; item ID
        dc.b    4                   ; type of item (selection, here: yes/no)
        dc.b    'M'                 ; selection keystroke
        dc.w    defmcol-*           ; pointer to item
        dc.w    0                   ; no pre-processing
        dc.w    0                   ; no post-processing
        dc.w    coldesc-*           ; description
        dc.w    ynattr-*            ; attributes

; item; choose main colour
        dc.l    '_COL'              ; item ID (global ID!)
        dc.b    4                   ; type of item (selection, here: colourway)
        dc.b    'C'                 ; selection keystroke
        dc.w    colw-*              ; pointer to item
        dc.w    pre1-*              ; pre-processing
        dc.w    post1-*             ; post-processing
        dc.w    mwcol_s-*           ; description
        dc.w    colattr-*           ; attributes

; item; choose dir name
        dc.l    'ExFE'              ; item ID
        dc.b    0                   ; type of item (string, here: directory)
        dc.b    'D'                 ; selection keystroke
        dc.w    dirnm-*             ; pointer to item
        dc.l    0                   ; no pre- nor post-processing
        dc.w    gendes-*            ; description
        dc.w    diratt-*            ; attributes

; what; is char for hex sign?
        dc.l    'ExFK'              ; item ID
        dc.b    2                   ; type of item (character)
        dc.b    'H'                 ; selection keystroke
        dc.w    hexsign-*           ; pointer to item
        dc.l    0                   ; NO pre- and post-processing
        dc.w    hexdes-*            ; description
        dc.w    charatt-*           ; attributes

        dc.w    -1                  ; end of list and config block
```

```
; THE ITEMS THEMSELVES
defmcol dc.b      0                ; (yes/no) here: 0 = no

colw    dc.b      4                ; main wdw colour : palette 1

dirnm   dc.w      30               ; dir name, max length = 30 bytes
        dc.w      11               ; current dir name length
        dc.b      'win1_my_dir'    ; name
        dcb       19,0             ; the 19 remaining bytes

hexsign
        dc.b      '$',0            ; NB make sure next is even

; the attributes
; yes no attribute; proposes yes or no as options 0 = NO, $80 = YES
ynattr  ds.w      0,0              ; make sure this label is at en even address
        dc.b      0                ; value
        dc.b      0                ; selection keystroke (none)
        dc.w      2,'No'           ; standard string
        dc.b      $80              ; value
        dc.b      0                ; selection keystroke (none)
        dc.w      3,'Yes'
        dc.w      -1               ; end of attributes

; the attribute to choose colourways, again a selection
colattr dc.b      0,0              ; value and selection keystroke (none)
        dc.w      11,'White/Green' ; standard string
        dc.b      1,0              ; value and selection keystroke (none)
        dc.w      9,'Black/Red'    ; standard string
        dc.b      2,0                    (... etc ...)
        dc.w      9,'White/Red'
        dc.b      3,0
        dc.w      11,'Black/Green'
        dc.b      4,0
        dc.w      16,'System palette 1'
        dc.b      5,0
        dc.w      16,'System palette 2'
        dc.b      6,0
        dc.w      16,'System palette 3'
        dc.b      7,0
        dc.w      16,'System palette 4'
        dc.w      -1

; attributes for the dirname (string)
diratt  dc.w      $200             ; = %0000001000000000 bit 9  is set

; attributes for the character
charatt dc.w      %0000000011111111 ; allow everything except ctrl codes


; now the item descriptions - they should be at the end of the file (cutoff flag
set)

coldesc dc.w      dcd2-*-2
        dc.b      'Get default colours from menu extensions?'
dcd2


mwcol_s dc.w      mwc-*-2
        dc.b      'Please choose the main window colourway'
mwc
```

```
gendes  dc.w      gd2-*-2
        dc.b      'Please the select the main working directory'
gd2

hexdes  dc.w      hd2-*-2
        'Input the character to be used as a hex sign'
hd2
```

3.2 An example of a configuration block, level 02, using the SMSQ/E macros

The SMSQ/E sources contain, in file *dev8_mac_config02*, a certain number of macros that help with generating a level 02 configuration block. In the example below I use "//" as introduction to a comment. This is done on purpose to distinguish these comments from real comments in an assembler file, as sometimes using comments with macros can generate errors.

This example was taken for the SMSQ/E *dev8_smsq_q68_hwinit_asm* file. I left out the labels with the "dc.b" etc. for the items themselves and some constants such as `q68.d4` defined elsewhere.

```
     mkcfstart                          // signal start

; SMSQ generic config items

       mkcfhead {SMSQ},{smsq_vers}      // make the config block header, ie.
                                        // config ID ,level, name,version

          mkcfitem 'OSPM',word,'M',qcf_mlan,,,\
          {Default Messages Language Code 33=F, 44=GB, 49=D, 39=IT},0,$7fff

// The mkfcitem makes an entire config item. Here, it has the item ID 'OSPM', is
// of item type word (=10), has the selection keystroke 'M', the item lies at
// label qcf_lan, and it has no pre- or post-processing routines (nothing
// between the commas). After the '\' come the description (within parentheses)
// and the attributes. The pointers to them will be generated by the macros.
// Note that the '\' sign is used to indicate that the parameters for the
// "mkcfitem" macro continue on the next line.

          mkcfitem 'OSPL',word,'L',qcf_lang,,,\
          {Default Keyboard Language Code 33=F, 44=GB, 49=D, 39=IT},0,$7fff

          mkcfitem 'OSPS',byte,'S',qcf_kstuf,,,\
          {Stuffer buffer key for edit line calls},0,$ff

noyes     mkcfitem 'OSPU',code,'U',qcf_curs,,,\      // here the \ is used twice
          {Use sprite for cursor?} \                 // description
          0,N,{No},1,Y,{Yes}                         // attributes

          mkcfitem 'OSPB',code,'B',qcf_bgio,,,\
          {Enable CON background I/O},.noyes
// Note how the attributes here are prefixed with a '.' and refer to a label
// previously defined. This means that the attributes pointer should point to
// the attributes that // are defined with the configuration item which lies at
// that label.

          mkcfitem 'OSPN',code,'N',qcf_ctrc,,,\
          {Use new CTRL+C switch behaviour},.noyes

        mkcfblend                   // end of this configuration block

// now follows the next configuration block
; Q68 specific config items

        mkcfhead {Q68},{smsq_vers}  // start a new configuration block

          mkcfitem 'OSPD',code,'D',qcf_ismode,,,\
          {Initial display mode}\
```

```
      q68.d4,4,{Normal QL Mode 4},q68.dl4,Q,{Large QL Mode 4},\
      q68.aur8,A,{8 bit Aurora},\
    q68.ds,S,{Small 16 bit},q68.md,M,{Medium 16 bit},q68.dl,L,{Large 16 bit}

      mkcfitem 'Q68A',code,0,qcf_bwin,,,\
      {Boot from}\
      1,1,{WIN1},2,2,{WIN2},3,3,{WIN3},4,4,{WIN4}\
      5,5,{WIN5},6,6,{WIN6},7,7,{WIN7},8,8,{WIN8}\
      9,F,{FAT1},0,N,{None}

      mkcfitem 'Q680',code,0,q68_led1,,,\
      {Switch LED on when SMSQ/E is initialising?},.noyes

      mkcfitem 'Q689',code,0,q68_led,,,\
      {Switch LED off when SMSQ/E is set up?},.noyes

      mkcfitem 'Q68L',code,0,q68_fst1,,,\
      {Card 1 : Use faster (40 MHz) SD Card speed if available?},.noyes

      mkcfitem 'Q68M',code,0,q68_fst2,,,\
      {Card 2 : Use faster (40 MHz) SD Card speed if available?},.noyes

      mkcfitem 'Q68N',code,0,q68_kbdtyp,,,\
      {Use standard home keys?},.noyes

    mkcfblend             //   end of this configuration block

  mkcfend                 //    signal end of configuration section
```

Have fun!

Wolfgang Lenerz