Contents

Part I
The C68 libc Library

1 Introduction

Use of the libc68 library provides extensions are specific to the implementation of C68 on the QDOS or SMS operating systems. It will help you to exploit QDOS or SMS facilities to the full, but will mean that the programs you write will not be easy to transfer to other operating systems. You should bear this fact in mind when you decide to use the routines in the libc68 library.

The implementation of C68 for QDOS and SMS also provides routines to allow the C programmer to access all the Operating System Call interfaces directly. These are documented in the LIBQDOS_DOC (using the QDOS names for such calls) or the LIBSMS_DOC (using the SMS names for the calls) files.

You do not have to make any special provison at the link stage if you want to include routines from the libsms library. The routines defined as being in this library are actually imbedded in the LIBC_A library which is automatically included at the end of the link by the LD linker. You must however include either #include <qdos.h> or #include <sms.h> in any program or module that use the routines defined in the libc68 library. Which of the two you include is not material (you can include both!), and will probably be determined by whether you intend to use QDOS or SMS names for any calls you make directly to the operating system interface.

1.1 Mixing C and QDOS/SMS Input / Output

If you wish to be able to use both C and QDOS/SMS level input/output calls to refer to the same file/device then it is imperative that you issue a 'setbuf' call (defined in stdio_h) to disable internal buffering within the C standard input/output routines, or use the fflush() call before switching from C level I/O to QDOS/SMS level I/O. Failure to do this can result in input/output reacting in unexpected ways.

1.2 Reference Material

The reference books listed below were used in preparing material for inclusion in this library:

- "QL Technical Guide" by David Karlin and Tony Tebby
- "QL Advanced User Guide" by Adrian Dickens
- "QDOS Reference Manual" as published by Jochen Merz

2 Library Routines

The following pages contain a list of all the routines contained in the C68 libc68_a library. These are routines that are specific to this QDOS or SMS implementations of C68. It is organised as a short list by function, and a longer list in alphabetical order.

2.1 File / Directory Handling

chddir chpdir fgetchid fnmatch fqstat fusechid getcdd getchid getcname getcpd getfnl opene open_qdir qdir_delete qdir_read qdir_sort qstat read_qdir usechid

2.2 Screen Input / Output

c_extop iop_outl

2.3 Sound

beep do_sound

2.4 Conversion

cstr_to_ql d_to_qlfp i_to_qlfp l_to_qlfp qlfp_to_d qlfp_to_f qlstr_to_c w_to_qlfp

2.5 String Handling

qstrcat qstrchr qstrcmp qstrcpy qstricmp qstrlen qstrncat qstrncmp qstrncpy qstrnicmp ut_cstr

2.6 Miscellaneous

baud iscon isdevice isdirchid isdirdev isnoclose keyrow poserr qdos1 qdos2 qdos3 qinstrn stackcheck stackreport waitfor _CacheFlush _ProcessorType _super _superend _user

2.7 Global Vectors

_bufsize _cmdchannels _cmdparams _cmdwildcard _endmsg _memincr _memmax _memqdos _mneed _oserr _pipesize _prog_name _stack _stackmargin _sys_var def_priority os_nerr os_errlist

2.8 Reference

2.8.1 *void argfree (char ** argv[])*

Routine to free all the memory that is associated with an *argv[]* style vector created using the *argunpack()* routine. This frees the memory associated with the argument strings as well as that associated with the argument vector itself.

2.8.2 *char * argpack (char * argv[], int flag)*

Routine to create a command line from an argv[] style vector. This is the complimentary routine to *argunpack()*. The command line will consist of the arguements from the *argv[]* vector separated by spaces. If the *'flag'* parameter is set then it will be assumed that the command line is for a C68 program, and the arguments will be processed so that quotes are added around them if they contain white space, and any embedded non-printable characters are converted to C escape sequences. If the flag is not set, then each argument is simply added unprocessed. The memory for the command line is allocated dynamically via *malloc()*.

The value returned is the address of the resulting command line. If any error occurs (typically no memory left) then *NULL* is returned.

It is expected that the main use of this routine will be internally within other library routines, but it is made available for any system programmers.

2.8.3 *int argunpack(char * cmdline, char ** argv[], int * argc, int (* function)(char *, char ***, int *))*

Routine to create an *argv[]* vector from a command line. This is the complimentary routine to *argpack()*. If any argument is surrounded by quotes these will be removed. Also, any enbedded C escape sequences will be converted into their internal values. The *'argc'* parameter will be used to return a count of parameters put into the array less one (i.e. 0 means one value in the array).

The *'function'* parameter is used to pass the address of a secondary routine that can be used to process further any argument before it is put into the array. A typical example of such a function might be the one that is used to do wild card expansion of parameters on the command line. If this function returns 0 then that means that it did nothing with the argument passed, and the *argunpack()* routine should add the value itself to the *argv[]* array. A return value of -1 inidicates an error occurred, and any positive value means that the function has handled the argument internally. The function' parameter can also be *NULL* to indicate that no additional processing needed of arguments.

The value returned is the number of arguments actually put into the array. If any error occurs (typically no memory left) then -1 is returned.

It is expected that the main use of this routine will be internally within other library routines, but it is made available for any system programmers. This is the routine that is used within the program startup code to parse the command line.

2.8.4 *void beep(duration, pitch)*

QDOS routine to make a quick beep, given duration in 50 (or 60) Hz ticks, and pitch (from 0 to 255).

2.8.5 *int c_extop (chanid_t channel, timeout_t timeout, int (*func), int number_of_params, ...)*

This routine allows a routine to be called to do an extended operation on a QDOS or SMS channel. The parameters are passed in a way that is compatible with this routine being written in C (c.f. *sd_extop()/iow_xtop()* for assembler only routines).

The C routine will be called in supervisor mode, with the parameters specified by ... above passed to it on the stack. Each parameter is assumed to be no larger than 4 bytes in size (i.e. no structures are to be passed on the stack).

NOTE It appears that QDOS cannot correctly handle error codes being returned in D0. Therefore the only values that should be returned are 0 or -1 (for operation not completed). If it is desired to pass an error code back to the application program it must be done indirectly via one of the parameters.

2.8.6 *int chddir(char *str)*

Changes the current destination directory (the one set by TK2 `SPL_USE` command in SuperBasic). If passed *NULL* then tries to go up a level. If passed a string starting with a device then replaces the current directory, else appends to current directory (adding _ at end if needed). Maximum length is 31 characters.
Returns 0 if ok, !0 if failed.

2.8.7 *int chpdir(char *str)*

Changes current program directory (the one set by TK2 `PROG_USE` command in SuperBasic). If passed *NULL* then tries to go up a level. If passed a string starting with a device then replaces the current directory, else appends to current directory (adding _ at end if needed). Maximum length is 31 characters.
Returns 0 if ok, !0 if failed.

2.8.8 *QLSTR_t * cstr_to_ql(QLSTR_t * ql_string, char * c_string)*

Routine to convert a C (zero terminated) string to a *struct QLSTR* (defined in qdos.h), a QL string with length first followed by the string. This routine is NOT safe to convert a C string in situ, eg.

```
cstr_to_ql ((QLSTR_t *)str, str)
```

will fail badly (the C string will become corrupt). Returns the address of the QL string.

2.8.9 *void do_sound(int duration, int pitch, int pitch2, int wrap, int g_x, int g_y,int fuzz, int random)*

QDOS call to make a sound. Parameters defined as for SuperBasic beep call.

2.8.10 *QLFLOAT_t * d_to_qlfp(QLFLOAT_t * qlf, double val)*

Routine to convert IEEE double precision (8 byte) floating point number to a QL floating point number. Returns the address of the *QLFLOAT* passed as the first parameter.

2.8.11 *long fgetchid(FILE *fp)*

Returns QDOS channel id of *FILE* pointer. Returns -1L on error Defined in stdio.h

---

2.8.12 *int fnmatch(char *fname, char *wildname)*

Non-recursive routine to match a QDOS wildcard. Similar to Unix style widlcard matching to make it more useful for GREP and 'C' programmers.

Examples of match

- `*_c` matches names *ending* with `_c` only (eg. `test_c` but NOT `test_c_doc`)
- `wom*_o` matches `wombat_o` but NOT `wombat_obj`
- `*tes*_vi*_obj` matches `flp1_wombat_test_yy_vile_obj` but NOT `flp1_wombat_testvile_obj`

Returns 1 if match, 0 if no match

2.8.13 *int fqstat(int fd, struct direct * stat)*

QDOS specific variant of *fstat()* call. Normally it would be recommended that you used the *fstat()* call instead as this is more portable. Gets the file information from QDOS, given a level 1 file descriptor. Returns the exact same information as in a QDOS directory entry (Note times are in QDOS format, not C format). The structure *'direct'* is defined in 'qdos.h'.

Return values:

0 success

-1 Standard C error code set in errno (as defined in errno.h)

other QDOS error code (as defined in qdos.h).

2.8.14 *FILE * fusechid (chanid_t channel)*

Create a Level 2 File Pointer for a file opened at Level 0 (the QDOS level) via the *io_open()* call. Also creates a level 1 file descriptor entry. Must *not* be called more than once for a given file.

Return values:

+ve FILE pointer

NULL failed

-ve details in errno

2.8.15 *char *getcdd(char *str, int size)*

Gets current destination directory path (as set by TK2 SPL_USE command) into buffer *str*. *If str == NULL* then allocates a buffer of length *size* using *malloc* and returns address of it. Returns *NULL* on error, else address where name is stored.

2.8.16 *chanid_t getchid(int fd)*

Gets QDOS channel id for level 1 file descriptor.
Return values: -1 error occurred - details in errno +ve QDOS channel id

---

2.8.17 *char *getcname (chanid_t channel, char *buffer)*

Obtains the name of a device associated with a QDOS channel and places it in the buffer.
Return values: +ve Pointer to the name NULL error occurred - details in errno.

2.8.18 *char *getcpd (char *str, int size)*

Gets current program directory path (as set by TK2 PROG_USE command) into buffer str. If str == NULL then allocates a buffer of length size using malloc and returns address of it.
Return values: NULL error occured - details in errno. +ve address where name is stored.

2.8.19 *int getfnl (char *wcard, char *fna, unsigned fnasize, int attr)*

wcard; - Wild card string to use, or NULL for all the files in the data directory
**fna*; - Area to hold returned list of file names
fnasize; - Size of file name area
attr; Search attributes. - Can be added together to provide criteria.

0 - all files (QDR_ALL)

1 - data only (QDR_DATA)

2 - prog only (QDR_PROG)

4 - directory only (QDR_DIR)

Symbolic names defined in qdos.h
Lattice compatible routine to get a list of filenames, separated by '\0' character. List terminated by an additional '\0' character.
Return values: -1 error occurred other number of names read.
Defined in stdlib.h. See also *read_qdir()*.

2.8.20 *int iop_outl (chanid_t channel, timeout_t timeout, short, short, short, void *)*

This is the call that sets the outline window for a Pointer Environment. It is included in this library as it is the one call that needs to be issued to make a program that is not otherwise aware of the pointer environment function correctly in that environment.

For more details refer to the LIBQPTR_DOC file provided as part of the QPTR library.

Note that the default console initialisation routines supplied with C68 will automatically issue a call to set the window outline to the size as defined in the '*_condetails*' global variable (see end of this document).

2.8.21 *int iscon (chanid_t long channel, timeout_t timeout)*

returns 1 if the channel specified by *channel* is a console (con_) device, 0 if not

---

2.8.22 *int isdevicem (char *str, int *extra)*

Routine to check if a string starts with a device name. TRUE if it is, with extra info in the *'extra'* parameter passed as well as the name, 0 if it's not. Actually searches system lists. **extra* can be DIRDEV (device is on directory driver lists) or DIRDEV | NETDEV (device is on a network - may not be directory device on remote machine). DIRDEV and NETDEV are defined in qdos.h.

2.8.23 *QDDEV_LINK_t * isdirchid (chanid_t channel_id)*

Routine to find out if a channel belongs to a directory device or not. If not, NULL is returned. If it does, then a pointer to the Device Driver Definition block is returned. This can then subsequently be used to find out the device type if required by looking at the name field in this Device Driver Definition block.

2.8.24 *int isdirdev (char *str)*

Routine to check if a string starts with a name corresponding to a directory device. Returns 0 if not.

The value returned has the same meaning as the *'extra'* parameter returned by the *isdevice()* routine.

2.8.25 *int isnoclose (int file_descriptor)*

Used to determine if the channel associated with a level 1 file descriptor was passed to this job on the stack (via the command line)q. Return values are:

-1 file does not exist

1 channel for this file was passed on the stack

0 channel for this file was not passed on the stack

2.8.26 *QLFLOAT_t * i_to_qlfp (QLFLOAT_t * qlf, int i)*

Fast routine (faster than inbuilt QDOS routine) to convert a integer into a QL floating point number. Returns the address of the QLFLOAT passed as the first parameter.

2.8.27 *int keyrow (int row)*

QDOS routine to read the QL keyboard directly. Equivalent to SuperBasic `KEYROW` with all attendant warnings. Does not set *_oserr*.

Row								
0	7	4	F5	F3	F2	5	F1	F4
1	DOWN	SPACE	\	RIGHT	ESC	UP	LEFT	ENTER
2	‘‘	M	£	B	C	.	Z	}
3	;	G	=	F	S	K	CAPS	[
4	J	D	P	A	1	H	3	L
5	O	Y	-	R	TAB	I	W	9
6	U	T	0	E	Q	6	2	8
7	,	N	/	V	X	ALT	CTRL	SHIFT

KEYROW layout on UK QL

2.8.28 *QLFLOAT_t * l_to_qlfp (QLFLOAT_t * qlf, int i)*

Fast routine (faster than inbuilt QDOS routine) to convert a long integer into a QL floating point number. Returns the address of the QLFLOAT passed as the first parameter.

2.8.29 *int opene (char *name, int mode, int paths)*

Routine to search more than just the default directory if *name* does not start with a device. If it does then that is opened, else if *paths*

==3 search program directory, then data directory

==2 just search program directory

==1 search data directory first, then program directory

==0 just search data directory (as *open()*)

Returns -1 on error, valid fd if OK. Defined in fcntl.h

2.8.30 *chanid_t open_qdir (char *name)*

Opens a directory on a device. Returns a negative value (the QDOS error code) if an error occurred at the QDOS level, 0 if any other error occurred (in which case *'errno'* contains the error code) or a positive channel id on success.

2.8.31 *int poserr (char *s)*

The QDOS specific equivalent of the standard C *'perror'* routine. Prints the error text relating to the QDOS error code in *_oserr*.

2.8.32 *void qdir_delete (DIR_LIST_t * list)*

Deletes all space allocated by a call to the *qdir_read()* routine.

2.8.33 *DIR_LIST_t * qdir_read (char *devwc, char *stext, int attr)*

devwc - Device and wildcard

stext: - Sort text

attr: - File types to get

0 = all,

1 = data,

2 = prog,

4 = directory

Routine to open, read and sort a QDOS directory. Sort text is the same as QRAM, N(ame) U(se) S(ize) D(ate) T(ime) lower case reverses sense of search.

The *DIR_LIST_t* structure is defined in sys_qlib.h. All space for directory entries and names is allocated via *malloc()* - it should be released when you have finished with it by calling *qdir_delete()*.

Return values: NULL No match found (or error occured if errno set) other pointer to list

```
typedef struct DIR_LIST {
   struct DIR_LIST *dl_next;
   struct qdirect dl_dir;
   char dl_cname [1]
} DIR_LIST_t;

typedef struct qdirect {
   unsigned long d_length;
   unsigned char d_access;
```

```
   unsigned char d_type;
   long d_reserved;
   unsigned short d_szname;
   char dname [36];
   long d_update;
   union {
      long d_refdate;
      struct {
         unsigned short d_version;
         unsigned short d_fileno;
      } v2;
   } u;
   long d_backup;
} qdirect_t;
```

2.8.34 *DIR_LIST_t * qdir_sort (DIR_LIST_t *list, char *stext, char (*dcomp)())*

list: Existing linked list

stext: Sort parameters

dcomp: Compare routine (default routine used if *dcomp* == NULL)

Routine to sort linked list of extended QDOS directory structure. Returns pointer to first of list.

Sort text is string containing: N or n sort on ascii name. U or u sort on file usage. S or s sort on file size. D or d sort on file date. T or t sort on file time. Uppercase = ascending sort, Lowercase = descending

If *strlen(stext)* > 1 then each sort is done in turn.

A default compare routine is used internally by *qdir_read()*. This is described below in case anyone wants to write better compare routines.

Specification of dcomp is:

*int (*dcomp)(DIR_LIST_t * d1, DIR_LIST_t *d2, char *sort_text)*

Return value indicates comparison result: +ve d1 > d2 0 d1 == d2 -ve d1 < d2

2.8.35 *long qdos1 (REGS_t *in, REGS_t *out)*

2.8.36 *long qdos2 (REGS_t *in, REGS_t *out)*

2.8.37 *long qdos3 (REGS_t *in, REGS_t *out)*

Lattice compatible routines to call specific operating systemp traps with registers set up as in a *REGS_t* structure. Not normally needed with C68 as there are routines in the libraries to call most QDOS and/or SMS traps directly. These routines cater for any that might be missing, and also provide compatibility with QLC which used these routines for accessing QDOS facilities. Returns the value of register D0.

```
typedef struct REGS {
   datareg_t D0, D1, D2, D3;
   addreg_t A0, A1, A2, A3;
} REGS_t;
```

2.8.38 *int qfork (...)*

Starts another process concurrently with the calling one. The new process starts with a default priority found in external variable *_def_priority*. Returns the process id of new process or error code. Sets *errno* (and if relevant *_oserr*). The arguments (other than *'owner'*) have the same meaning as in the *exec()* and *fork()* family of calls.

These are variants of the *fork()* family of calls (that are defined in LIBUNIX_DOC). The difference is that the *qfork()* variants allow the owner of the new process to be specified whereas with the *fork()* set the current job is always the owner. This is important under QDOS or SMS if you do not want the daughter job to be automatically terminated by the operating system when the parent job terminates. If you specify zero as the parent job then the daughter job is in complete control of its own destiny!

*pid_t qforkv(jobid_t owner, char * name, int * file_desc, char * argv[])*

*pid_t qforkvp(jobid_t owner, char * name, int * file_desc, char * argv[])*

*pid_t qforkl(jobid_t owner, char * name, int * file_desc, char * argvs, ...)*

*pid_t qforklp(jobid_t owner, char * name,int * file_desc, char * argvs, ...)*

The directories searched in each case are as follow:

qforkv program directory only

qforkvp program directory and then data directory

qforkl program directory only

qforklp program directory and then data directory

The qforkl() and qforklp() routines must have a *NULL* parameter to terminate their parameter lists.

2.8.39 *int qinstrn (char * string, int max)*

Function to type a C style string into the current keyboard queue (c.f. the Turbo Toolkit command TYPE_IN).

On success returns the number of characters typed in, on failure returns a QDOS error code.

2.8.40 double qlfp_to_d (QLFLOAT_t * qlfp)

Routine to convert the 6 byte representation of floating point numbers used on QDOS and SMS systems to the IEEE 8 byte floating point format used internally by C68 for doubles.

2.8.41 *long qlfp_to_f (QLFLOAT_t * qlfp)*

Routine to convert the 6 byte representation of floating point numbers into the bit pattern corresponding to an IEEE floating point number as a long. This is NOT the routine to use if you merely wish the result to be assigned to a C 'float' variable - use *qlfp_to_d()* instead.

2.8.42 *char *qlstr_to_c (char *c_string, QLSTR_t * ql_string)*

Routine to convert a QDOS or SMS string (length first, followed by string) (the *QLSTR_t* strucutre is defined in sys/qlib.h which is included by both qdos.h and sms.h) to a C string (zero terminated). Note that this routine is safe to call to convert a QDOS or SMS string in situ eg. *qlstr_to_c(q_string, (char *)q_string)* as nothing is corrupted.

```
typedef struct QLSTR {
   short qs_strlen;
   char qs_str [1];
} QLSTR_t;
```

2.8.43 *int qopen (const char *, int mode, ...)*

This is a variant of the *open()* routine that is specifically designed to make it easy to handle filenames that originate on foreign systems. These foreign systems often have special characters in their filenames to indicate sub-directories or file extensions. On a QDOS or SMS system one would typically use underscores for both these purposes. This routine handles an automatic conversion between these different types of name in a relatively transparent manner.

If the filename supplied does not contain any of the special characters '.' (fullstop), '/' (forward slash) or '\' (backward slash) then this routine is functionally identical to the *open()* routine. If the filename supplied does contain any of the special characters then the way it operates depends on whether the file is being opened with a READ ONLY mode or some variant of a WRITE mode:

READ A copy of the filename is made with the special characters replaced by underscores and an attempt made to open the file with this revised filename (i.e. the typical QDOS/SMS variant is tried first). If this fails, then the original name as supplied is tried as well.

WRITE A check is made to see if a file with the name as supplied is present, and if so this name is used (i.e. the foreign name is tried first). If such a file is not present then a copy of the name is made and the special characters replaced by underscores. This name is then used to open the file.

Note.

The *qopen()* routine would typically be used in conjunction with the *_Open* vector described later in this document.

2.8.44 *int qstat (char *name, struct qdirect *buffer)*

Routine to get file information given the filename. This is a QDOS and SMS specific variant of the *stat()* call. It is recommended that you try and use the *stat()* call in preference as this is more portable. The *qdirect* structure is defined in sys/qdos.h which is included by both qdos.h and sms.h.

Return values:

0 Success

-1 Standard C error code set in *errno* (as defined in errno.h).

other QDOS error code (as defined in qdos.h)

2.8.45 *QLSTR_t * qstrcat (QLSTR_t * target, const QLSTR_t * source)*

Concatenate two QDOS or SMS strings. Similar to the C routine *strcat()* except that it operates on QDOS and SMS strings. The *QLSTR_t* structure is defined in sys/qlib.h which is included by both qdos.h and sms.h.

The target string will also have a *NUL* byte appended to the end (although this will not be included in the length count) so that it is possible to treat the text part as a C string.

2.8.46 *int qstrchr (const QLSTR_t * target, int ch)*

Search a QDOS or SMS string for a specified character. Similar to the C routine *strchr()* except that it operates on QDOS and SMS strings. The *QLSTR_t* structure is defined in sys/qlib.h which is included by both qdos.h and sms.h.

The value returned will be the address of the character, or NULL if the character was not found.

2.8.47 *int qstrcmp (const QLSTR_t * string1,const QLSTR_t * string2)*

Compare two QDOS or SMS strings for equality. Similar to the C routine *strcmp()* except that it operates on QDOS and SMS strings. The *QLSTR_t* structure is defined in sys/qlib.h which is included by both qdos.h and sms.h.

The QDOS/SMS collating order is used to determine the less than/greater than return conditions.

2.8.48 *QLSTR_t * qstrcpy (QLSTR_t * target, const QLSTR_t * source)*

Copy a QDOS or SMS string. Similar to the C routine *strcpy()* except that it operates on QDOS and SMS strings.

The *QLSTR_t* structure is defined in sys/qlib.h which is included by both qdos.h and sms.h.

An additional NUL byte is added to the end of the target string (although not included in the length count) so that it is possible to treat the *text* part as a C string. This extra byte must be allowed for in determining the required size of the target area.

2.8.49 *int qstricmp (const QLSTR_t * string1, const QLSTR_t * string2)*

Compare two QDOS or SMS strings for equality ignoring case. Similar to the C routine *stricmp()* except that it operates on QDOS and SMS strings.

The *QLSTR_t* structure is defined in sys/qlib.h which is included automatically by qdos.h or sms.h.

The QDOS/SMS collating order is used to determine the less than/greater than return conditions.

2.8.50 *int qstrlen (const QLSTR_t * target)*

Get the length of a QDOS or SMS string. Similar to the C routine *strlen()* except that it operates on QDOS and SMS strings.

The *QLSTR_t* structure is defined in sys/qlib.h which is included automatically by both qdos.h and sms.h.

2.8.51 *QLSTR_t * qstrncat (QLSTR_t * target, const QLSTR_t * source, size_t maxlength)*

Concatenate one QDOS or SMS string to another one up to a specified length. Similar to the C routine *strncat()* except that it operates on QDOS and SMS strings.

The *QLSTR_t* structure is defined in sys/qlib.h which is included autoamtically by both qdos.h and sms.h.

The target string will also have a *NUL* byte appended to the end (although this will not be included in the length count) so that it is possible to treat the *text* part as a C string.

2.8.52 *int qstrncmp (const QLSTR_t * string1, const QLSTR_t * string2, size_t maxlength)*

Compare two QDOS or SMS strings for equality up to a maximum length. Similar to the C routine *stricmp()* except that it operates on QDOS and SMS strings.

The *QLSTR_t* structure is defined in sys/qlib.h which is included automatically by both qdos.h and sms.h.

The QDOS/SMS collating order is used to determine the less than/greater than return conditions.

2.8.53 *QLSTR_t * qstrncpy (QLSTR_t * target, const QLSTR_t * source, size_t maxlength)*

Copy a QDOS or SMS string up to a maximum length. Similar to the C routine *strncpy()* except that it operates on QDOS and SMS strings.

The *QLSTR_t* structure is defined in sys/qlib.h which is included automatically by both qdos.h and sms.h.

An additional NULL byte is added to the end of the target string (although not included in the length count) so that it is possible to treat the *text* part as a C string. This extra byte must be allowed for in determining the required size of the target area.

2.8.54 *int qstrnicmp (QLSTR_t * string1, QLSTR_t * string2, size_t maxlength)*

Compare two QDOS or SMS strings for equality ignoring case up to a specified length. Similar to the C routine *strnicmp()* except that it operates on QDOS and SMS strings.

The *QLSTR_t* structure is defined in sys/qlib.h which is included automatically by both qdos.h and sms.h.

The QDOS/SMS collating order is used to determine the less than/greater than return conditions.

2.8.55 *int read_qdir (chanid_t chid, char *devwc, char *ret_name, struct direct *ret_dir, int attr)*

chid: QDOS channel id for directory

devwc: device and wildcard

ret_name: Name to return

ret_dir: Directory structure to read into

attr: Types to read: 0 = all, 1 = data, 2 = prog, 4 = directory

Reads the next directory entry matching a specified wildcard and attribute. If first part of wild matches _dir_ then only last part of the name is returned.

Return values:

1 Success

0 End-of-file reached

-1 Error as indicated by *errno*

See also *getfnl()*.

2.8.56 *int sendsig (chanid_t chid, jobid_t jobid,int signo, int priority, u_int uval)*

Low level routine to send a signal to the SIGNAL device driver. Returns 0 on success, or QDOS/SMS error code on failure.

Defined in signal.h

2.8.57 *int set_timer_event (struct TMR_MSG *msg)*

Signal related routine that returns *msg.len* if a previous event was cancelled, 0, or QDOS error

Defined in signal.h

2.8.58 *int sigcleanup()*

Routine that should be called only when leaving a signal handler through *longjmp()*. It inhibits reassigning of the handler and *sigprocmask* and calls *_CheckSig()*.

It is normally better to call the Posix defined routines *sigsetjmp()* and *siglongjmp()* instead of *setjmp()* and *longjmp()* as then this routine is not required.

Returns 0 on success, QDOS/SMS error code on failure.

Defined in sys/signal.h

2.8.59 *int stackcheck ()*

This routine acts like *stackreport()*, except that if the margin is breached, a 0 value is always returned (rather than a negative value). This means you can easily test for failure using *assert* statements of the form

assert(stackcheck());

in your program, and an assert error message is generated if it fails.

2.8.60 *long stackreport ()*

Report the current amount of stack available before the safety margin (as specified by the global variable *_stackmargin*) is reached. A negative value means that you are below the safety margin by the specified amount, and are could well be corrupting your data areas.

A program crash (or even system crash if you are unlucky) is probably imminent!

2.8.61 *int strfnd (char *tofind, char *tosearch)*

Find the position of string *'tofind'* in string *'tosearch'* doing case independent match. Returns -1 if not found, position in string if found.

Note that if you want a case dependant version you should use the Unix compatible *strfind()* routine (defined in LIBUNIX_DOC).

Defined in string.h

2.8.62 *void strmfe (char * newname, const char * oldname, const char * extension)*

Lattice compatible routine to take a filename, remove any existing extension, and then to add the given extension.

Defined in string.h

2.8.63 *void strmfn (char * newname, const char * drive, const char * path, const char * basename, const char * extension)*

Lattice compatible routine to build up a filename from its components. Any required underscore seperator characters will be added automatically. Any of the components can be a zero length string if not required.

Defined in string.

2.8.64 *void strmfp (char * newname, char * path, char * name)*

Lattice compatible routine to build a filename from its path and base name. If needed an underscore character will be added between the *'path'* and *'name'* components. The *'path'* string can be an zero length string.

Defined in string.h

2.8.65 *int usechid (chanid_t channel)*

Create a Level 1 file descriptor for a file opened at level 0 (ie the QDOS level using *io_open()*). Must not be called more than once for a given file.

Returns file descriptor if successful, -1 on failure.

2.8.66 *QLFLOAT_t * w_to_qlfp (QLFLOAT_t * qlf, int w)*

Routine to convert a short integer (word) to a QL floating point number. This routine is included mainly for completeness as normally you would use the *i_to_qlfp()* routine.

2.8.67 *int waitfor (jobid_t jobid, int * ret_value)*

Wait for the specified job to terminate.

If the '*ret_value*' parameter is not NULL, then it should point to the address at which the exit code of the specified job should be put.

Returns 0 on success, -1 if specified job could not be found.

2.8.68 *void _CacheFlush (void)*

Routine to force a flush of the cache on 68030 (or higher) processors - will do nothing on 68020 or less. Needs to be used if you ever have self-modifying code. Ths means that it should very rarely be used in practise!

2.8.69 *int _ProcessorType (void)*

Routine to determine the type of processor you are running on. If the system variable that specifies the processor type is set, then this value is returned. If the system variable is not set then tests are done to determine the processor type, the value is stored in the system variable and also returned. The values returned will indicate the basic processor type as follws:

$00 68000/68008

$20 68020

$30 68030

$40 68040

In addition the following bits can be 'or'ed to the above values to indicate special features:

$01 Internal MMU

$02 68851 MMU

$04 Internal FPU

$08 68881/68882 FPU

Experience has shown however, that one cannot guarantee that the bits inidcating extra features will always be set up - and there is quite a bit of code around that works on the assumption they will not be set up.

2.8.70 *void _super()*

Routine to go into supervisor mode. You MUST return to user mode before exiting the function in which you went into supervisor mode or you will probably crash the machine.
WARNING This function should be used with great care, and only if absolutely essential.

2.8.71 *void _superend()*

Routine to go back into user mode after having been in Supervisor mode. Does not check if any signals have occurred.
Complementary function to *_super()* (see also *_user()* routine).

2.8.72 *void _user()*

Routine to go back into user mode after having been in Supervisor mode. Checks if any signals have occurred while in supervisor mode, and if so handles them.
Complementary function to *_super()* (see also *__superened()* routine).

2.9 Structures, Macros and Typedefs

These are various definitions in the sys/qlib.h header file that are used when refering to QDOS or SMS based systems. This header file is included automatically by the qdos.h, sms.h and qptr.h header files.

2.9.1 *JOBHEADER* and *JOBHEADER_t*

These are the structure name and typedef respectively that are used to refer to a QDOS/SMS job header.

```
typedef struct JOBHEADER {
   long jb_len;
   long jb_start;
   jobid_t jb_owner;
   long jb_hold;
   unsigned short jb_tag;
   unsigned char jb_prior;
   unsigned char jb_princ;
   short jb_stat;
   char jb_rela6;
   char jb_wflag;
   jobid_t jb_wjob;
   QLVECTABLE_t * jb_trapv;
   struct FULLREGS jb_regs;
   short jb_resvd;
} JOBHEADER_t
```

2.9.2 *QLFLOAT* and *QLFLOAT_t*

These are the structure name and typedef respectively that are used when refering to QL/SMS format floating point numbers.

```
typedef struct QLFLOAT {
   short qfp_exp;
   long qfp_mant;
} QLFLOAT_t;
```

2.9.3 *QLRECT* and *QLRECT_t*

These are the structure name and typedef respectively that are used to define the width, height and origin of a rectangular area on a screen.

```
typedef struct QLRECT {
   unsigned short q_width;
   unsigned short q_height;
   unsigned short q_x;
   unsigned short q_y;
} QLRECT_t;
```

2.9.4 *QLSTR* and *QLSTR_t*

These are the structure definition and typedef respectiviely used to refer to QL/SMS string types. They are defined in the sys/qlib.h header file.

```
typedef struct QLSTR {
   short qs_strlen;
   char qs_str[1];
} QLSTR_t;
```

2.9.5 *QLSTR_DEF (name, length)*

Macro to define the space for a QL/SMS style string in a QLSTR style structure. Typically used in a statement of the form:
QLSTR_DEF (string_name,20);
You can also give the string an initial value by using a statement of the form:
QLSTR_DEF (string_name,20) = {5,"Hello");
However if the string will never be changed, you will find it easier to use the *QLSTR_INIT* macro.

2.9.6 *QLSTR_INIT (name, "value")*

Macro to define a constant initialised QL/SMS style string. Typically used in a statement of the form:
QLSTR_INIT (string_name,"Hello");
The space allocated will allow for the *NULL* byte that is used to terminate a C style string, but the *NULL* byte will not be included in the count of characters in the QL/SMS part.
If you use this macro inside a function then you need to precede it with the static keyword.

2.9.7 *TIME_QL_UNIX (ql_time_in_seconds)*

Macro to convert a QDOS/SMS time in seconds (measured since 1st January 1961) into a Unix time in seconds (measured since 1st January 1970).

2.9.8 *TIME_UNIX_QL (unix_time_in_seconds)*

Macro to convert a Unix time in seconds (measured since 1st January 1970) into a QDOS/SMS time in seconds (measured since 1st January 1961).

2.9.9 *WINDOWDEF* and *WINDOWDEF_t*

These are the structure and typedef respectively that are used to define the details of a screen window.

```
typedef struct WINDOWDEF {
   unsigned char border_colour;
   unsigned char border_width;
   unsigned char paper;
   unsigned char ink;
   unsigned short width;
   unsigned short height;
   unsigned short x_origin;
   unsigned short y_origin;
} WINDOWDEF_t;
```

2.10 Global Variables

The following are global variables that are available to user programs. Some of them are for information only while others can be set in user programs to control certain default settings of elements of the C68 run-time environment. In these cases, if the user does not provide a value, then the specified default values will be used.

2.10.1 *extern long _def_priority*

Used to set priority of new jobs. Default is a value of 32.

2.10.2 *extern int os_nerr*

Number of QDOS error messages catered for in '*os_errlist*' table.

2.10.3 *extern char *os_errlist[]*

Table giving text for all the standard QDOS error codes. Use the negation of the QDOS error code (to convert it to a positive number) as an index into this table to get the text for a particular error code.

2.10.4 *extern WINDOWDEF_t _condetails*

This contains the definition details for the initial console window. The *WINDOWDEF_t* type refers to a structure that is defined in sys/qlib.h The default values are equivalent to a C statement of the form:

```
WINDOWDEF_t _condetails = {
   2,    /* border colour (red) */
   1,    /* border width */
   0,    /* paper (black) */
   7,    /* ink (white) */
   464,   /* width (pixel) */
   180,   /* height (pixels) */
   24,    /* x origin */
   26    /* y origin */
};
```

This global variable is used by *the consetup_default()* and *consetup_title()* routines to determine the console details.

2.10.5 *extern char _copyright[]*

This variable is used by the *consetup_title()* routine. It inserts this string at the left side of the menu bar. The default value is a zero length string, but the user can define his own text.

2.10.6 *extern char * _endmsg*

The message that will be used when a program closes down. Default is "Press a key to exit". After displaying the message, the program waits for a keypress. Setting this pointer to *NULL* will mean that the program exits without displaying any message.

2.10.7 *extern timeout_t _endtimeout*

The timeout that will be used when displaying the closedown message and waiting for a response. The default is -1 which means wait forever. Positive values are the number of 1/50 second units to wait.

2.10.8 *extern long _memincr*

Sets the minimum increment in which new memory allocations will be made from the stack. Default value is 4K bytes.

2.10.9 *extern long _memmax*

Sets the maximum memory that a program is allowed to allocate. Default is as much as the program wants.

2.10.10 *extern long _memfree*

Sets the amount of memory that must always be left for QDOS or SMS when trying to allocated additional memory for a program. Default is 20K bytes.

2.10.11 *extern long _mneed*

Sets program initial memory allocation. A negative value can be set which means allocate all the memory except this amount. Default is 8K bytes.

2.10.12 *extern long _oserr*

Used to return QDOS/SMS error codes for some of the QDOS/SMS trap and/or vector calls. It can also be set when an error return is made from a standard C level routine with the *errno* global variable set to the value *EOSERR* (as defined in errno.h).

2.10.13 *extern long _pipesize*

Sets default pipe size.

2.10.14 *extern char _prog_name[]*

Sets default program name. Default is a name of `C-PROG`.

2.10.15 *extern char _Qopen_in[]*

This is the list of special characters that are checked for by the *qopen()* library routine. It should be *NULL* terminated. Its default value is the string "/.\\".

2.10.16 *extern char _Qopen_out[]*

This is the list of what each character that is found in the *_Qopen_in* string should be converted to. It must be at least the same length as *_Qopen_in* or the effect is undefined. Its default value is the string "___".

2.10.17 *extern long _stack*

Sets program stack size. Default is 2Kb bytes.

2.10.18 *extern long _stackmargin*

Sets the default value for the *'stackcheck'* routine to start reporting failures. Default is 256 bytes.

2.10.19 *extern char *_sys_var*

Base of system variables. Set when the program starts-up.

2.10.20 *extern char _version[]*

This is a string used by the *consetup_title()* routine. It is inserted at the right hand end of the menu bar. Default value if this string is not defined explicitly in the users program is a zero length string.

2.11 Global Vectors

The following are global vectors that can be set in user programs to control certain default actions of the C68 run-time environment. If the user does not provide a value, then the specified default values will be used.

N.B. Setting other values that are specified here can have undefined effects and are very likely to cause a system crash.

2.11.1 *extern long (*_cmdchannels)()*

This can be set to *NULL* if the program cannot be passed channels directly from SuperBasic. Default is to include code to allow channels to be accepted from SuperBasic.

2.11.2 *extern void (*_cmdparams)()*

This can be set to NULL if the program does not take any parameters. This will stop code for parsing the command line being included. Default is to include code for parsing the command line.

2.11.3 *extern void (*_cmdwildcard)()*

This can be set to specify the routine to expand wild cards if they are found in the command line. Default is *NULL* which means that wildcards are not expanded. The routine *cmdexpand()* is provided which will simulate the filename expansion that is done by the Unix shell.

2.11.4 *extern void (*_consetup)()*

This contains a pointer to the routine that will be called to initialise the console window on program startup. It will only be called if the console channel was NOT passed on the stack from another program.

The default routine *consetup_default()* merely clears the window and puts a border around it. The routine *consetup_title()* is also provided in the standard library. This will additionally provide a title bar at the top of the window (c.f. the *_copyright* and *_version* global variables).

If this vector is set to *NULL*, then no default initialisation is done.

Alternatively, the user can provide his own alternative routine. See QDOSC68_DOC for more details.

2.11.5 *extern long (*_conread)(UFB_t * uptr, void * buffer, long length)*

This is a pointer to a routine that will handle any input translation for console/screen devices of any special characters during a read. The supplied default routine acts on the following special characters:

CTRL-D Treated as EOF

CTRL-X Treated as "Kill Job"

This vector can be set to *NULL* if console input translation is definitely not required. This will cause the relevant code to be omitted from the program.

This vector can be set to point to an alternative routine if more comprehensive input translation is required. The value returned is the number of characters read into the buffer.

2.11.6 *extern long (*_conwrite)(UFB_t * uptr, void * buffer, long length)*

This is a pointer to a routine that will handle any output translation for console/screen devices of any special characters during a write. The supplied default routine handles the ANSI C specified escape sequences.

This vector can be set to NULL if console output translation is definitely not required. This will cause the relevant code to be omitted from the program.

Alternatively, if more sophisticated output translation is required then a user routine can be substituted. The return values from this routine are treated as follows:

0 an error occurred

+ve output the specified number of characters from the buffer without translation.

-ve the specified number of characters from the buffer required special translation which has been done.

2.11.7 *extern int (*_Open)(const char * name, int mode, ...)*

This is a pointer to the routine that will be used for any *open()*, *fopen()* or *stat()* routines in the program. By default this points to a standard internal library routine that implements the *open()* call. If the special additional actions carried out by the *qopen()* routine are required then this can be invoked by setting the *_Open* vector as follows:

#include <fcntl.h>

*int (*_Open)(const char *, int, ...) = qopen;*

If you wish to write some other variant of the *open()* call, then look at the source of the *qopen()* module for an example of how to go about this.

2.11.8 *extern int (*_readkbd) (chanid_t channel, timeout_t timeout, char *, byte_read);*

This is a pointer to the routine that is used to read the keyboard. Normally it would point to the standard operating system call for reading a byte.

Setting this to another value allows you to write a routine that can intercept keyboard input before it is passed back to the main C program. For an example of such a routine and how it might be used see the *readmove()* routine provided in the QPTR part of the standard C library.

---

3 Change History

- 16 Jun 93 Added descriptions for the new string handling routines *qstrcat()*, *qstrchr()*, *qstrcmp()*, *qstrcpy()*, *qstrlen()*, *qstricmp()*, *qstrncat()*, *qstrncmp()*, *qstrncpy()*, *qstrnicmp()*, *ut_cstr()*
- 10 Jul 93 Description of calls amended to remove the statement that they set the *_oserr* global variable (where this is no longer true).
- 31 Dec 93 Documented the *_copyright'* and *'_version'* global variables.
- 24 Jan 94 Removed all references to the direct QDOS and SMS operating system calls. These are now documented in the LIBQDOS_DOC and LIBSMS_DOC files.
- 03 Sep 94 Added descriptions of the *argfree()*, *argpack()* and *argunpack()* routines.
- 20 Jan 95 Added descriptions of the *qopen()* routine and the associated *'_Open'* vector.
- 10 Feb 95 Documented the *qfork()* family of routines.
- 16 Apr 95 Added descriptions of the more important structures and typedefs that are defined in the sys/qlib.h header file.
- 28 Sep 95 Added description of *_endtimeout* global variable.
- Updated to reflect implementation of Richard Zidlicky's signal handling extension.
- 07 Dec 96 Added description of *strfnd()* routine, amended to be always case independent.

- 16 May 98 Added descriptions for the *_CacheFlush()* and *_ProcessorType()* routines.

Part II
The libQDOS_a Library

This section of the C68 library documentation covers those routines in the C68 standard library that provide access to the QDOS operating system interfaces.

All of the calls in this part of the library map directly onto the QDOS System Calls available to Assembler (machine code) programmers. It is therefore useful to have access to documentation covering the Assembler level interface to QDOS if you want more details on how many of these calls work.

You do not ever need to tell the linker explicitly that you want to include routines defined in this document. These routines are imbedded in the LIBC_A library which is automatically included at the end of the link by the LD linker. You must always, however have the statement

```
#include <qdos.h>
```

in any program or module that makes use of the routines in this library.

It is worth noting that all the calls defined here also work on the SMS family of operating systems. However in that case they traditionally have alternative names. If you wish to find the functions listed and described under their SMS names, then refer to the LIBSMS_DOC file.

4 Reference Material

The reference books listed below were used in preparing material for inclusion in this library:

- "QL Technical Guide" by David Karlin and Tony Tebby
- "QL Advanced User Guide" by Adrian Dickens
- "QDOS Reference Manual" as published by Jochen Merz

4.1 Reference

4.1.1 *int c_extop (chanid_t channel, timeout_t timeout, int (*func), int number_of_params, ...)*

This routine allows a routine to be called to do an extended operation on a QDOS channel. The parameters are passed in a way that is compatible with this routine being written in C (c.f. *sd_extop()/iow_xtop()* for assembler only routines).

The C routine will be called in supervisor mode, with the parameters specified by ... above passed to it on the stack. Each parameter is assumed to be no larger than 4 bytes in size (i.e. no structures are to be passed on the stack). Note also that due to a bug in QDOS, it seems to hang if the routine does not return zero in D0. Therefore, if it is desired to pass an error code back to the application program it must be done indirectly via one of the parameters.

4.1.2 *char *cn_date(char *asciidate, time_t qldate)*

Converts a date from internal QL format into an ASCII string in the format "YYYY mmm dd hh:mm:ss". The *asciidate* parameter must point to a buffer of at least 25 characters in length to hold the return data. The buffer returned is in QL string format - which is a 2 byte length field, followed by the data (NULL terminated for convenience to C programmers). The return value is the address of the start of the text.

Note that if you intend to access the *length* field of the buffer you MUST ensure that it starts on an even address - preferably by defining it using the *QLSTR_DEF* macro to define the buffer.

4.1.3 *char *void cn_day(char *asciiday, time_t qldate)*

Returns the 3 character day of the week given a date in QL internal format. The *asciidate* parameter must point to a buffer of at least 7 characters in length to hold the return data. The buffer returned is in QL string format - which is a 2 byte length field, followed by the data (NULL terminated for convenience to C programmers).

The return value is the address of the start of the text.

Note that if you intend to access the *length* field of the buffer you MUST ensure that it starts on an even address - preferably by defining it using the *QLSTR_DEF* macro to define the buffer.

4.1.4 *void cn_ftod (char * target, char * value)*

Routine to convert a QDOS floating point value into a decimal character ASCII string.

4.1.5 *void cn_itobb (char * target, char * value)*

Routine to convert a byte into a 8 character ASCII string of binary.

4.1.6 *void cn_itobl (char * target, long * value)*

Routine to convert a long integer into a 32 character ASCII string of binary.

4.1.7 *void cn_itobw (char * target, short * value)*

Routine to convert a short integer (word) into a 16 character ASCII string of binary.

4.1.8 *void cn_itod (char * target, short * value)*

Routine to convert a short integer into a decimal ASCII string.

4.1.9 *void cn_itohb (char * target, char * value)*

Routine to convert a byte into a 2 character ASCII hex string.

4.1.10 *void cn_itohl (char * target, long * value)*

Routine to convert a long integer into a 8 character ASCII hex string.

4.1.11 *void cn_itohw (char * target, short * value)*

Routine to convert a short integer (word) into a 4 character ASCII hex string.

4.1.12 *int fs_check(chanid_t channel, timeout_t timeout)*

QDOS routine to check for pending operations on a file. Returns 0 if operations have completed, QDOS error code (typically -1 for Not complete) if they haven't.

4.1.13 *int fs_date(chanid_t chan, timeout_t timeout, int type, long * sr_date)*

type

=0 Access update date of file,

=2 Access backup date.

**sr_date*

=-1 Read requested date (returned from call in *sr_date)

=0 Set requested date to current date.

else Set requested date to date given in *sr_date.

Read/Set update or backup dates. Available on Miracle Systems hard disk, ST/QL sytems and SMS systems. The date set/read is returned in **sr_date*. Returns QDOS error code.

4.1.14 *int fs_flush(chanid_t channel, timeout_t timeout)*

QDOS routine to flush all buffers on a file. Returns QDOS error codes.

4.1.15 *int fs_headr(chanid_t chan, timeout_t timeout, void * buf, short buflen)*

QDOS routine to read a file header. Returns length read on success, QDOS error code (which is negative) on failure.

4.1.16 *int fs_heads(chanid_t chan, timeout_t timeout, void * buf, short buflen)*

QDOS routine to save a file header. Returns length written on success, QDOS error code (which is negative) on failure. You must have opened the file with a mode that allows writing for this call to be successful.

4.1.17 *long fs_load(chanid_t channel, char * buf, unsigned long len)*

Routine to load a complete file. Returns length loaded on success, QDOS error code (which is negative) on error.

4.1.18 *int fs_mdinf(chanid_t chan, timeout_t timeout,char * medname, short * unused_secs, short * goodsecs)*

Routine to get media information. Returns 10 character name of media (N.B. not *NULL* terminated), number of unused_sectors, and number of good sectors. Returns QDOS error code.

4.1.19 *int fs_mkdir(chanid_t channel)*

Make the file specified by the QDOS channel into a directory. Requires support for Level 2 filing system (e.g. Miracle hard Disk, ST/QL or SMS systems). Returns QDOS error code.

4.1.20 *long fs_pos(chanid_t chan, long pos, int mode)*

QDOS equivalent to C *seek()* routine to seek to a point in a file (no timeout as it's always -1). The parameter *'mode'* can have the following values:

0 absolute

1 relative to current position

2 relative to EOF.

Returns new position on success, and QDOS error code (which is negative) on failure.

4.1.21 *long fs_posab(chanid_t chan, timeout_t timeout, unsigned long * pos)*

Routine to seek to an absolute point in a file. The new file position is returned via the *'pos'* parameter. Returns QDOS error code.

4.1.22 *long fs_posre(chanid_t chan, timeout_t timeout, long * pos)*

Routine to seek to a point in a file relative to the current position. The new file position is returned via the *'pos'* parameter. Returns QDOS error code.

4.1.23 *int fs_rename(char * old, char * new)*

Routine to rename a file. Uses C strings. Calls toolkit 2 routine. Returns QDOS error code.

4.1.24 *int fs_save(chanid_t channel, char * buf, unsigned long len)*

Routine to save a complete file to a channel. Returns length saved on success, QDOS error code (which is negative) on failure.

4.1.25 *int fs_trunc(chanid_t channel, timeout_t timeout)*

Routine to truncate a file at the current byte position.

This call may not be available on very basic QL systems (unless Toolkit 2 present) but all other types of system can be expected to support it. Returns QDOS error code.

4.1.26 *int fs_vers(chanid_t channel, timeout_t timeout, long * key)*

Set/Read a file version number. Only available on systems that support version2 filing systems (such as Miracle hard disk, ST/QL and SMS systems). The action is defined as follows:

**key* =

-1 Return version number in **key*.

0 Keep old version number when file closed (return it on **key*)

+ve and < 65536 Set version number to given number.

Returns QDOS error code.

4.1.27 *int fs_xinf(chanid_t channel, timeout_t timeout, struct ext_mdinf * fsinf)*

Get extended file system info. Only available on systems that support version 2 filing system (such as Miracle hard disk, ST/QL and SMS systems). Requested data is returned in *struct ext_mdinf* (defined in qdos.h) on success.

Returns QDOS error code

```
typedef struct ext_mdinf {
   union {
      char m_size[22];
      QLSTR_t m_name;
   } xm_name;
   union {
      char m_dsize[6];
      QLSTR_t m_dname;
   } xm_dname;
   unsigned char xm_dnum;
   char xm_rdonly;
   unsigned short xm_alloc;
   unsigned long xm_total;
   unsigned long xm_free;
   unsigned long xm_hdrlen;
   char xm_spare[20];
   char xm_spare2[36];
} ext_mdinf_t;
```

4.1.28 *int io_close (chanid_t channel)*

Closes a channel. Returns QDOS error code.

4.1.29 *int io_delete (char *name)*

Routine to delete a file. Uses C strings. Returns QDOS error code.

4.1.30 *int io_edlin (chanid_t channel, timeout_t timeout, char **cptr, int bufsize, int current_offset, int *current_linelen);*

Routine to do edited line read call. Returns QDOS error code.

4.1.31 *int io_fbyte (chanid_t channel, timeout_t timeout, char *char_pointer)*

Routine to read 1 byte. Returns QDOS error code.

4.1.32 *int io_fdate (chanid_t chan, timeout_t timeout, int type, unsigned long *sr_date)*

Obsolete form - should now use *fs_date()* (or even better, the SMS name *iof_date()*)instead.

4.1.33 *int io_fline (chanid_t channel, timeout_t timeout, void *buf, short length)*

Routine to read a linefeed terminated string of bytes. Returns length read on success, QDOS error code (which is negative) on failure.

4.1.34 *int io_format (char *device, short *totsecs, short *goodsecs)*

Routine to format a medium, uses C string name. Returns total and good sector count. Returns QDOS error code.

4.1.35 *int io_fstrg (chanid_t channel, timeout_t timeout, void *buf, short length)*

Routine to fetch a string of bytes. Returns length read on success, or QDOS error code (which is negative) on failure. The amount read can be less than the amount requested. This would normally caused by an end-of-file or timeout condition occurring during the read.

4.1.36 *int io_fvers (chanid_t channel, timeout_t timeout, long *key)*

Obsolete form - should now use *fs_vers()* instead.

4.1.37 *int io_fxinf (chanid_t channel, timeout_t timeout, struct ext_mdinf *fsinf)*

Obsolete form - should now use *fs_xinf()* instead.

4.1.38 *int io_mkdir (chanid_t channel)*

Obsolete form - should now use *fs_mkdir()* instead.

4.1.39 *chanid_t io_open (char *name, int mode)*

Routine to open a file. Uses C strings.
The modes are defined in sys/qlib_h as follows:

```
#define OLD_EXCL        0
#define OLD_SHARE       1
#define NEW_EXCL        2
#define NEW_OVER        3
#define DIROPEN         4
```

Returns channel id or QDOS error code (which is negative).

4.1.40 *int io_pend (chanid_t chan, timeout_t timeout)*

Routine to test for any pending input on a channel, returns 0 if data is to be read, else -1 (not complete).

4.1.41 *int io_qeof (char * queue_pointer)*

Insert an EOF (end-of-file) marker into a queue.
Returns QDOS error code (if any).

4.1.42 *int io_qin (char * queue_pointer, int byte_to_insert)*

QDOS routine to insert a byte in a queue. Returns the QDOS error code (if any).

4.1.43 *int io_qout (char * queue_pointer, char * next_byte)*

Remove a byte from a queue. Returns the QDOS error code (if any).

4.1.44 *void io_qset (char * queue_pointer, long queue_length)*

Routine to set up a queue.

4.1.45 *int io_qtest (char * queue_pointer, char * next_byte, long * free_space)*

Test the status of a queue. The variables whose addresses are passed as parameters are updated to the free space in the queue, and (if there is data in the queue) the value of the next byte is returned (although the byte is not removed from the queue).
The QDOS error code is returned.

4.1.46 *int io_rename (char *old, char *new)*

Obsolete form - should now use *fs_rename()* instead.

4.1.47 *int io_sbyte (long chan, timeout_t timeout, unsigned char ch)*

Routine to output char *ch* to channel.

Returns QDOS error code.

4.1.48 *int io_serio (chanid_t channel_id, timeout_t timeout, int routine_number, long * D1, long * D2, char ** A1, char * routine_array[4])*

General serial IO handling routine.

This routine is used when the *io_serq()* routine is not sufficient.

The values passed as the parameters 'D1', 'D2' and 'A1' are pointers to the values to be put into the registers D1, D2 and A1 respectively. These values may be changed by this routine. The *routine_array* is an array of at least 4 elements, the first three of which contain the addresses of the routines for testing pending input, fetching a byte and sending a byte. The fourth element will be used as workspace, and thus corrupted by this call.

4.1.49 *int io_serq (chanid_t channel_id, timetout_t timeout, int routine_number, long * D1, long * D2, char ** A1)*

Serial IO Direct Queue handling routine.

The values passed as the parameters *D1*, *D2* and *A1* are pointers to the values to be put into the registers D1, D2 and A1 respectively. These values may be changed by this routine.

4.1.50 *int io_sstrg (chanid_t channel, timeout_t timeout, void *buf, short length)*

Routine to write a string of bytes. Returns length written on success, and a QDOS error code (which is negative) on failure.

The amount written can be less than the amount requested. This would normally be caused by a timeout condition occuring during the write.

4.1.51 *int io_trunc (chanid_t channel, timeout_t timeout)*

Obsolete form - should now use *fs_trunc()* instead.

4.1.52 *int iop_outl (chanid_t channel, timeout_t timeout, short xShad, short yShad, short keep, void * winDef)*

This is the call that sets the outline window for a Pointer Environment. It is included in this library as it is the one call that need to be issued to make a program that is not otherwise aware of the pointer environment function correctly in that environment.

For more details refer to the LIBQPTR_DOC file provided as part of the QPTR library.

Note that the default console initialisation routines supplied with C68 will automatically issue a call to set the window outline to the size as defined in the *_condetails* global variable (for more information see LIBC68_DOC).

xShad, *yShad*: Shadow widths

keep: keep contents of window

winDef: Pointer to QLRECT structure that holds the window limits

---

4.1.53 *char * mm_alchp (long size, long *sizegot)*

Routine to allocate memory from common heap. It is passed the requested size and returns address of area allocated (or a QDOS error code on failure).

The area will always be allocated with the current job as the owner. If you are not interested in the true size obtained, then set *sizegot* to NULL. Otherwise set it to the address of a variable that will be set to contain the actual size obtained (Note that even if the call succeeds this may not be the same as the size requested, as the amount requested is often rounded up by QDOS.

It is recommended that you use *mt_alchp()* in preference to *mm_alchp()* unless you are sure you know what you are doing.

WARNING The size requested must allow for the QDOS heap header, and the address returned is the start of the area allocated - not the useable area. This is in contrast to the *mt_alchp()* call for which the user does not have to worry about the QDOS heap header.

4.1.54 *char *mm_alloc (char **ptr, long *len)*

QDOS routine to allocate a user area from an allocated area of common heap.

ptr is a pointer to a pointer to free space, len is the length requested to put in the user heap, and returns as the length actually allocated.

Returns the address of the area allocated on success, and the QDOS error code on failure.

4.1.55 *void mm_lnkfr (char *area, char **ptr, long len)*

QDOS routine to link an area back into a user heap area. Given area to link in, pointer to pointer to free space, and length to link in. This call is also used to set up a user heap.

4.1.56 *void mm_rechp (char *area)*

QDOS routine to free an area of common heap previously allocated via *mm_alchp()*.

Returns no errors. It always succeeds unless the parameter points to an invalid address, in which case the machine nearly always crashes!

4.1.57 *void mt_aclck (long ql_time)*

Routine to adjust the clock by *ql_time* seconds.

4.1.58 *int mt_activ (long jobid, unsigned char priority, timeout_t timeout)*

Routine to start a activate a job with a given priority. There are two valid values for the timeout, 0 and -1. Execution of the current job will continue if the timeout is set to zero, and the QDOS error code for this call returned. If the timeout is -1 then the current job is suspended until the activated Job has finished. This call will then return the error code from that Job.

4.1.59 *char * mt_alchp (long size, long * sizegot, long jobid)*

Routine to allocate memory from common heap. Is passed requested size, plus job id which is to own the heap. Returns address of area allocated, or a QDOS error code on failure.

Note that even if the call succeeds the amount of memory actually allocated will not be the same as the size requested, as the amount requested is rounded up to the nearest 16 bytes and then the length of the common heap header is added on to it. If you are not interested in the true size obtained, then set *'sizegot'* to NULL. Otherwise set it to the address of a variable that will be set to contain the actual size obtained.

4.1.60 *void * mt_alloc (char **ptr, long *len)*

Routine to allocate a user area from an allocated area of common heap. 'ptr' is a pointer to a pointer to free space, len is the length requested to put in the user heap, and returns as the length actually allocated. Returns the address of the area allocated on success, and the QDOS error code on failure.

4.1.61 *void * mt_alres (long size)*

Routine to allocate memory from resident procedure area. Returns address of area allocated, or a QDOS error code on failure. On standard QDOS systems this call will always fail if called while any program except SuperBasic is executing. Most later systems and those fitted with Minerva ROMs do not suffer from this limitation.

4.1.62 *void mt_baud (int rate)*

Routine to set the baud rate for both serial ports.

4.1.63 *jobid_t mt_cjob (long codespace, long dataspace, char *start_address, jobid_t owner, char **job_address)*

Routine to create another job in the transient program area, given the size of the new job's code, data. the start address of the new job, and its owner. Returns either positive job id of new job, or QDOS error code. Also returns address of newly created job in last parameter.

4.1.64 *void mt_dmode (short *s_mode, short *d_type)*

Routine to set/read display mode.

*s_mode = 4 for mode 4, 8 for mode 8, -1 for read

*d_type = 0 for monitor mode, 1 for TV mode, -1 for read

Notes:

1. Other values are available for use in these parameters on Minerva sytems - refer to the Minerva documentation for details
2. There is a bug in some QL roms that corrupts the return d_type when it is read.

4.1.65 *long mt_free ()*

Routine to find largest contiguous area available for loading a program. This is normally also a good indicator of the total free memory in the machine.

4.1.66 *int mt_frjob (jobid_t jobid, int error_code)*

Routine to force remove a job, giving an error code for it to return.

Returns QDOS error code (if we are not removing the current job).

4.1.67 *jobid_t mt_inf (char **system_variables, long *version_code)*

Routine to get the address of the system variables and the current operating system version code. The version code is actually returned as 4 bytes in the form x.xx. Returns job id of current job.

4.1.68 *int mt_ipcom (char *param_list)*

Routine to send a command to the 8049 second processor. Uses INTEL byte format (low byte first).

Returns value returned by 8049.

4.1.69 *int mt_jinf (jobid_t *jobid, jobid_t *topjob, long *job_priority, char **job_address)*

Get information on a job within a job tree.

Passed the jobid you want information on and the current top of the job tree you are looking at (with the first call set `*topjob = *jobid`). It is designed to be called repeatedly without changing `jobid` and `topjob` until `*jobid == 0`.

Returns:

0 OK with '`job_address`' contains address of job '`jobp`' contains job priority in least significant byte, and if the job is suspended the most significant byte is negative. '`jobid`' and '`topjob`' are changed to those of the next job in the tree.

-ve QDOS error code.

4.1.70 *void mt_lnkfr (char *area, char **ptr, long len)*

Routine to link an area back into a user heap area.

Given area to link in, pointer to pointer to free space, and length to link in. This call is also used to set up a user heap.

4.1.71 *void mt_lxint (QL_LINK_t * lnk)*

Link in external interrupt handler

*void mt_rxint (QL_LINK_t * lnk)*

Unlink external interrupt handler

void mt_lpoll(QL_LINK_t * lnk)

Link in polled task handler

void mt_rpoll(QL_LINK_t * lnk)

Unlink polled task handler

void mt_lschd(QL_LINK_t * lnk)

Link in scheduler list handler

void mt_rschd(QL_LINK_t * lnk)

Unlink scheduler list handler

void mt_liod(QLD_LINK_t * lnk)

Link in simple I/O device handler

void mt_riod(QLD_LINK_t * lnk) Unlink simple I/O device handler

void mt_ldd(QLDDEV_LINK_t * lnk)

Link in directory I/O device handler

void mt_rdd(QLDDEV_LINK_t * lnk)

Unlink directory I/O device handler
The QL_LINK_t, QLD_LINK_t and QLDDEV_LINK_t structures are defined in sys/qlib.h

```
typedef struct QL_LINK {
   struct QL_LINK *l_next;
   void (*l_rtn)_P_((void));
} QL_LINK_t;

typedef struct QLD_LINK {
   struct QLD_LINK *ld_next;
   long (*ld_io)_P_((void));
   long (*ld_open)_P_((void));
   long (*ld_close)_P_((void));
} QLD_LINK_t;

typedef struct QLDDEV_LINK {
   struct QLDDEV_LINK *ldd_next;
   long (*ldd_io)_P_((void));
   long (*ldd_open)_P_((void));
   long (*ldd_close)_P_((void));
   long (*ldd_slave)_P_((void));
   long (*ldd_rename)_P_((void));
   long (*ldd_trunc)_P_((void));
   long (*ldd_format)_P_((void));
   long ldd_plen; QLSTR_t ldd_dname;
} QLDDEV_LINK_t;
```

4.1.72 int mt_prior(long jobid, int new_priority)

Routine to set the priority of a job. Sets current job's priority if *jobid* = -1.
Returns old priority of this job or a QDOS error code.

4.1.73 long mt_rclck()

Routine to read clock.
Returns time in seconds from Jan 1 1961.

4.1.74 void mt_rechp(char *area)

Routine to free an area of common heap previously allocated. Returns no errors.
WARNING The way this call is implemented in QDOS and SMS is such that it either succeds or crashes the sytem if given an invalid area address. Do not therefore try and call it twice for the same area or call it for an area not allocated via the *mt_alchp()* call.

4.1.75 JOBHEADER_t * mt_reljb(jobId_t jobid)

Routine to release a suspended job, sets *_oserr*, returns address of job header (the JOBHEADER_t structure is defined in sys/qlib.h) or QDOS error code.

4.1.76 int mt_reres(char *area)

Routine to free an area of the resident procedure area previously allocated. Returns QDOS error code. On QDOS systems, will always fail if called when aby program except SuperBasic is running.

4.1.77 int mt_rjob(jobid_t jobid, int error_code)

Routine to remove a suspended job, giving an error code for it to return. Returns QDOS error code.

4.1.78 void mt_sclck(long ql_time)

Routine to set the clock.

4.1.79 int mt_shrink(char *block, long newsize)

Routine to shrink an area of QDOS allocated common heap.

This is used when you have grabbed an area of common heap and realise you do not need all of it. Rather than freeing all of it then re-allocating (by which time another job may have grabbed the space) you can use this call to release the top part of it that you do not need. *newsize* **MUST** be less than the size originally allocated or this call can fail badly, after the call the allocated block will be only *newsize* bytes long (not including common heap header), the higher portion of it will have been given back to QDOS and placed on the free list.

Returns a QDOS error code.

4.1.80 int mt_susjb(jobid_t jobid, int number, char *zero)

Routine to suspend a job for a number of 50Hz (or 60Hz if an American QL) clock ticks. *char *zero* is an address of a byte to set to zero on release of the job if required. If this is not required pass NULL in place of 'char *zero'. If number = -1 then the job is suspended indefinitely.

Returns a QDOS error code.

4.1.81 int mt_trans (char * trans_table, char * msg_table)

Routine to set the translate table and message table. This routine will not work on QL systems with ROMS that are of version JS or earlier.

Returns QDOS error code.

4.1.82 int mt_trapv(QLVECTABLE_t * table, long jobid)

Routine to change the exception vector table for a particular job. The *QLVECTABLE_t* structure is defined in sys/qlib.h

```
typedef struct QLVECTABLE {
   long (*qv_adderr)_P_((void));
   long (*qv_illegal)_P_((void));
   long (*qv_divzero)_P_((void));
   long (*qv_CHK)_P_((void));
   long (*qv_TRAPV)_P_((void));
   long (*qv_priviol)_P_((void));
   long (*qv_tracexpt)_P_((void));
   long (*qv_intlev7)_P_((void));
   long (*qv_5trap)_P_((void));
   long (*qv_6trap)_P_((void));
   long (*qv_7trap)_P_((void));
   long (*qv_8trap)_P_((void));
   long (*qv_9trap)_P_((void));
   long (*qv_10trap)_P_((void));
```

```
   long (*qv_11trap)_P_((void));
   long (*qv_12trap)_P_((void));
   long (*qv_13trap)_P_((void));
   long (*qv_14trap)_P_((void));
   long (*qv_15trap)_P_((void));
} QLVECTABLE_t;
```

Returns QDOS error code.

4.1.83 int sd_arc(chanid_t channel, timeout_t timeout, double x_start, double y_start, double x_end, double y_end, double angle)

Routine to draw an arc using graphics coordinates. sd_arc uses C double precision floating point coordinates (cf. sd_iarc).
Returns QDOS error code.

4.1.84 int sd_bordr(chanid_t channel, timeout_t timeout, unsigned char colour, short width)

Routine to redifine a window border with new colour and width.
Returns QDOS error codes.

4.1.85 int sd_chenq(chanid_t channel, timeout_t, QLRECT_t *rect)

Routine to read a window size in characters. On success *'rect'* is set to details of answer.
Returns QDOS error code.

4.1.86 int sd_clear(chanid_t channel, timeout_t timeout)

Routine to clear entire window.
Returns QDOS error code

4.1.87 int sd_clrbt(chanid_t channel, timeout_t timeout)

Routine to clear area of window below cursor line.
Returns QDOS error code.

4.1.88 int sd_clrln(chanid_t channel, timeout_t timeout)

Routine to clear all of cursor line. Returns QDOS error code.

4.1.89 int sd_clrrt(chanid_t channel, timeout_t timeout)

Routine to clear cursor line, to right of cursor position (including cursor).
Returns QDOS error code.

4.1.90 int sd_clrtp(chanid_t channel, timeout_t timeout)

Routine to clear area of window above cursor line.
Returns QDOS error code.

4.1.91 int sd_cure(chanid_t chan, timeout_t timeout)

Routine to enables cursor on screen channel.
Returns QDOS error code.

4.1.92 int sd_curs(chanid_t chan, timeout_t timeout)

Routine to suppress cursor on screen channel.
Returns QDOS error code.

4.1.93 int sd_donl(chanid_t channel, timeout_t timeout)

Routine to flush any pending newlines on a window channel.
Returns QDOS error code.

4.1.94 int sd_elipse(chanid_t channel, timeout_t timeout, double x_centre, double y_centre, double eccentricity, double radius, double angle_of_rotation)

Routine to draw a circle or ellipse using graphics coordinates. sd_elipse uses C double precision floating point coordinates (cf. sd_ielipse).
Returns QDOS error code.

4.1.95 int sd_extop(chanid_t channel,timeout_t timeout,int (*rtn)(), long paramd1, long paramd2, void *parama1)

Routine to do extended operation on screen channel. Passed address of routine to call and parameters for d1, d2 and a1. Returns QDOS error code. See also c_extop().

NOTE. Due to a bug in QDOS, it appears that D0 must always be zero on exiting the *rtn()* function. Any error code therefore needs to be passed back indirectly via one of the other parameters.

4.1.96 int sd_fill(chanid_t channel, timeout_t timeout, colour_t colour, QLRECT_t * rect)

Routine to plot a rectangular block of a certain colour. Can be used to draw very fast horizontal and vertical lines.

Returns QDOS error code.

4.1.97 int sd_flood(chanid_t channel, timeout_t timeout, int onoff)

Routine to set flood fill mode on or off.

Returns QDOS error code.

4.1.98 int sd_fount(chanid_t channel, timeout_t timeout, char *font1, char *font2)

Routine to set normal and alternative character font in a window. Passed pointers to two font definitions (format as described in QDOS manuals).

Returns QDOS error code.

4.1.99 int sd_gcur(chanid_t channel, timeout_t timeout, double vert_offset, double horiz_offset, double x_pos, double y_pos)

Routine to set the graphics text cursor. sd_gcur uses C double precision floating point coordinates (cf. sd_igcur).

Returns QDOS error code.

4.1.100 int sd_iarc(chanid_t channel, timeout_t timeout, double x_start, double y_start, double x_end, double y_end, double angle)

Routines to draw an arc using graphics coordinates. sd_iarc takes integer coordinates (c.f. sd_arc)

Returns QDOS error code.

4.1.101 int sd_ielipse(chanid_t channel, timeout_t timeout, int x_centre, int y_centre, int eccentricity, int radius, int angle_of_rotation)

Routine to draw a circle or ellipse using graphics coordinates. sd_ielipse uses integer coordinates (cf. sd_elipse).
Returns QDOS error code.

4.1.102 int sd_igcur(chanid_t channel, timeout_t timeout, int vert_offset, int horiz_offset, int x_pos, int y_pos)

Routine to set the graphics text cursor. sd_igcur uses integer coordinates (cf. sd_gcur).
Returns QDOS error code.

4.1.103 int sd_iline(chanid_t channel, timeout_t timeout, int x_start,int y_start, int x_end, int y_end)

Routine to draw a line with graphics coordinates. sd_iline takes integer coordinates (cf. sd_line).
Returns QDOS error code.

4.1.104 int sd_ipoint(chanid_t channel, timeout_t timeout, int x, int y)

Routine to plot a point using graphics coordinates. sd_ipoint takes integer coordinates (cf. sd_point).
Returns QDOS error code.

4.1.105 int sd_iscale(chanid_t channel, timeout_t timeout, int scale, int x_origin, int y_origin)

Routine to change a windows graphics origin and scale. sd_iscale uses integer coordinates (cf. sd_scale).
Returns QDOS error code.

4.1.106 int sd_line(chanid_t channel, timeout_t timeout, double x_start, double y_start, double x_end, double y_end)

Routine to draw a line with graphics coordinates. sd_line uses C double precision floating point coordinates (cf. sd_iline).
Returns QDOS error code.

4.1.107 int sd_ncol(chanid_t channel, timeout_t timeout)

Routine to move cursor right one column.
Returns QDOS error code.

4.1.108 int sd_nl(chanid_t channel, timeout_t timeout)

Routine to move the cursor to start of next line.
Returns QDOS error code.

4.1.109 int sd_nrow(chanid_t channel, timeout_t timeout)

Routine to move cursor down one row.
Returns QDOS error code.

4.1.110 int sd_pan(chanid_t channel, timeout_t timeout, int ampix)

Routine to pan window left or right. *ampix* < 0 means pan left, *ampix* > 0 means pan right.
Returns QDOS error code.

4.1.111 int sd_panln(chanid_t channel,timeout_t timeout, int ampix)

Routine to pan cursor line left or right. *ampix* < 0 means pan left, *ampix* > 0 means pan right.
Returns QDOS error code.

4.1.112 int sd_panrt(chanid_t channel,timeout_t timeout, int ampix)

Routine to pan right of cursor line left or right (includes character at cursor position). *ampix* < 0 means pan left, *ampix* $>$ 0 means pan right.
Returns QDOS errors code.

4.1.113 int sd_pcol(chanid_t channel, timeout_t timeout)

Routine to move cursor left one column.
Returns QDOS error code.

4.1.114 int sd_pixp(chanid_t channel, timeout_t timeout, short x_pos, short y_pos)

Routine to reposition the cursor to an x, y pixel position in a window.
Returns QDOS error code.

4.1.115 int sd_point(chanid_t channel, timeout_t timeout, double x, double y)

Routine to plot a point using graphics coordinates. *sd_point* takes C double precision floating point coordinates (cf. sd_ipoint).
Returns QDOS error code.

4.1.116 int sd_pos(chanid_t channel, timeout_t timeout, short x_pos, short y_pos)

Routine to reposition the cursor to an x, y character position in a window.
Returns QDOS error code.

4.1.117 int sd_prow(chanid_t channel, timeout_t timeout)

Routine to move cursor up one row.
Returns QDOS error code.

4.1.118 int sd_pxenq(chanid_t channel, timeout_t timeout, QLRECT_t * rect)

Routine to read a window size in pixels. Returns size in the QLRECT_t structure (defined in sys/qlib.h).
Returns QDOS error code.

4.1.119 int sd_recol(chanid_t channel, timeout_t timeout, char *colourlist)

Routine to recolour a window. Done in software and very slow. *colourlist* points to eight characters containing new colours for eight possible QL colours.
Returns QDOS error code.

4.1.120 int sd_scale(chanid_t channel, timeout_t timeout, double scale,double x_origin, double y_origin)

Routine to change a window's graphics origin and scale. sd_scale uses C double precision floating point coordinates (cf. sd_iscale).
Returns QDOS error code.

4.1.121 int sd_scrbt(chanid_t channel,timeout_t timeout, int ampix)

Routine to scroll window below cursor line up or down. *ampix* < 0 means scroll down, *ampix* > 0 means scroll up.
Returns QDOS error code.

4.1.122 int sd_scrol(chanid_t channel,timeout_t timeout, int ampix)

Routine to scroll entire window up or down. *ampix* < 0 means scroll down, *ampix* > 0 means scroll up.
Returns QDOS error code.

4.1.123 int sd_scrtp(chanid_t channel,timeout_t timeout, int ampix)

Routine to scroll window above cursor line up or down. *ampix* < 0 means scroll down, *ampix* > 0 means scroll up.
Returns QDOS error code.

4.1.124 int sd_setfl(long chan, timeout_t timeout, int onoff)

Routine to set flash mode on or off (only works in 8 colour mode).
Returns QDOS error code.

4.1.125 int sd_setin(long chan, timeout_t timeout, int colour)

Routine to set ink colour. Colour value (0-7) dependent on mode.
Returns QDOS error code.

4.1.126 int sd_setmd(chanid_t channel, timeout_t timeout, int mode)

Routine to set type of drawing mode (DM_XOR, DM_OVER, DM_OR).
Returns QDOS error code.

4.1.127 int sd_setpa(long chan, timeout_t timeout, int colour)

Routine to set paper colour. Colour value (0-7) dependent on mode.
Returns QDOS errors code.
Colours defined in qdos.h

4.1.128 int sd_setst(long chan, timeout_t timeout, int colour)

Routine to set strip colour. Colour value (0-7) dependent on mode.
Returns QDOS error code.

4.1.129 int sd_setsz(chanid_t channel, timeout_t timeout, short c_width, short c_height)

Routine to set character width and height in a window.

Possible widths are:

0	6 pixels wide
1	8 pixels wide,
2	12 pixels wide
3	16 pixels wide

Possible heights are:

0	10 pixels high
1	20 pixels high

Returns QDOS error code.

4.1.130 int sd_setul(chanid_t chan, timeout_t timeout, int onoff)

Routine to set underline mode for characters on or off.

Returns QDOS error code.

4.1.131 int sd_tab(chanid_t channel, timeout_t timeout, int pos)

Routine to move to a column position (pos) on a line.

Returns QDOS error code.

4.1.132 int sd_wdef(chanid_t channel, timeout_t timeout, colour_t b_colour, short b_width, QLRECT_t *rect)

Routine to redefine the position and shape of a window. The old window contents are not moved or modified, but the cursor is positioned at the top left hand corner of the new window. The values for border colour and border width are passed explicitly as parametsr. The new position and size for the window are passed as a pointer to a QLRECT_t structure whose members define the origin, height and width.

Returns QDOS error code.

4.1.133 int sms_fthg (char * thing_name, jobid_t jobid, long * d2, long d3, char * a1, char **a2)

Free the named *'thing'*. Available as standard with SMS systems, and on QDOS compatible systems with THING support code loaded.

Returns the QDOS error code.

The parameters d2, d3, a1 and a2 are used to pass extra parameters as defined in the definition of the 'thing' that is being freed.

Note also that the d2 and a2 parameters are pointers to these values as new values can be passed back from the 'thing' being freed. The d3 and a1 parameters are not changed, so pointers are not used for these parameters.

4.1.134 int sms_lthg (THING_LINKAGE * thing_linkage)

Routine to link in a new Thing. Available as standard with SMS, and on QDOS compatible systems with THING support code loaded. The structure THING_LINKAGE is defined if you include the qdos.h or sms.h header files.

```
typedef struct _thing_linkage {
   struct _thing_linkage * th_nxtth;
   long th_usage;
   char * th_frfre;
   char * th_frzap;
   char * th_thing;
   char * th_use;
   char * th_free;
   char * th_ffree;
   char * th_remov;
   char th_nshar;
   char th_check;
   long th_verid;
   short th_name;
   char th_name_text[1];
};
```

4.1.135 int sms_nthg (char * thing_name, THING_LINKAGE **next_thing)

Routine to find next Thing. Available as standard with SMS, and on QDOS compatible systems with THING support code loaded. The 'thing_name' parameter is a C style NULL terminated string. The 'next_thing' parameter is used to return the Thing Linkage block for the next Thing, or 0 if no further Thing exists. The THING_LINKAGE structure is defined in the sms.h header file.

Returns SMS error code.

4.1.136 int sms_nthu (char *name, THING_LINKAGE ** thing_linkage, jobid_t * owner_job)

SMS routine to get the owner of a job, and the next linkage block. If the pointer pointed to by thing_linkage is 0, then this the value returned in 'owner_job' is undefined, and this routine functions like the sms_nthg() routine. defined in sms.h

---

4.1.137 int sms_rthg (char * thing_name)

SMS routine to remove a Thing if it is not in use. The 'thing_name' parameter is a C style (NULL terminated) string. defined in sms.h

---

4.1.138 char * sms_uthg (char * thing_name, jobid_t job_id, timeout_t timeout, long *d2, char *a2, long *version, THING_LINKAGE **linkage)

SMS routine to use a Thing. The name is passed in C (NULL terminated) format. The version is returned in the 'version' parameter. The additional values passed/returned in the 'd2' and passed in the 'a2' parameters are dependent upon the definition of the THING being used. The 'linkage' parameter is used to get back the Thing linkage address on a successful call. If an error occurs, then the error code (which is negative) is returned.

If successful, the address of the Thing is returned, and a pointer to its linkage in the 'linkage' parameter.

The THING_LINKAGE structure is defined in the sms.h header file. Defined in sms.h

---

4.1.139 int sms_zthg (char * thing_name)

Zap a thing. The name is supplied in C (NULL terminated) format. Returns SMS Error code. Defined in sms.h

---

4.1.140 chanid_t ut_con(WINDOWDEF_t * wdef)

Simplified routine to open a console window The WINDOWDEF_t structure is defined in sys/qlib.h.

Returns QDOS channel id on success, and QDOS error code (which is negative) on failure.

4.1.141 int ut_cstr (const QLSTR_t * string1, const QLSTR_t * string2, int mode)

Compare two QDOS strings. The QLSTR_t structure is defined in sys/qlib_h. The type of comparison is determined by mode as follows:

0 Compare on a character by character basis. Case is significant

1 As type 0, but ignore case

2 Embedded numbers are converted to binary before comparison. Text characters are case significant.

3 As type 2, but case is ignored.

The order of comparison uses the QDOS defined collating sequence (which is not the same as the ASCII values of the characters). The value returned is 0 if the strings match, -1 if 'string1' is less than 'string2', and +1 if 'string1' is greater than 'string2'.

4.1.142 void ut_err(int qdoserror, chanid_t channel)

Write the message corresponding to the error code to the specified channel.

4.1.143 void ut_err0 (int qdoserror)

Write the message corresponding to the QDOS error code to channel 0.

4.1.144 void ut_link (char *previous_item, char * nextitem)

Link an item into a linked list.

4.1.145 int ut_mint(chanid_t channel, int value)

Convert a value to ASCII and send it to the specified channel.
Returns QDOS error code (if any).

4.1.146 int ut_mtext(chanid_t, QLSTR * message)

Send a message to a specified channel. Returns QDOS error code (if any).

4.1.147 chanid_t ut_scr (WINDOWDEF_t * windef)

Simplified routine to open a screen window. The WINDOWDEF_t structure is defined in sys/qlib.h.
Returns channel on success, QDOS errror code (which is negative) on failure.

4.1.148 void ut_unlnk (char *previous_item, char * old_item)

Unlink an item from a linked list.

4.1.149 chanid_t ut_window (char *name, char *details)

Simplified routine to open a window. the 'name' parameter is a C type string that specifies the type and dimensions. The details parameter specifies the border details and the paper/ink colours. Returns the QDOS channel id on success and a QDOS error code (which is negative) on failure.

4.2 MANIFEST CONSTANTS

There following manifest constants are defined in QODS.H for the error codes returned by QDOS.

ERR_OK NO error occured

ERR_BL Bad line in BASIC

ERR_BN Bad device name

ERR_BO Buffer overflow

ERR_BP Bad parameter

ERR_DF Drive full

ERR_EF End of file

ERR_EX File already exists

ERR_FE File error

ERR_FF Format failed

ERR_IU File or device in use

ERR_NC Operation not complete

ERR_NF File or device not found

ERR_NI Not implemented

ERR_NJ Not a valid job

ERR_NO Channel not open

ERR_OM Out of memory

ERR_OR Out of range

ERR_OV Arithmetic overflow

ERR_RO Read only

ERR_RW Read or Write Failed

ERR_TE Transmission error

ERR_XP Error in expression

4.3 CHANGE HISTORY

- 20 Jun 93 Added descriptions for the Queue Handling routines *io_qeof(), io_qin(), io_qout(), io_qset(), io_qtest(), io_serq(), io_serio().*
- 10 Jul 93 Description of the majority of the trap calls amended to remove the statement that they set the *_oserr* global variable (where this is no longer true).
- 08 Sep 93 Added *c_extop()* call (based on a contribution by PROGS of Belgium).
- 31 Dec 93 Documented the *iop_outl()* call.
- 24 Jan 94 Reworked this document to only include the direct calls to QDOS. Direct calls under SMS names are now documented in LIBSMS_DOC, and all more generic alls on LIBC68_DOC. Added the names of the standard QDOS error codes as manifest constants.
- 10 Jun 94 The cross-reference list of the routines by function removed from this document. All such lists are now consolidated into the LIBINDEX_DOC file.

Part III
The LibQPTR Library

5 Introduction

The libqptr library is designed to allow you to write programs that exploit the Pointer Environment. The Pointer Environment is built into SMS2 systems, but need to be explicitly loaded for systems running standard QDOS or SMSQ.

You should bear in mind that the Pointer Environment is very specific to the QDOS, SMSQ and SMS2 family of operating systems. If you use these facilities it will not be easy (or sometime not even possible) to port such programs to other operating environments. You should bear this fact in mind when you decide to use the routines in the libqptr library.

You do not ever need to tell the linker explicitly that you want to include routines from the libqptr library. The routines defined as being in the libqptr library are embedded in the LIBC_A library which is automatically included at the end of the link by the LD linker. You should always, however have the statement

```
#include <qptr.h>
```

in any program or module that makes use of the routines defined as being in this library. If you do not you will get error messages from the linker stating that the LIBQPTR routines are undefined.

5.1 typedef'ed Structures

To help you to produce readable code, all the structures used in the LIBQPTR_A routines have been typedef'ed. The names of the typedef are always constructed by adding '_t' to the structure name.

This means that instead of writing something like *struct WM_wdef* you can use *WM_wdef_t* which is slightly more readable, and also helps the compiler do stricter type checking.

5.2 Reference Material

The reference books listed below were used in preparing the material for inclusion in this library:

"QPTR Pointer Environment" manual sold by Jochen Merz

LIBRARY CONTENTS

The routines in this library are split into the following sections:

- Button Frame Utility Functions, - Window Manager Utility Functions (C68 compatible), - Window Manager Wrappers and Internal Routines (not callable from C68), - Pointer Interface Trap Wrappers.

5.3 Button Frame Utility Functions

5.3.1 int bt_frame (chanid_t, WM_swdef_t *sw)

Using size and attributes from the sub-window definition sw, bt_frame allocates a space in the button frame for the channel, sets the origin in sw, sets border/paper/strip and, if specified, clears the window.

```
typedef struct WM_swdef {
   short xsize;
   short ysize;
   short xorg;
   short yorg;
   short flag;
   short borw;
   short borc;
   short papr;
   WM_sprite_t * sprite;
} WM_swdef_t;
```

5.3.2 int bt_free (void)

Frees the button frame allocation.

5.3.3 int bt_prpos (WM_wwork_t *)

As *wm_prpos* but positions primary window in the button frame. If successful, the shadow width is set to zero.

Returns QDOS/SMS error code.

```
typedef struct WM_wwork {
   WM_wstat_t * wstat;
   WM_wscale_t * wscale;
   long chid;
   WM_prec_t * pprec;
   long psave;
```

```
    long spar1;
    short spar2;
    char spar3;
    char pulld;
    void * splst;
    short xsize;
    short ysize;
    short xorg;
    short yorg;
    short flag;
    short borw;
    short borc;
    short papr;
    void * sprite;
    short curw;
    short curc;
    short uback;
    short uink;
    WM_blob_t * ublob;
    WM_pattern_t * upatt;
    short aback;
    short aink;
    WM_blob_t * ablob;
    WM_pattern_t * apatt;
    short sback;
    short sink;
    WM_blob_t * sblob;
    WM_pattern_t * spatt;
    void * help;
    short ninfo;
    short ninob;
    WM_infw_t * pinfo;
    short nlitm;
    WM_litm_t * plitm;
    short nappl;
    WM_appl_t * pappl;
} WM_wwork_t;
```

5.4 Pointer Interface Calls

These routines allow C programs to call the Pointer Interface system calls.

5.4.1 int iop_flim (chanid_t, timeout_t, WM_wsiz_t * limits)

Find window limits. Values returned via the *'limits'* parameter.

Returns standard QDOS/SMS error codes.

```
typedef struct WM_wsiz {
   short xsize;
   short ysize;
   short xorg;
   short yorg;
} WM_wsiz_t;
```

5.4.2 int iop_lblb (chanid_t, timeout_t, short xs, short ys, short xe, short ye, WM_blob_t *, WM_pattern_t *)

Draw a line of blobs from xs,ys to xe,ye.

Returns standard QDOS/SMS error codes.

NOTE: Not supported on QL high-color drivers

5.4.3 int iop_outl (chanid_t, timeout_t, short shadx, short shady, short keep(0/1), WM_wsiz_t *)

Set outline. Keep = 1 to keep the current contents of the window.

Returns standard QDOS/SMS error codes.

5.4.4 int iop_pick (chanid_t, timeout_t, jobid_t job_ID)

Pick windows for a Job to the top.

Returns standard QDOS/SMS error codes.

5.4.5 void * iop_pinf (chanid_t, timeout_t, long *version)

Get Window Manager vector and version information. On success returns the address of the Window Manager vector. On failure returns the QDOS/SMS error code (which is negative).

N.B. Prior to Release 3 of the QPTR library, a value of 0 was returned on error. This means that code that used that version might need slight modification in its use of this call.

5.4.6 int iop_rptr (chanid_t, timeout_t, short *x, short *y, short termination_vector, WM_prec_t *)

Reads pointer and suspend until termination conditions (as specified in the termination vector parameter) or timeout occurs. Returns QDOS/SMS error code.

N.B. Prior to Release 3 of the QPTR library, the x and y parameters were treated as though they pointed to 'int' rather than 'short' values as the specification said. This means that code that used that version might need slight modification in its use of this call.

```
typedef struct WM_prec {
   chanid_t chid;
   unsigned short swnr;
   signed short xpos;
   signed short ypos;
   unsigned char kstk;
   unsigned char kprs;
   unsigned long evnt;
   signed short xsiz;
   signed short ysiz;
   signed short xorg;
   signed short yorg;
} WM_prec_t;
```

5.4.7 int iop_rpxl (chanid_t, timeout_t, short *x, short *y, short scan, short *pixel)

Read pixel. Returns QDOS/SMS error code.

N.B. Prior to Release 3 of the QPTR library, the x and y parameters were treated as though they pointed to 'int' rather than 'short' values as the specification said. This means that code that used that version might need slight modification in its use of this call.

5.4.8 int iop_rspw (chanid_t, timeout_t, WM_wsiz_t *save, short xorg, short yorg, int keepflag, void *save_area)

Restore partial window.

Returns QDOS/SMS error code.

5.4.9 void * iop_slnk (chanid_t, timeout_t, void * values, short start, short count)

Set pointer linkage. On success returns the base address of the linkage.

On failure returns a QDOS/SMS error code (which is a negative value).

5.4.10 int iop_spry (chanid_t, timeout_t, short x, short y, WM_blob_t *, WM_pattern_t *, long num_pixels)

Spray pixels of pattern within blob. Returns QDOS/SMS error code.

5.4.11 int iop_sptr (chanid_t, timeout_t, short *x, short *y, char origin_key)

Set pointer absolute (origin=0), relative to hit area (origin=-1).
Sets the absolute pointer position in x,y.

5.4.12 int iop_svpw (chanid_t, timeout_t, WM_wsiz_t *, short xorg, short yorg, short xsize, short ysize, void **save_area)

Save partial window. If xsize and ysize are zero, then the area should already exist. If they are non-zero then a new save area is set up and its address stored at the location specified by the 'save_area' parameter.
Returns QDOS/SMS error code.

5.4.13 int iop_swdf (chanid_t, timeout_t, long *wdef_list)

Sets window definition list.
Returns QDOS/SMS error code.

5.4.14 int iop_wblb (chanid_t, timeout_t, short x, short y, WM_blob_t *, WM_pattern_t *)

Write blob.
Returns QDOS/SMS error code.

5.4.15 int iop_wrst (chanid_t, timeout_t, void *save, char keep)

Restore window. If save==NULL, then do it from internal area, otherwise use area supplied by user. If keep!=0 then keep save area, otherwise, free it. Returns QDOS/SMS error code.

5.4.16 int iop_wsav (chanid_t, timeout_t, void *save, long length)

Save window area. If save==NULL and length==0, then the area is allocated internally. If not, area supplied by user is used. Returns QDOS/SMS error code.

5.4.17 int iop_wspt (chanid_t, timeout_t, short x, short y, WM_sprite_t *)

Write sprite.

Returns QDOS/SMS error code.

5.5 Window Manager Functions (C68 compatible)

These are C equivalents to the standard Window Manager calls available to assembler programmers. More details can be found in the QPTR manual.

5.5.1 int wm_chwin (WM_wwork_t *, short *dx, short *dy)

Change window position (automatic) or size (returns the dx,dy of the pointer). Returns QDOS/SMS error code on failure, 0 or a positive event number if successful.

5.5.2 int wm_clbdr (WM_wwork_t *)

If there is a current item, it is cleared: useful before re-drawing menus. Returns QDOS/SMS error code. N.B. This is a C68 extension to the standard Window Manager set of vectors.

5.5.3 int wm_cluns (WM_wwork_t *)

Close channel and unset window. (Actually a call to wm_unset then the channel is closed. Use it to get rid of pull-down windows. Returns QDOS/SMS error code. N.B. This is a C68 extension to the standard Window Manager set of vectors.

5.5.4 int wm_drbdr (WM_wwork_t *)

Draws a border using the current item definition in WM_wstat. Returns QDOS/SMS error code.

5.5.5 int wm_ename (chanid_t, QD_text_t * name)

Edit name (QDOS string): writes out current name, puts cursor at end. Returns QDOS/SMS error code. C.f. wm_rname.

5.5.6 int wm_erstr (long error_code, QD_text_t * reply_string)

Converts the error code to a QDOS string. Returns the QDOS/SMS error code.

```
typedef struct QD_text {
   short len;
   qchar_t chrs[1];
} QD_text_t;
```

5.5.7 void * wm_findv (chanid_t channel)

Check that the Window Manager has been loaded, and if so get the Window Manager Vector:

Returns the vector or NULL if not found.

It is not necessary to use this call if you have already used the *iop_pinf()* call to check for the presence of the Window Manager. This routine also stores the value of the Window Manager vector internally for use by the other *wm_xxxx* calls so that the user need not store the value.

N.B. This is a C68 extension to the standard Windows Manager set of vectors.

5.5.8 short wm_fsize (short *xsize, short *ysize, WM_wdef_t *)

Given a target size and a window definition, this routine returns the appropriate layout number and sets the size to the actual size. Returns layout size (or QDOS/SMS error code if Window Manager Vector not known). C.f wm_setup.

5.5.9 int wm_idraw (WM_wwork_t *, long bits)

Re-draws any of information windows 0-31. For each window required to be drawn, the corresponding bit in bits should be set.

Returns QDOS/SMS error code.

5.5.10 int wm_index (WM_wwork_t *, WM_swdef_t *)

Draws the index (not implemented), pan and scroll bars for an application sub-window.

5.5.11 int wm_ldraw (WM_wwork_t *, char select)

Loose menu Item Drawing.

Returns QDOS/SMS error codes.

5.5.12 int wm_mdraw (WM_wwork_t *, WM_swdef_t *, int select)

Draws all menu items (*select* =0) or those items with change bit set in status area (*select*<>0).

5.5.13 int wm_mhit (WM_wwork_t *, WM_appw_t *, short x, short y, short key, short event)

C68 compatible wrapper for wm.mhit. Can be called from application sub-window hit routine. C.f. wm__mhit.

5.5.14 short wm_msect (WM_wwork_t *, WM_appw_t *, short xpos, short ypos, short key, short event, WM_mctrl_t *)

Called from an application sub-window hit routine, wm_msect determines the section of a menu and whether a pan or scroll event has occurred. The general information is returned in the structure WM_mctrl. If there has been an pan/scroll event, this is returned (+ve) otherwise wm_msect returns 0 or a QDOS/SMS error code.

```
typedef struct WM_mctrl {
    short psit;
    short hitp;
    short barl;
    short evnt;
} WM_mctrl_t;
```

5.5.15 int wm_pansc (WM_wwork_t *, WM_appw_t *, WM_mctrl_t *)

If wm_msect returns a pan or scroll event: this routine can handle it.

5.5.16 int wm_prpos (WM_wwork_t *, short xpos, short ypos)

Position Primary Window. Returns QDOS/SMS error code.

---

int wm_pulld (WM_wwork_t *, short xpos, short ypos)

Pull down a secondary window. Returns QDOS/SMS error code.

5.5.17 int wm_rname (chanid_t, QD_text_t *)

Read name (QDOS string): writes out current name, puts cursor at start. Typing any printable character erases name. Returns QDOS/SMS error code. C.f. wm_ename.

5.5.18 int wm_rptr (WM_wwork_t *)

Returns QDOS/SMS error code.

5.5.19 int wm_setup (chanid_t, short xsize, short ysize, WM_wdef_t *, WM_wstat_t *, WM_wwork_t **, long alloc)

If the alloc size is non-zero, then a new Working Defintion area of this size will be allocated on the common heap. If it is zero, then it is assumed that the area is already allocated. Returns QDOS/SMS error code (if Window Manager Vector cannot be located).

5.5.20 int wm_smenu (short xscale, short yscale, WM_wstat_t *, WM_wdef_t **, WM_wwork_t **)

Setup standard sub-window menu. Returns QDOS/SMS error code if unable to find Window Manager.

5.5.21 int wm_stiob (WM_wwork_t *, void *object, short window nr, short object number)

Set information object. Returns QDOS/SMS error code.

5.5.22 int wm_stlob (WM_wwork_t *, void *; short item number)

Set loose object. Returns QDOS/SMS error code.

5.5.23 chanid_t wm_swapp (WM_wwork_t *, short window nr, long ink)

Set window to application window. Returns channel ID or QDOS/SMS error code.

5.5.24 chanid_t wm_swdef (WM_wwork_t *, WM_appw_t *, chanid_t channel)

Set channel to application sub-window. Does not set colours. Returns Channel ID or QDOS/SMS error code.

5.5.25 chanid_t wm_swinf (WM_wwork_t *, short window nr, long ink)

Set window to information window.
Returns channel ID or QDOS/SMS error code.

5.5.26 chanid_t wm_swlit (WM_wwork_t *, short window nr, long status)

Set window to loose item. Returns channel ID or QDOS/SMS error code.

5.5.27 chanid_t wm_swsec (WM_wwork_t *, WM_appw_t *, short xsection, short ysection, long ink)

Set window to application sub-window section. Returns channel ID or QDOS/SMS error code.

5.5.28 int wm_unset (WM_wwork_t *)

Unset window: obligatory before scrumpling the working definition. Also used to remove pull-down windows. Returns QDOS/SMS error code. C.f. *wm_cluns.*

5.5.29 int wm_upbar (WM_wwork_t *, WM_swdef_t *, short xsection, short ysection)

Update a section of the pan/scroll bar.

5.5.30 *int wm_wdraw (WM_wwork_t *)*

Draw window: after *wm_prpos* or *wm_pulld.*
Returns QDOS/SMS error code.

5.5.31 int wm_wrset (WM_wwork_t *)

Reset window definition.
Returns QDOS/SMS error code.

5.5.32 Window Manager Routines Referenced From Working Definition

int wm_smenu (...) referenced from wda_setr (assembly language)
int wm__mhit (...) referenced from WM_appw.hit
int wm__pnsc (...) referenced from WM_appw.ctrl

5.5.33 Window Manager Action (etc) Routine Wrappers

These wrappers allow C68 functions to be called from the Window Manager via the WM_action structure.

wm_actli(...) referenced from WM_litm.pact

wm_actme(...) referenced from WM_mobj.pact

wm_drwaw(...) referenced from WM_appw.draw

wm_hitaw(...) referenced from WM_appw.hit

wm_ctlaw(...) referenced from WM_appw.ctrl

5.5.34 Standard Sprites

The following pre-defined sprites that are commonly used in Pointer Environment programs are included in this library.

Any further contributions that could be added to this standard sprite list would be welcomed.

`struct WM_sprite wm_sprite_arrow`	Arrow symbol
`struct WM_sprite wm_sprite_cf1`	CTRL-F1 key symbol
`struct WM_sprite wm_sprite_cf2`	CTRL-F2 key symbol
`struct WM_sprite wm_sprite_cf3`	CTRL-F3 key symbol
`struct WM_sprite wm_sprite_cf4`	CTRL-F4 key symbol
`struct WM_sprite wm_sprite_f1`	F1 key symbol
`struct WM_sprite wm_sprite_f2`	F2 key symbol
`struct WM_sprite wm_sprite_f3`	F3 key symbol
`struct WM_sprite wm_sprite_f4`	F4 key symbol
`struct WM_sprite wm_sprite_f5`	F5 key symbol
`struct WM_sprite wm_sprite_f6`	F6 key symbol
`struct WM_sprite wm_sprite_f7`	F7 key symbol
`struct WM_sprite wm_sprite_f8`	F8 key symbol
`struct WM_sprite wm_sprite_f9`	F9 key symbol
`struct WM_sprite wm_sprite_f10`	F10 key symbol
`struct WM_sprite wm_sprite_hand`	
`struct WM_sprite wm_sprite_insg`	
`struct WM_sprite wm_sprite_insl`	
`struct WM_sprite wm_sprite_left`	
`struct WM_sprite wm_sprite_move`	Move symbol. Used to indicate item used a window.
`struct WM_sprite wm_sprite_null`	
`struct WM_sprite wm_sprite_size`	Size symbol. Used to indicate menu item that is used to re-size a window.
`struct WM_sprite wm_sprite_sleep`	Sleep symbol. Used to indicate menu item for putting a program to sleep.
`struct WM_sprite wm_sprite_wake`	Wake symbol. Used to indicate a menu item for waking a program.
`struct WM_sprite wm_sprite_zero`	This is really just a blank background. It is used as the pattern mask for many of the sprites.

6 Change History

The following is a brief summary of the significant changes made to this document. It is intended to help those who are upgrading from previous releases to determine what (if anything) has changed in this document.

- 30 Oct 93 DJW - Extensive changes as part of making the QPTR library usable with C68 Release 4.
- 02 Nov 93 DJW - Added list of sprites that are included in this library.
- 13 Aug 94 DJW - Changed the definition of the *iop_rspw()* routine to make the last parameter only 'void *' (it was 'void **').
- 03 Apr 95 DJW - Changed all function definitions reflect fact that all structures are now 'typedef'ed. Also 'char *' parameters changed to more generic 'void *' format.

Contents