The following features are available on the following systems: QDOS, Minerva, SMS2, SMSQ and SMSQ/E.

Additional information on WM.ERSTR

This manual did not mention that there is a limit on the length of own error messages. An own error messages is easy to create:

```
LEA       own_msg,A0    ; get address
MOVE.L    A0,D0         ; into our "error" register
BSET      #31,D0        ; an error is negative
```

Now the limit: the length of the string is limited to 40 ($28) characters. If it is longer, "unknown error" is returned instead!

Additional information on WM.LDRAW

WM.LDRAW clears the change bit in the status are of every item which is selectively redrawn.

Undocumented SuperBASIC Procedure

SPTR has never been documented. Easy to guess, it does the same as IOP.SPTR, i.e. moves the pointer to a given position. The syntax is:

SPTR [#channel], xpos, ypos [,key]

Option	Default	Meaning
xpos, ypos	none	new pointer position
key	-1	origin key

The origin key should be zero if the pointer coordinates are absolute. A key of -1 will set the position relative to the current window definition. A key of 1 will set it relative to the hit area.

Undocumented selection keystroke for SuperBASIC

It is possible to put an underscore under a selection key for text loose menu items and text info items. To do this, specify the type to be text minus twice the underscore position. This means, to underscore the first character, give 0-2 (=-2), to underscore the fifth position give -10 etc.

The following features are supported on the following systems: SMSQ and SMSQ/E from Version 2.71 onwards only.

Additions to RPTR

The QPTR RPTR call has been modified to accept job events in the most significant byte of the termination parameter. The job event values are, therefore, multiplied by 256. Note that while all pointer events that have occurred since the call are returned in term%, only those job events (including pending events) which caused the return are returned in term%.

term% = $2001	**Return on button / keystroke or job event $20**
RPTR #ch, xabs%,yabs%, term%, swnum%, xrel%, yrel%, bt$	
	Event 32 is notified by another job so the wait is terminated and term% is set.
PRINT term% DIV 256	**Prints 32**

Additions to IOP.RPTR and Pointer Record

Bits 23 to 8 of the event vector in the pointer record are already used by the Window Manager. The 8 job events are, therefore, mapped into the most significant 8 bits (pp_jevnt) of the event vector within the pointer record and for the IOP.RPTR operating system call.

Note that while all pointer events that have occurred since the call are filled into pt_pevnt in the pointer record, only those job events (including pending events) which caused the return are filled into pt_jevnt.

New Pointer Event

Pointer event bit 6 (number 64) is now used to indicate that the pointer sprite has hit the edge of the screen.

sms.sevt		**Trap #1, D0=$3A**
Send Event		
	call	return
D1	destination job ID	destination job ID
D2.b	events to notify	preserved
Common error returns	err.ijob	

The events in D2 are sent the the destination job. If the job is waiting for one of these events, the job is released, otherwise the all the events are pended.

sms.wevt		**Trap #1, D0=$3B**
Wait for Event		
	call	return
D2.b	events to wait for	events causing return
D3.w	timeout (-1 is forever)	preserved
Common error returns	none	

The job waits for one or more of the events in D2 or the timeout. The events returned in D2 are removed from the job's pending event vector (event accumulator).

From WMAN V1.50 onwards, an additional Window Manager Vector has been introduced. Note: return on timeout will work on all systems, return on Job-Events only on SMSQ and SMSQ/E V2.71 onwards!

Vector $78		**WM.RPTRT**
Read Pointer		
	call	return
D2.B	job-events to return on	preserved
D3.W	timeout	preserved
A0		channel ID of window
A1		preserved
A2		preserved
A3		preserved
A4	pointer to working defn	preserved
A5	not used by any routine	
A6	not used by any routine	
Error returns:		
	Any I/O sub system errors	
	Any error returned by action or hit routine	

This routine does in general what WM.RPTR does, but allows the parameters D2 and D3 to be specified.

To access the new WMAN vector from BASIC, QPTR V0.14 (Disk Version number V0.29) or higher is required, which introduces a new keyword:

SuperBASIC Access routines

RD_PTRT wdef,item%,swnum%,event%,timeout%,xrel%,yrel%[,lflag%] {,aflag%[,ctx%][,cty%]}

RD_PTRT is a an extended version of RD_PTR which allows control over events and timeout. The lsbyte of event% contains the window events, the msbyte the job-events on call on which you wish the call to return. 255 (no job-events, all window events) is equivalent to RD_PTR. The timeout should be -1 for indefinite wait (equivalent to RD_PTR) or the number of ticks. On return, event% contains the event causing the return.

Changes in WMAN_TEXT_MAC

The file WMAN_TEXT_MAC contains a new macro: MKXUSTR.

It has the same syntax and needs the same parameters as MKXSTR, but it also generates an external MEU.name, which holds the position of the first occurance of the selection key in the name. This allows easy language-dependant items with the selection underscored keystroke at different positions in the text.

To set the type of an underscored item, XREF.S to MEU.name and define the type to be text-MEU.name.

MKXUSTR test,'S',{This is a test}

in the language-specific file would defined the externals

MEK.TEST to be 'S'
MET_TEST to be 'This is a test'
MEU.TEST to be 4

whereas in the window definition file the underscore is set by

TYPE TEXT-MEU.TEST

CONFIG Level 2

We felt that a number of things were missing in the definition of level 1 of the QJUMP Standard configuration definition. Therefore, after a number of discussions, the following suggestions were made to be implemented on level 2.

First of all, 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 is expanded to make room for an config-item-ID, i.e.

The configuration level 2 consists of the following information:

Configuration ID
Configuration level
Software name
Software version
List of
  Item ID (long) **<---- NEW!!!**
  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)

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 Jochen Merz Software is created, to avoid multiple name conflicts. If you wish to register for one or more ID names, please write to Jochen Merz Software and enclose an I.R.C. You may suggest one or more name, otherwise JMS will try to find a sensible abbreviation for you.

ID names consist of a longword (i.e. four characters). The first three characters have to be reserved by JMS, the fourth character can freely be assigned by the software house for the various items.

The function of the MENUCONFIG program

Please note: the MENUCONFIG program requires the MENU Extension (file MENU_rext) to be loaded. When the MENUCONFIG program starts up, the user selects the file to configure (which should contain one or more level 1 or level 2 config blocks). Level 1 blocks are treated as before (i.e. they can be printed or configured), but for level 2, there is an additional UPDATE facility. CONFIG "learns" level 2 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 sasme items have got the same ID's. Care should be taken for items, which are language-dependent filenames (i.e. help-files, auto-save filenames etc.), which SHOULD have different ID's, otherwise the German program would save to an English file or vice versa.

Local IDs are not stored by MENUCONFIG by default. You can configure MENUCONFIG from V3.21 onwards to enable the save of local IDs, but it may crash your system if you update files with the same "local" ID with different meaning, e.g. a string assignment is done to an ID which was defined as a word. There is no type check!!! We think it is safer not to save local IDs and update as follows: 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.

MENUCONFIG V2 stores the learned settings in a file called MenuConfig_INF on your current PROGram default device. It will try to read it from there the next time to execute MENUCONFIG. You can, of course, tell MENUCONFIG to load a different _INF file containing other configuration information, for example if you prefer having different configurations for colour and monochrome versions. When you terminate MENUCONFIG and you changed or learned new settings, MENUCONFIG asks you whether you want to update the _INF file, so that the settings are preserved for the next update.

An additional item type

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/postprocessing 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/postprocessing 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**

Working Copy

If the configured file contains a flag "<<QCFC>>" BEFORE the "<<QCFX>>" flag (which can be generated with the new Macro MKCFCUT) then MENUCONFIG offers the user the choice to save a configured version without the config texts, 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.
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!!!

List of Global ID's

_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.**
_FFU	**Flash-frequency for update icon**	**Byte**	**0 (steady) or ticks**

New Macro files CONFIG02_MAC and MULTICONFIG02_MAC

The new macro file CONFIG02_MAC behaves exactly like CONFIG_MAC, but generates config blocks level 2, which cannot be used with the standard CONFIG program (which is level 1), but, for example, with MENUCONFIG (which is level 2). MENUCONFIG requires the MENU Extension (MENU_rext) to be loaded as a resident extension in order to work.

MULTICONFIG02_MAC allows you to have more than one config block contained in the same source file. You need to introduce every config block seperately, and you can make full use of all the CONFIG and CONFIG02 features within every independent block, e.g.

```
mkcfstart       introcude multiple config blocks
 mkcfhead       ... as before for first config block
  ...             contents of first config block
 mkcfblend      end of first config block

 mkcfhead       ... as before, for second config block
  ...             contents of second config block
 mkcfblend      end of second config block

 ...            any number of additional config blocks
mkcfend         the end of all config blocks.
```

The new macro MKCFCUT defines the position from where config blocks can automatically be deleted after the configuration process (level 2 only). This means, further configuration will be impossible because this information is lost. The idea is to create shorter files after the configuration has been done, and to fix the configuration. Please note that the configuration blocks have to be last part of the file, because the file will be truncated from this position to the end. Make sure, that the configured items come before the configuration block!!!

Additions to the CONFIG standard

The attributes for strings have been extended, to allow menu-driven CONFIG programs better options for a selection, depending on the type. There are two additional bits used in the string attributes: 8 and 9. These define the type of string, so that the CONFIG program can treat these strings in a special way. The possible combinations are:

```
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, these features are supported by MENUCONFIG only, and ignored by the standard config.

%001xxxxx00000000 is reserved for PROGS (ProWesS).