CONFIG

Configuration Information Specification

Content

1st edition – June 2014 W. Lenerz

Configuration Information Specification

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.

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 : Programs already exists that create config blocks for Basic programs.

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.

This document contains the definition of such a configuration standard.

Configuration operations and programs

When a config program starts up, the user selects the file to configure (which should contain one or more level 01 or level 02 config blocks). Level 01 blocks are treated as before (i.e. they can be printed or configured), but for level 02, there is an additional UPDATE facility.

The config program "learns" level 02 configurations and stores the settings of the item for any ID in a separate file, giving a "global" default configuration file. When the user selects UPDATE, the config block is scanned for IDs, and every ID is checked in the global default configuration file. If it is found, the preferred setting is automatically copied in the file which is to be configured. This way, updating programs is MUCH easier and nearly automatic. In fact, in could be made completely automatic (via parameter string).

Another advantage is, that the configuration can be made language-independent. The "learned" configuration of an English file could be used to configure a German or French file, for example, provided that the same items have got the same IDs. Care should be taken for items, which are language-dependent filenames (i.e. help-files, auto-save filenames etc.), which SHOULD have different IDs, otherwise the German program would save to an English file or vice versa.

MenuConfig

For the time being, only one program handles all currently existing configuration levels and the above way of dealing with config information: MenuConfig by Jochen Merz Software. This stores the configuration information in a file called MenuConfig_inf in the PROG_USE default directory. Local IDs are not stored by MenuConfig. When a user wants to update a file containing local IDs, then MenuConfig has to "learn" the old settings from the old (already configured) version of the file, and these settings are then updated to the new version of the file. The local IDs are not stored anywhere else, as this could lead to ID clashes between different files containing the same local ID for different purposes.

Configuration block

The configuration information is contained in a configuration block.

A configurable file may contain one or several configuration blocks. A configuration program should thus not stop after the first configuration block found in a file, but continue to scan it until the end of the file to be configured. The configuration blocks in an application program must be correct. Config programs may crash if this isn't the case.

There are, up to now, two configuration levels : level 01 and level 02. The configuration blocks for each level are slightly different. These blocks may also be preceded by a flag for a working copy.

The configuration block structure for level 01

This consists of 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 (relative, word)
    Pointer to item pre-processing routine (relative, word)
    Pointer to item post-processing routine (relative, word)
    Pointer to description of item (relative, word)
    Pointer to attributes of item (item type dependent)
End word (value -1)

The configuration block structure for level 02

The configuration structure for level 02 consists of the following information:

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

As you can see, there is an additional Item ID for each item, compared to level 01.

As time goes by, additional types of item, or additional levels, may be added. This will mean that new versions of the configuration program will be required. These new versions will, of course, be able to configure all lower level configurable files. But, if an old configuration program is used, and the level specified in the configuration block is greater than the level supported by the configuration program, it will have to give up gracefully.

The rest of this documents explains each element of the structure of a configuration block:

Configuration ID and configuration level

The **configuration ID** is word aligned and is 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”.

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"). The word aligned list of items follows.

Item ID

(level 02 only)
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 of level 01 was expanded to make room for a configuration tem ID, which allows for the previous configuration information to be applied to a program being configured.

The configuration 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.

For all ID names, a list, which is maintained by me, is created, to avoid multiple name conflicts. If you wish to register for one or more ID names, please email me. See below for the current list.

ID names consist of a longword (i.e. four characters). The first three characters should be registered, the fourth character can freely be assigned by the author / software house for the various items.

Types of item

Levels 01 and 02 support at least 7 types of item. These are: string, character, code selection, code as well as byte, word and long word values. Application specific types of item can be processed by treating them as strings which are handled entirely by an application supplied routine.

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 (config level 02)
bit 9	set if string is directory (config level 02)
bits 8 and 9	if both are set, string is an extension (config level 02)
bit 13	reserved for Prowess, If this is set, bits 12-8 may have other meaning

```
cfs.sspc     equ  %0000000000000001  string strip spaces
cfs.file     equ  %0000000100000000  string is filename
cfs.dir      equ  %0000001000000000  string is directory
cfs.e        equ  %0000001100000000  string is extension
cfs.pws      equ  %001xxxxx00000000  reserved for Prowess:
   cfw.pwfn  equ  %0010010000000000     Prowess filename
```

At present, the features corresponding to bytes 8 and 9 are supported only by programs handling level 02 (MenuConfig) and ignored by the old (standard) config.

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.

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).

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.

Values (types 8, 10, 12)
Largely self explanatory. The attributes are the minimum and maximum values. All values are treated as unsigned. Type 8 is a byte value, type 10 a word value and type 12 a long value.

Item Selection Keystroke
The item selection keystroke is an uppercased keystroke which will select the item in the main menu. 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 items, 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.

Pointer to Item
The pointer to item, and all the other pointers in the definition, are relative addresses stored in a word (e.g. `dc.w item-*`).

Pointer to Item Pre-Processing Routine

It is possible to provide a pre-processing routine within the main program which will be called, by the configuration program, 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. Note that the post-processing routine might be called before the pre-processing routine.

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. Please see below for additional information on the A3 space.

The application code may use up to 512 bytes of stack. It is the responsibility of the configuration program to provide for this stack space and the space pointed to by A3 before calling the pre-processing routine.

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

Pointer to Item Post-Processing Routine

It is possible to provide a post-processing routine within the main program which will be called, by the configuration program :

- for each item before configuration starts (i.e., in MenuConfig, before the configuration block is presented for editing),
- for each item after any other item is reset (d0>0 when exiting the post processing routine).

It can be used to set limits or other dependencies. This means that the post-processing routine is called once (for each item) as soon as the configuration of a config block starts, and before the configuration of each item. Then it is called every time an item is set, even if it is another item.

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. It is the responsibility of the configuration program to provide for this stack space and the space pointed to by A3 before calling the post-processing routine.

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 postprocessing routines will be called, and (for an interactive Configuration program) the item just set will be re-presented.

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

The space pointed to by A3 should also contain more information, which lies BELOW the base of the space as pointed to by A3. This is thus negatively indexed with reference to A3.

$0000	4k	base of workspace passed to pre/postprocessing routine
-$0004	long	(Menu)Config version
-$0008	long	primary channel ID
-$000c	long	pointer to working definition
-$0010	2 words	primary window x/y size
-$0014	2 words	primary window x/y origin
-$0018	2 words	work area x/y size
-$001c	2 words	work area x/y origin
-$001d	byte	text info window number in working definition
-$001e	byte	work info window number in working definition
-$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

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.

Pointer to attributes

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

Working copy flag

If the configured file contains a flag "<<QCFC>>" (i.e. 2 long words with these characters) immediately BEFORE the "<<QCFX>>" flag some configurations programs (e.g. MenuConfig) may offer the user the choice to save a configured version of the file without the configuration texts (description of items), to reduce the required file size to the minimum (as the configuration texts are not required anymore after configuration). Of course, a file treated this way cannot be configured afterwards anymore. Moreover, the description texts should be at the very end of file that is being configured. 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 items and texts are always the last section in your file, the texts coming last!!!

List of global and reserved IDs

List of global IDs

_COL	Main Colourway :	Byte : range -1, 0 to 3.
_COS	Sub-Window Colourway :	Byte : range -1, 0 to 3.
_COB	Button Colourway :	Byte : range -1, 0 to 3.
_COE	Explanation window Colourway : Byte : range -1, 0 to 3.	
_FFU	Flash-frequency for update icon : Byte : 0 (steady) or ticks	

List of reserved IDs

AGD.	Agenda
APP.	APPMan
ATA.	ATARI-Rext
BLC.	BASIC Linker
BLD.	Balled
BLG.	BASIC Linker
BLO.	BASIC Linker
CCT.	C68Tool (T. Roesner)
CSH.	CueShell
CDK.	CueDark
DDE.	DataDesign
DIS.	DISA V2
DRN.	Disk Rename
EMN.	EASYMenu
EMU.	ATARI-Emulator

EPM.	EPROM-Manager
ESP.	EASYSprite
EXT.	EASYExt
FiF.	FiFi
HLP.	HyperHelp
MBT.	MultiButton
MCF.	MenuConfig
MEN.	Menu Extension
MPK.	MultiPick
OSP.	Operating System Preferences (SMSQ)
PAD.	Notepad
PAR.	Paragraph (F. Lanciault)
PDF.	Page Designer 3 Fonts 1
PDf.	Page Designer 3 Fonts 2
PDG.	Page Designer 3 General
PDP.	Page Designer 3 Page
PDp.	Page Designer 3 Pattern
PF..	Proforma & Applications
PRP.	PrinterPanel
PRM.	QPrommer
PT_.	Pointer Interface Configuration
PW..	ProWes & Applications
PXF.	PFF device
QBS.	QBASIC
QDA.	QD (Tab Options)
QDE.	QD (Editor)
QDF.	QD (Files)
QDG.	QD (General)
QDS.	QDesign
QMK.	QMake
SQ_.	SMSQ Configuration
QPC.	QPC general
QPS.	QPC serial
QPP.	QPC parallel
QP1.	QPC1 specific
QP2.	QPC2 specific
Q2_.	QPAC2 Main
Q2S.	QPAC2 Sysdef
Q2F.	QPAC2 Files
Q2B.	QPAC2 Buttons
QSN.	QSnap
SST.	Systat
SYP.	System Password
SYS.	System
TAB.	QSpread
TAb.	QSpread
TaB.	QSpread
Tab.	QSpread
TRA.	TRA Extension
WED.	WIN Ed
WSL.	WIN Select

A perhaps more up-to-date list may be found at the SMSQ/E site (http://www.wlenerz.com/smsqe/ then goto Additional Information and data).