Rtfs with exFAT

API Reference Guide

Revised January 2011

For on-line viewing navigate using the Adobe Acrobat's **Bookmarks** tab or use hyperlinks in the table of contents.

http://www.ebsembeddedsoftware.com

TABLE OF CONTENTS

Initialize and shutdown

pc_ertfs_init

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

pc_ertfs_init() must be called by the application before it calls any Rtfs API functions. **pc_ertfs_init()** in turn calls an application specific callback subroutine named **rtfs_init_configuration()** configure Rtfs and acquire operating memory.

SUMMARY

BOOLEAN **pc_ertfs_init** (void)

DESCRIPTION

This function works in conjunction with an application supplied callback subroutine named **rtfs_init_configuration()** to configure and initialize Rtfs memory. It then allocates memory dynamically, if so instructed, and allocates necessary semaphores for the operating system porting guide.

NOTE: Please consult the manual page for **rtfs_init_configuration()** *for detailed instructions on what this function must provide to Rtfs.*

RETURNS

TRUE	All memory and system resource initialization succeeded and Rtfs is usable
FALSE	Memory and system resource initialization failed and Rtfs is not usable

This function does not set errno.

SEE ALSO

rtfs_init_configuration()

rtfs_init_configuration

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Set system wide operating parameters. This is a callback subroutine that must be provided by the application layer to configure Rtfs and provides operating memory.

SUMMARY

void **rtfs_init_configuration** (preply)

struct rtfs_init_resource_reply *preply	Contains operating parameters for Rtfs .

DESCRIPTION

rtfs_init_configuration() configures the global operating parameters and buffering that Rtfs will use. It is called by **pc_ertfs_init()** when Rtfs is first initialized.

A reference version of this function is provided in the file named rtfsconfig.c in the subdirectory rtfsprojects\msvc.net\source. These files and rtfscallbacks.c should be copied to your project file and reconfigured to suit your application's needs.

The reference version of **rtfs_init_configuration()** is controlled by the following compile time constants defined in rtfsconfig.h.

RTFS_CFG_SINGLE_THREADED
RTFS_CFG_INIT_DYNAMIC_ALLOCATION
RTFS_CFG_MAX_DRIVES
RTFS_CFG_MAX_FILES
RTFS_CFG_MAX_SCRATCH_BUFFERS
RTFS_CFG_MAX_SCRATCH_DIRS
RTFS_CFG_MAX_USER_CONTEXTS
RTFS_CFG_MAX_REGION_BUFFERS
RTFS_CFG_SINGLE_THREADED_USER_BUFFER_SIZE
RTFS_CFG_SINGLE_THREADED_FAILSAFE_BUFFER_SIZE
RTFS_CFG_DIRS_PER_DRIVE
RTFS_CFG_DIRS_PER_USER_CONTEXT

rtfs_init_configuration() *must initialize an* **rtfs_init_resource_reply***.*

struct rtfs_init_resource_reply {

int	*max_drives*
int	*max_scratch_buffers*
int	*max_file*
int	*max_user_contexts*
int	max_region_buffers
int	spare_user_directory_objects
int	spare_drive_directory_objects
int	use_dynamic_allocation
int	run_single_threaded
dword	single_thread_buffer_size
dword	single_thread_fsbuffer_size
void *	single_thread_buffer
void *	single_thread_fsbuffer
void *	mem_drive_pool
void *	mem_mediaparms_pool
void *	mem_block_pool
void *	mem_block_data
void *	mem_file_pool
void *	mem_finode_pool
void *	mem_finodeex_pool
void *	mem_drobj_pool;
void *	mem_region_pool;
void *	mem_user_pool;
void *	mem_user_cwd_pool

};

struct rtfs_init_resource_reply – *This table describes the fields that must be initialized to configure Rtfs. A sample version of* **rtfs_init_configuration()** *is provided in rtfsconfig.c. It may be modified for your application's requirements.*

Field name	Meaning
max_drives	The maximum number of drives that may be mounted at one time. The maximum value is 26.
max_files	The maximum number of files that may be opened at one time.
max_scratch_buffers	The number of blocks in the scratch buffer pool. These are used by Rtfs as scratch memory buffers when performing certain operations. Each scratch buffer consumes approximately 536 bytes. The default value is 32 but it may be reduced to as low as 8 in most applications.
spare_drive_directory_objects spare_user_directory_objects	These constants controls allocation of extra "dirent" objects for use in certain non-file operations like **pc_enumerate()** and **pc_getcwd()** that consume dirent structures as they execute. The default values are 16 and 4 respectively. They should not be changed lightly, but if ram is a precious resource you may wish to reduce them and then verify that

	your application still runs correctly.
max_region_buffers	The number of 12 byte **REGION_FRAGMENT** structures dedicated to run length encoding of cluster fragments in open files and free space. The default setting is 5000, this consumes 60 K and provides enough buffering for 5000 fragments in free-space and in open files. Increase this value if your application can spare the memory. *If not enough fragment buffers are available Rtfs resorts to much slower disk based FAT scans when allocating clusters. If not enough buffers are available for open files then file IO operations will fail.*
max_user_contexts	The number of separate threads that will have their own separate current working directory, and errno contexts.
run_single_threaded	Set to one to force Rtfs to run in single threaded mode. In single threaded mode all drive accesses use the same semaphore and thus execute sequentially. This eliminates the need for individual user buffers and failsafe restore buffers per drive, resulting in reduced memory consumption, with marginal to no performance degradation in most systems.
The following fields, **single_thread_buffer_size** and **single_thread_fsbuffer_size**, should be set only if **run_single_threaded** is true. If **run_single_threaded** is not true, buffers must be provided for each mounted drive. These buffers are used for certain bulk fat table access operations and for Failsafe journaling respectively. They are specified in bytes and since they are shared among all drives they must be large enough to accommodate all media types, for optimal performance with NAND flash they should be the size of an erase block. For large rotating media, buffers sized 32 K or 64 K provide performance improvements.	
single_thread_buffer_size	Set to zero if **run_single_threaded** is zero.
single_thread_fsbuffer_size	Set to zero if **run_single_threaded** is zero or if you are not using Failsafe.
use_dynamic_allocation	Set to 1 to instruct Rtfs to dynamically allocate system wide resources. Set to 0 to instruct Rtfs that system wide resources are provided.
If **use_dynamic_allocation** is set to zero, the following fields must be initialized with pointers to enough space for the objects being configured. The sample code provided in rtfsconfig.c, does this for you and it is unlikely that you will need to modify it.	

void *single_thread_buffer;	void *mem_finode_pool;
void *single_thread_fsbuffer;	void *mem_finodeex_pool;
void *mem_drive_pool;	void *mem_drobj_pool;
void *mem_mediaparms_pool;	void *mem_region_pool;
void *mem_block_pool;	void *mem_user_pool;
void *mem_block_data;	void *mem_user_cwd_pool;
void *mem_file_pool;	

RETURNS

Nothing	

If an error occurred: errno is set to one of the following:

Errno is not set	

pc_ertfs_shutdown

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

void **pc_ertfs_shutdown** (void)

SUMMARY

Shut down Rtfs.

DESCRIPTION

pc_ertfs_shutdown() puts Rtfs in an un-initialized state releasing all allocated memory and system resources. Rtfs may be restarted by calling **pc_ertfs_init()**.

RETURNS

Nothing	

If an error occurred: errno is set to one of the following:

Errno is not set	

Media driver interface

pc_rtfs_media_insert

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

This function must be called by the device driver when media is inserted or the device powered up. It sets operating parameters for the media.

SUMMARY

int **pc_rtfs_media_insert**(struct rtfs_media_insert_args *pmedia_parms)

struct rtfs_media_insert_args *pmedia_parms	Operating parameters for Rtfs to use when accessing this device. See below for a complete description of pmedia_parms.

DESCRIPTION

pc_rtfs_media_insert() alerts Rtfs that new media is available and provides it with operating parameters and buffering that Rtfs stores internally and uses when *accessing* this device.

The device driver must pass the address of an initialized **rtfs_media_insert_args *structure*** *to* **pc_rtfs_media_insert()** *.* Configuration parameters are copied to internal structures, so the configuration structure itself may reside on the stack.

struct rtfs_media_insert_args {

void *	devhandle
int	device_type
int	unit_number
int	write_protect
dword	media_size_sectors
dword	numheads
dword	numcyl
dword	Secptrk
dword	sector_size_bytes
dword	eraseblock_size_sectors
int	(*device_io) ()
int	(*device_erase) ()

int	(*device_ioctl) ()
int	(*device_configure_media)()
int	(*device_configure_volume)()

};

struct rtfs_media_insert_args	
devhandle	Rtfs will pass this handle as one of the arguments to certain device driver callback functions. The device layer uses this to identify the device and retrieve system specific information. Rtfs does not interpret devhandle, but it must be a unique non-zero value.
device_type	Rtfs will pass this device_type as one of the arguments to certain driver callback functions. The device layer uses this to identify the device type when providing configuration information. Rtfs does not interpret device_type.
unit_number	Rtfs will pass unit_number as one of the arguments to certain driver callback functions. The device layer uses this to identify the device type when providing configuration information. Rtfs does not interpret device_type.
write_protect	Initial write protect state of the device. Rtfs will not write to the media if this is non-zero. The driver can change the write protect state later by calling **pc_rtfs_media_alert()**.
media_size_sectors	Total number of addressable sectors on the media.
eraseblock_size_sectors	Sectors per erase block for NAND devices. Must be set to zero for media without erase blocks
Numheads, numcyls and secptrk must be valid HCN values, they are placed into the FAT boot structures where needed but they are otherwise not used. HCN values should be calculated and then clipped to fit within the legal values.	
numheads	Must be <= 255
numcyl	Must be <= 1023
Secptrk	Must be <= 63
sector_size_bytes	512, 124, 2048 etc.
(*device_io) ()	Device sector IO function
(*device_erase) ()	Device erase block erase function
(*device_ioctl) ()	Device IO control function
(*device_configure_media)()	Device media Configuration function
(*device_configure_volume)()	Device volume mount Configuration function

};

device_io – media driver callback

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

This is a callback function that must be implemented for the target device. This function is called by Rtfs and should not be called externally.

SUMMARY

int (*device_io)(void *devhandle, void *pdrive, dword sector, void *buffer, dword count, int reading)

DESCRIPTION

This function is called when Rtfs wants to perform sector reads or writes to media that was attached by **pc_rtfs_media_insert()** .

devhandle	Handle passed to **pc_rtfs_media_insert()** when the device was inserted. **(*device_io)** may use this to locate media properties and state.
pdrive	Void pointer to the Rtfs drive structure. Advanced device drivers may caste this with (DDRIVE *) to access the drive structure.
sector	Starting sector number to read or write
buffer	Buffer to read to write from
count	Number of sectors to transfer
reading	True for a read, False for a write request

RETURNS

0	Returned if IO failed.
1	Returned if IO successful

device_erase – media driver callback

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

This is a callback function that must be implemented for the target device. This function is called by Rtfs and should not be called externally.

SUMMARY

int (*device_erase)(
void *devhandle, void *pdrive, dword start_sector, dword nsectors)

DESCRIPTION

This function is called when Rtfs wants to erase sectors on media that was attached by **pc_rtfs_media_insert()**.

Note: This function will only be called if the value of eraseblock_size_sectors passed to **pc_rtfs_media_insert()** is non zero.

NOTE: The region spanned by start_sector and nsectors is not always guaranteed to be on erase block boundaries ! If the volume was formatted by Rtfs with enforced erase block alignment the span will be erase block bound, but if the media was not formatted this way the span could possibly not lie on erase block boundaries. If the span is not erase block bound the device driver should return success without erasing the sectors.

devhandle	Handle passed to **pc_rtfs_media_insert()** when the device was inserted. **(*device_erase)** may use this to locate media properties and state.
pdrive	Void pointer to the Rtfs drive structure. Advanced device drivers may caste this with (DDRIVE *) to access the drive structure.
sector	Starting sector number to erase
nsectors	Number of sectors to erase

RETURNS

0	Returned if erase failed.
1	Returned if erase was successful

device_ioctl – media driver callback

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

This is a callback function that must be implemented for the target device. This function is called by Rtfs and should not be called externally.

SUMMARY

int (*device_ioctl)(void *devhandle, void *pdrive, int opcode, int iArgs, void *vargs)

DESCRIPTION

This function is called when Rtfs wants to perform an ioctl subroutine call directly to the device driver. Most devices can simply return 0 whenever this function is called. See the opcode descriptions below for more information.

devhandle	Handle passed to **pc_rtfs_media_insert()** when the device was inserted. **(*device_ioctl)** may use this to locate media properties and state.
pdrive	Void pointer to the Rtfs drive structure. Advanced device drivers may caste this with (DDRIVE *) to access the drive structure.
opcode	Ioctl opcode to perform.
iArgs	Integer argument for ioctl.
vArgs	Pointer argument for ioctl.

OPCODE	Description
RTFS_IOCTL_FORMAT	Format the media if that is a supported operation. Flash media drivers may erase all blocks on the media. Most other media type don't require formatting. These devices should return 0 when asked to format.
RTFS_IOCTL_INITCACHE	Advanced feature. Devices should return 0 when passed this argument.
RTFS_IOCTL_FLUSHCACHE	Advanced feature. Devices should return 0 when passed this argument.

RETURNS

-1	If the command failed.
0	If the command was successful

device_configure_media – media driver callback

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

This is a callback function that must be implemented for the target device. This function is called by Rtfs and should not be called externally.

SUMMARY

int (*device_configure_media)(
struct rtfs_media_insert_args *pmedia_parms,
struct rtfs_media_resource_reply *pmedia_config_block ,
int sector_buffer_required)

DESCRIPTION

This function is called by Rtfs while it is executing **pc_rtfs_media_insert()** on behalf of the device driver. **pc_rtfs_media_insert()** passes a **rtfs_media_insert_args** structure containing information about the device. This function must fill in the **rtfs_media_resource_reply** structure with configuration and buffering information.

See the manual page for **pc_rtfs_media_insert()** for a descriptions of the **rtfs_media_insert_args** structure**.**

struct rtfs_media_resource_reply {

int	use_dynamic_allocation
int	requested_driveid
int	requested_max_partitions
int	use_fixed_drive_id
dword	device_sector_buffer_size_bytes
byte	*device_sector_buffer_base
void	*device_sector_bffer_data

};

struct rtfs_media_resource_reply	
use_dynamic_allocation	Set to 1 to instruct Rtfs to dynamically allocate media buffers. Set to 0 to instruct Rtfs that media buffers are provided.
requested_driveid	Drive Id (0 – 25) to assign to the media if not partitioned or to the first partition on the media if it is.
requested_max_partitions	Maximum number of volumes to mount on this media. *Note:* **device_configure_volume()** *must be prepared to configure this many*

	partitions.
use_fixed_drive_id	Must be set to 1.
device_sector_buffer_size_bytes	This buffer is used for certain bulk FAT table access operations. It is specified in bytes, for optimal performance with NAND flash it should be the size of an erase block. For large rotating media, buffers sized 32 K or 64 K provide performance improvements. *Note: If* **rtfs_init_configuration()** *was configured for* **run_single_threaded***, then* device_sector_buffer_size_bytes *should be set to zero.*
*device_sector_buffer_data	If **use_dynamic_allocation** is zero, this must be initialized to point to an area of ram device_sector_buffer_size_bytes wide. If **use_dynamic_allocation** is one, leave this field blank, Rtfs will allocate the necessary memory.
*device_sector_buffer_base	Internal, do not set.

RETURNS

0	Return if successful
-1	Return if an unsupported device type was encountered
-2	Returned if out of resources

device_configure_volume – media driver callback

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION
This is a callback function that must be implemented for the target device. This function is called by Rtfs and should not be called externally.

SUMMARY
Int (*device_configure_volume)(
struct rtfs_volume_resource_request *prequest_block,
struct rtfs_volume_resource_reply *pvolume_config_block)

DESCRIPTION
This function is called when the volume on a device must be configured. This function is passed in an **rtfs_volume_resource_request** structure containing information about the volume. The configuration of the volume is passed back in the **pvolume_config_block**. The purpose of this function is to fill in the values of the **rtfs_volume_resource_reply** structure. Below are description of the **rtfs_volume_resource_request** structure and the **rtfs_volume_resource_reply** structure.

Note: If an exFAT volume is being mounted some additional resources are required. These resources must be provided though the callback layer. For more information see the manual page for:
rtfs_sys_callback(RTFS_CBS_GETEXFATBUFFERS)*.*

struct rtfs_volume_resource_request {

void	*devhandle
int	device_type
int	unit_number
int	driveid
int	partition_number
dword	volume_size_sectors
dword	sector_size_bytes
dword	eraseblock_size_sectors
int	buffer_sharing_enabled
int	failsafe_available

};

struct rtfs_volume_resource_request	
*devhandle	Device driver access Handle
device_type	Device type returned by **device_configure_media()**
unit_number	Unit number type returned by **device_configure_media()**
Driveid	Drive letter (0-25).
partition_number	Which partition it is.
volume_size_sectors	Total number of addressable sectors on the partition or media containing the volume
sector_size_bytes	Sector size in bytes: 512, 2048, etc…
eraseblock_size_sectors	Sectors per erase block. Zero for media without erase blocks

buffer_sharing_enabled	If 1, Rtfs is configured to share restore buffers.
failsafe_available	If 1, failsafe is available and operating policy and failsafe buffering may select failsafe.

struct rtfs_volume_resource_reply {

int	use_dynamic_allocation
dword	drive_operating_policy
dword	n_sector_buffers
dword	n_fat_buffers
dword	fat_buffer_page_size_sectors
dword	n_file_buffers
dword	file_buffer_size_sectors
dword	fsrestore_buffer_size_sectors
dword	fsjournal_n_blockmaps
void	*blkbuff_memory
void	*fatbuff_memory
void	*filebuff_memory
void	*fsfailsafe_context_memory
void	*fsjournal_blockmap_memory
byte	*sector_buffer_base
byte	*file_buffer_base
byte	*fat_buffer_base
byte	*failsafe_buffer_base
byte	*failsafe_indexbuffer_base
void	*sector_buffer_memory
void	*file_buffer_memory
void	*fat_buffer_memory
void	*failsafe_buffer_memory
void	*failsafe_indexbuffer_memory

};

struct rtfs_volume_resource_reply	
use_dynamic_allocation	Set to one to request Rtfs to allocate structures and buffers dynamically.
drive_operating_policy	Drive operating policy, defaults to zero, See app notes.
n_sector_buffers	Total number of sector sized directory buffers.
n_fat_buffers	Total number of FAT table buffers.
fat_buffer_page_size_sectors	Number of sectors per FAT table buffer.
Required for NAND Flash. Otherwise use defaults.	
n_file_buffers	Total number of file buffers.
file_buffer_size_sectors	File buffer size in sectors.
Required for Failsafe. Otherwise use defaults.	
fsrestore_buffer_size_sectors	Failsafe restore buffer size in sectors.
fsjournal_n_blockmaps	Number of Failsafe sector remap records provided. Determine the number of outstanding remapped sectors permitted.
The rest of the fields may be left blank if **use_dynamic_allocation** is selected. If dynamic allocation is not selected please populate the following fields according to	

the descriptions.	
*blkbuff_memory	Must point to **n_sector_buffers * sizeof(BLKBUFF**) bytes (*sizeof(BLKBUFF) is around 40 bytes)*
*fatbuff_memory	Must point to **n_fat_buffers * sizeof(FATBUFF)** bytes. *sizeof(FATBUFF) is around 40 bytes)*
Required for NAND Flash. Otherwise use defaults.	
*filebuff_memory	Must point to **n_file_buffers * sizeof(BLKBUFF**) bytes. *sizeof(BLKBUFF) is around 40 bytes)*
Required for Failsafe. Otherwise use defaults	
*fsfailsafe_context_memory	Must point to **sizeof(FAILSAFECONTEXT**) bytes. *sizeof(FAILSAFECONTEXT) is around 300 bytes)*
*fsjournal_blockmap_memory	Must point to **fsjournal_n_blockmaps** * **sizeof(FSBLOCKMAP)** bytes. sizeof(FBBLOCKMAP) equals 16
These pointers contain arrays do require IO address alignment if that is a system requirement	
*sector_buffer_memory	Must point to **sector_size * n_sector_buffers** bytes.
*fat_buffer_memory	Must point to **sector_size * n_fat_buffers * fat_buffer_page_size_sectors** bytes.
Required for NAND Flash. Otherwise use defaults.	
*file_buffer_memory	Must point to **sector_size*n_file_buffers******file_buffer_size_sectors** bytes.
Required for Failsafe. Otherwise use defaults	
*failsafe_buffer_memory	Must point to **sector_size * fsrestore_buffer_size_sectors** bytes.
*failsafe_indexbuffer_memory	Must point to **sector_size** bytes.
These fields are used internally, do not change them.	
*sector_buffer_base	*failsafe_buffer_memory;
*file_buffer_base	*failsafe_indexbuffer_memory
*fat_buffer_base	

RETURNS

0	Return if successful
-1	Return if an unsupported device type was encountered
-2	Returned if out of resources

pc_rtfs_media_alert

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

This function must be called from the device driver when the write protect status changes or the device is ejected.

SUMMARY

int **pc_rtfs_media_alert**(void *devhandle, int alertcode, void *vargs)

DESCRIPTION

This function takes as arguments, the devhandle that was passed to **pc_rtfs_media_insert()** when the device was inserted and an alert code.

devhandle	The same handle that was passed to pc_rtfs_media_insert when the device was inserted.
alertcode	The alert that the driver is passing to Rtfs.
vargs	Unused.

Alert Codes	Behavior
RTFS_ALERT_EJECT	All drive identifiers, mount structures, control structure, semaphores and buffers associated with the device are released.
RTFS_ALERT_WPSET	Sets the internal write protect status for the media. Rtfs will not write to the media unless the status is cleared.
RTFS_ALERT_WPCLEAR	Clear the internal write protect status for the media. Rtfs will write to the media.

RETURNS

Nothing	

If an error occurred: errno is set to one of the following:

Errno is not set	

Application callbacks

rtfs_sys_callback

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

System service callback function.

SUMMARY

int **rtfs_sys_callback**(int cb_code, void *pvargs)

DESCRIPTION

This call back provides the system services. Rtfs calls this function for certain system functions. The cb_codes are described in the table below. Sample source for this function is located in rtfscallbacks.c.

cb_code	Required Functionality
RTFS_CBS_INIT	This callback is made by **pc_ertfs_init** before any other callbacks are made or any operating system porting layer functions are called. System initializations like opening the terminal window may be performed by the handler.
RTFS_CBS_PUTS	Print the string pointed to by (char*)pvargs to the console.
RTFS_CBS_GETS	Retrieve a string from the console and store in (char*)pvargs.
RTFS_CBS_GETDATE	Retrieve the system date and store in (char*)pvargs. *Modify the function named* ***pc_get_system_date()*** *in rtfscallbacks.c to interface with your system's calendar function.*
RTFS_CBS_ 10MSINCREMENT	exFat only - Retrieve the one byte 10 millisecond precision component for the previous RTFS_CBS_GETDATE call. (0-199 for up to 1990 milliseconds).
RTFS_CBS_UTCOFFSET	exFat only - Retrieve the one byte value to place in the offset from UTC field. The default is 0xf0, Eastern time zone US.
RTFS_CBS_POLL_DEVICE_READY	Poll for device changes if your system cannot provide insert/remove interrupts.
RTFS_CBS_GETEXFATBUFFERS	This callback is made by Rtfs when it detects insertion of an exFat volume. The callback layer is passed a structure of type EXFATMOUNTPARMS which contains informational fields suggesting what the

	callback should provide. The callback layer must provide the necessary buffering. Note: If the callback can't provide the memory it should set all return values to 0. See the table below describing the fields.
RTFS_CBS_RELEASEEXFATBUFFERS	This callback is made by Rtfs when it detects removal of an exFat volume. The callback layer is passed a structure of type EXFATMOUNTPARMS which contains informational fields plus values that were allocated by **RTFS_CBS_GETEXFATBUFFERS.** The callback should free the memory. If staic pools are in use driveID my be used to identify the pool.

EXFATMOUNTPARMS Field descriptions. (see **RTFS_CBS_GETEXFATBUFFERS**)	
These values are passed in.	
driveID	integer 0-25 == A:-Z: - Informational but you may use it as a handle to help keep track of static buffer pools if you are not using dynamic allocation.
pdr	Informational void pointer, you may caste this to access the drive structure directly from the callback routine.
SectorSizeBytes	Sector size – You will need this to allocate buffers.
BitMapSizeSectors	This value contains the size of the volume's free space bitmap (BAM). If you allocate enough mempry to buffer the whole BAM then no page swapping of the BAM is required. Otherwise if less than the optimal value is allocated Rtfs will swap the BAM sectors to disk as required. Note: exFAT requires one bit in the BAM per cluster. (one sector per 4096 clusters). The BAM of a 512 GIG drive is approximately 450 sectors (225k). Assuming memory starved systems the example provided arbitrarily limits the size of the BAM cache to 64 sectors (32 K), but this can be removed.
UpcaseSizeBytes	Size to allocate for the Upcase table cache. If the volume has a standard upcase table Rtfs will uses a precompiled standard table and this value will be zero, because. Otherwise this value will be 128K. If the value is 0 you need not provide any memory. If the value is 128K you may either provide 128 K of memory for full upcase support or you may allocate no memory and Rtfs will use the internal table to upcase only the lower 128 characters.
These values are returned.	
BitMapBufferSizeSectors	Return the number of sectors allocated for the bit map cache up to BitMapSizeSectors.
BitMapBufferPageSizeSectors	You should always return 1.

BitMapBufferCore	A memory array of size (BitMapBufferSizeSectors * SectorSizeBytes). Up to (BitMapSizeSectors * SectorSizeBytes)
BitMapBufferControlCore	Must return a memory array of size: **sizeof(FATBUFF) *** **(BitMapSizeSectors/ BitMapBufferPageSizeSectors)** (not BitMapBufferSizeSectors) sizeof(FATBUFF) is approximately 50 bytes so in the 512 GIG example above we would return (450*50) or approximately 22K. *The memory consumption may be reduced by setting BitMapBufferPageSizeSectors to a larger value, but this is not recommended unless memory is very tight.*
UpCaseBufferCore	Return 0 or a pointer to UpcaseSizeBytes bytes (128K).

RETURNS

Nothing	

If an error occurred: errno is set to one of the following:

Errno is not set	

rtfs_app_callback

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Application callback function.

SUMMARY

int **rtfs_app_callback**(int cb_code, int iarg0, int iargs1, void *pvargs)

DESCRIPTION

This function provides the callback for the application layer. The cb_codes and necessary parameters are described in the table below. Sample source for this function is located in rtfscallbacks.c.

cb_code	Required Functionality
Informational codes, no response is required	
RTFS_CBA_INFO_MOUNT_ STARTED	Called when a mount has been started. iarg0 contains the drive number.
RTFS_CBA_INFO_MOUNT_ FAILED	Called when a mount has failed. iarg0 contains the drive number.
RTFS_CBA_INFO_MOUNT_ SUCCEEDED	Called when a mount has succeeded iarg0 contains the drive number.
Rtfs ProPlus informational and response codes. These codes may be used to in certain application settings.	
RTFS_CBA_ASYNC_MOUNT _CHECK	Check if the current mount should proceed or if it should fail and request an asynchronous mount. iarg0 contains the drive number. Return 0 to proceed with the mount. Return 1 to abort the mount and request an asynchronous mount. *Note: The API call that initiated the mount will fail with **errno** set to **PENOTMOUNTED**.*
RTFS_CBA_ASYNC_MOUNT _START	Start an asynchronous mount. This will be called if **RTFS_CBA_ASYNC_MOUNT_CHECK** returned 1. This callback should signal the application to call **pc_diskio_async_mount_start()** to start an asynchronous mount on the drive number contained in iarg0. *Note: A foreground or background task must execute **pc_async_continue** to complete the mount.*
RTFS_CBA_ASYNC_DRIVE_ COMPLETE	An asynchronous drive operation has completed. iarg0 contains the drive number. iarg1 contains the id of the completed operation. iarg2 contains the completion status. *See the RtfsProPlus - Asynchronous operations API manual section for more information.*
	An asynchronous file operation has completed.

RTFS_CBA_ASYNC_FILE_COMPLETE	iarg0 contains the file descriptor. iarg1 contains the completion status. *See the RtfsProPlus - Asynchronous operations API manual section for more information.*
RTFS_CBA_DVR_EXTRACT_RELEASE	A DVR extract file has been released from sharing sectors with the circular buffer and may be closed. iarg0 contains the file descriptor. iarg1 contains the status. *See the RtfsProPlusDVR - Circular File IO API manual section for more information.*

RETURNS

0	Rtfs proceeds with default behavior.
1	For non-informational callbacks returning 1 alters behavior.

If an error occurred: errno is set to one of the following:

Errno is not set	

rtfs_diag_callback

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Diagnostic callback function.

SUMMARY

void **rtfs_diag_callback**(int cb_code, int iarg0)

DESCRIPTION

Provides an interface for fielding Rtfs asserts and for monitoring Rtfs errnos and to detect when device IO errors occur. Sample source for this function is located in rtfscallbacks.c.

cb_code	Required Functionality
RTFS_CBD_ASSERT	Monitor for when Rtfs detects an unexpected internal state.
RTFS_CBD_ASSERT_TEST	Monitor for when an Rtfs regression test fails
RTFS_CBD_IOERROR	Monitor for IO errors. iarg0 contains the drive number.
RTFS_CBD_SETERRNO	Inspect Rtfs errno values and monitor for system errors. iarg0 contains the error value.

rtfscallbacks.c. provide an example of **rtfs_diag_callback** with a switch table that may be populated to monitor for the following error conditions.

Normal application errors	Device level failures	Resource errors
PEACCES	PEDEVICEFAILURE	PERESOURCEBLOCK
PEBADF	PEDEVICENOMEDIA	PERESOURCEFATBLOCK
PEEXIST	PEDEVICEUNKNOWNMEDIA	PERESOURCEREGION
PENOENT	PEDEVICEWRITEPROTECTED	PERESOURCEFINODE
PENOSPC	PEDEVICEADDRESSERROR	PERESOURCEDROBJ
PESHARE	PEINVALIDBPB	PERESOURCEDRIVE
PEINVALIDPARMS	PEIOERRORREAD	PERESOURCEFINODEEX
PEINVAL	PEIOERRORWRITE	PERESOURCEFINODEEX64
PEINVALIDPATH	PEIOERRORREADMBR	PERESOURCESCRATCHBLOCK
PEINVALIDDRIVEID	PEIOERRORREADBPB	PERESOURCEFILES
PECLOSED	PEIOERRORREADINFO32	PECFIONOMAPREGIONS
PETOOLARGE	PEIOERRORREADBLOCK	PERESOURCEHEAP
Other application errors	PEIOERRORREADFAT	PERESOURCESEMAPHORE
PENOEMPTYERASEBLOCKS	PEIOERRORWRITEBLOCK	PENOINIT
PEEINPROGRESS	PEIOERRORWRITEFAT	PEDYNAMIC
PENOTMOUNTED	PEIOERRORWRITEINFO32	PERESOURCEEXFAT
PEEFIOILLEGALFD	Corrupted volume errors	
PE64NOT64BITFILE	PEINVALIDCLUSTER	
	PEINVALIDDIR	
	PEINTERNAL	

RETURNS

Nothing	

rtfs_failsafe_callback

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Failsafe run time configuration callback function.

SUMMARY

void **rtfs_failsafe_callback**(int cb_code, int driveno, int iarg0, void *pvargs, void *pvargs1)

DESCRIPTION

This callback provides the functionality previously provided by multiple callback functions that were recompiled along with the Failsafe source code. This callback interface provides the same functionality as the previous interface and defaults to the same configuration. Listed below are the cb_codes available for this function. Sample source for this function is located in rtfscallbacks.c.

RTFS_CB_FS_RETRIEVE_FIXED_JOURNAL_LOCATION
RTFS_CB_FS_FAIL_ON_JOURNAL_FULL
RTFS_CB_FS_FAIL_ON_JOURNAL_RESIZE
RTFS_CB_FS_RETRIEVE_JOURNAL_SIZE
RTFS_CB_FS_RETRIEVE_RESOTRE_STATEGY
RTFS_CB_FS_FAIL_ON_JOURNAL_CHANGED
RTFS_CB_FS_CHECK_JOURNAL_BEGIN_NOT
RTFS_CB_FS_RETRIEVE_FLUSH_STRATEGY

Note: For more information on the cb_codes and operations see the FailsafeTechnicalReferenceManual under Callback API

All Rtfs packages - Basic API

pc_diskclose

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Unconditionally dismount a volume without flushing and optionally clear values and buffers established by **device_configure_volume**.

*Note: You must call **pc_diskflush()** before calling **pc_diskclose()** if you wish to flush the disk before closing,*

SUMMARY

#include <rtfs.h>

BOOLEAN pc_diskclose(byte *driveid, BOOLEAN clear_init)

Driveid	Name of the volume "A:" "B:" etc.
clear_init	If clear_init is **TRUE**, all buffers and configuration values provided by **device_configure_volume** are released.

DESCRIPTION

This routine unconditionally dismounts a volume if it is currently mounted. There is no flushing of FAT buffers, file buffers, block buffers or of Failsafe.

*Note: To flush the disk before closing, call **pc_diskflush()** before you call **pc_diskclose()**.*

If clear_init is **TRUE**, the configuration is cleared. This releases all buffers that were assigned to the drive by **device_configure_volume.** The next time the drive is accessed **device_configure_volume** will be called.

Note: This function is used mainly for testing, a better way to dismount a drive is to flush it and then call pc_rtfs_media_alert.

RETURNS

TRUE	Success
FALSE	Invalid drive specified in an argument

Application Level Error Return Codes

PEINVALIDDRIVEID	Invalid drive specified in an argument

pc_diskflush

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Flush the FAT and all files to a disk

SUMMARY

BOOLEAN **pc_diskflush** (byte *drive_name)

DESCRIPTION

Given a valid drive specifier (A:, B:, C:…) in drive_name, flush the file allocation table and all changed files to the disk. After this call returns, the disk image is synchronized with the Rtfs internal view of the volume.

RETURNS

TRUE	The disk flush succeeded
FALSE	The operation failed consult errno

errno is set to one of the following:

0	No error
PEINVALIDDRIVEID	Drive component is invalid
An Rtfs system error	See Appendix for a description of system errors

EXAMPLE

```
#include <rtfs.h>
if (!pc_diskflush("A:"))
        printf("Flush operation failed \n");
```

- **pc_async_flush_start() is also available**
- **fs_api_commit() is also available**

pc_set_cwd

pc_set_cwd_uc

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Set current working directory.

SUMMARY

BOOLEAN **pc_set_cwd** (byte *path)

DESCRIPTION

Make path the current working directory for this task. If path contains a drive component, the current working directory is changed for that drive; otherwise the current working directory is changed for the default drive.

RETURNS

TRUE	The current working directory was changed
FALSE	The operation failed consult errno

errno is set to one of the following:

0	No error
PEINVALIDPATH	Path specified badly formed
PENOENT	Path not found
PEACCESS	Not a directory
An Rtfs system error	See Appendix for a description of system errors

EXAMPLE

```
if(!pc_set_cwd(“D:\\USR\\DATA\\FINANCE”))
   printf(“Can’t change working directory\n”);
```

pc_get_cwd
pc_get_cwd_uc

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Return the current working directory.

SUMMARY

BOOLEAN **pc_get_cwd** (byte *drive, byte *return_buffer)

DESCRIPTION

Fill return_buffer with the full path name of the current working directory for the current task for the drive specified in drive. If drive is a NULL pointer or a pointer to an empty string ("") or is an invalid drive specifier, the current working directory for the default drive is returned. In a multitasking system Rtfs maintains a current working directory for each task.

*Note: Rtfs must be configured correctly in order for each task to have its own current working directory. Please see the documentation of the routine **pc_ertfs_config()** for a complete explanation of this requirement.*

Note: return_buffer must point to enough space to hold the full path without overriding user data. The worst case possible is 260 bytes.

RETURNS

TRUE	A valid path was returned in return_buffer
FALSE	The operation failed consult errno

errno is set to one of the following:

0	No error
PEINVALIDDRIVEID	Drive component is invalid
An Rtfs system error	See Appendix for a description of system errors

EXAMPLE

```
if (pc_get_cwd("A:", pwd))
    printf ("Working dir is %s\n", pwd);
else
    printf ("Can't find working dir for A:\n");
```

pc_set_default_drive

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Set the default drive.

SUMMARY

BOOLEAN **pc_set_default_drive** (byte *drive)

DESCRIPTION

Use this function to set the current default drive that will be used when a path specifier does not contain a drive specifier.

Note: ***pc_set_default_drive()*** *does not try to access the drive, it will succeed as long as the specified drive id is between "A:" and "Z:". If the drive is not mounted the first API call to it will try to mount it. To test if a drive is present after calling* ***pc_set_default_drive()*** *you must call other APIs.* ***pc_set_cwd()*** *and* ***pc_get_cwd()*** *are convenient for this purpose.*

RETURNS

TRUE	The default drive id was set successfully.
FALSE	The operation failed consult errno

errno is set to one of the following:

0	No error
PEINVALIDDRIVEID	Driveno is incorrect

EXAMPLE

```
#include <rtfs.h>
if(!pc_set_default_drive("C:"))
   printf("Can't change working drive\n");
```

pc_get_default_drive

pc_get_default_drive_uc

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Get the default drive name and drive number.

SUMMARY

int **pc_get_default_drive** (byte *drive_name)

DESCRIPTION

This function returns the default drive. The default drive name, (A:, B:, C: etc) is returned in the drive_name buffer that is passed in.

The default drive number (0, 1, 2 ,3) is the return value of the functions.

Note: A NULL pointer may me passed in as the drive_name argument.

RETURNS

driveno	The drive number of the default drive id

errno is not set

EXAMPLE

```
int drive_no;
byte drive_name[8];
drive_no = pc_get_default_drive (drive_name);
printf("Drive name == %s, drive number == %d\n",  drive_name, drive_no);
```

pc_drno_to_drname

pc_drno_to_drname_uc

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Get the drive name associated with a drive number.

SUMMARY

void **pc_drno_to_drname** (int driveno, byte* pdrive_name)

DESCRIPTION

Use this function to get the drive name associated with the supplied drive number. This function populates the buffer pointed to by pdrive_name (A:, B:, C:.. Z:) with the drive identifier for the drive number passed in driveno (0,1,2..25).

Note: The buffer pointed to by pdrive_name must be large enough to contain the NULL terminated drive identifier. This is 3 bytes in ASCII, 6 bytes in UNICODE.

RETURNS

Nothing

EXAMPLE

```
#include <rtfs.h>
byte drive_name[6];
pc_drno_to_drname (3, drive_name);
printf("Drive name == %s\n", drive_name);  /* "C:" */
pc_drno_to_drname (25, drive_name);
printf("Drive name == %s\n", drive_name); /* "C:" */
```

pc_drname_to_drno

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Get the drive number associated with a drive name.

SUMMARY

int **pc_drname_to_drno** (byte* pdrive_name)

DESCRIPTION

Use this function to get the drive number associated with the supplied drive name. This function interprets the buffer pointed to by pdrive_name (A:, B:, C:.. Z:) and returns a drive number (0,1,2..25).

RETURNS

driveno	The drive number for driveid

This function does not set errno.

EXAMPLE

```
#include <rtfs.h>
int driveno;
driveno = pc_drname_to_drno("C:");
printf("Drive number== %d\n", driveno); /* 2 */
driveno = pc_drname_to_drno("Z:");
printf("Drive number== %d\n", driveno); /* 25 */
```

pc_diskio_info

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Return useful information about the specified drive.

SUMMARY

BOOLEAN **pc_diskio_info** (driveid, pinfo, extended)

byte *driveid	Name of a mounted volume "A," "B," etc.
DRIVE_INFO *pinfo	Drive information is placed into this structure.
BOOLEAN extended	If this argument is **TRUE** additional statistics are provided. The additional statistics are listed in the table below under the heading extended statistics. If this argument is **FALSE** the extended statistics are all set to zero. *Note: Extended statistics are calculated and thus may require additional processing time.*

DESCRIPTION

The drive capacity information provided by **pc_diskio_info()** is useful for developing certain applications and for monitoring device utilization.

Detailed description of the info structure fields. All fields are of type **dword** unless the type is specifically mentioned.	
Volume and device information. The sector size, cluster size total clusters and FAT type (12, 16 or 32) are useful things to know so they are provided.	
sector_size	The sector size in bytes (normally 512)
cluster_size	The cluster size in blocks
total_clusters	The total number of clusters in the volume
free_clusters	The current number of free clusters left in the volume.
fat_entry_size	12, 16 or 32
is_exfat	TRUE if an exFAT volume. fat_entry_size is 32.
drive_operating_policy	Drive operating policy bits may be set to control certain aspects of drive operating policy. For most applications there is no need to change them. (For more information see device_configure*_volume in the media driver callback section of the API manua).*
drive_opencounter	Number of times the drive has been mounted. This value is incremented every time the device is mounted.

	It will increment when a device change event is detected and the device is remounted.
Region buffer usage statistics. Rtfs relies on region buffers extensively. The number of region buffers required at any one time can rely on several factors such as the degree of fragmentation of the disk and the number of open files. These statistics are system wide, not drive specific, but they are provided here to allow you to determine if your region buffer configuration is correct	
Free_manager_enabled **BOOLEAN**	This field indicates if the region manager is currently enabled. *Note: free_manager_enabled will be FALSE only when the free manager is purposely disabled or Rtfs exhausts its region buffers and recovers by disabling the region manager.*
region_buffers_total	This field will always contain the number of region buffers that were provided in apicnfig.c. It will always be equal to NREGIONS.
region_buffers_free	This field contains the number of region buffers that are not currently being used.
region_buffers_low_water	This field contains the count of free region buffers at the point when the most region buffers were being used by the application. Note: Some API functions will fail and set errno to PERESOURCEREGION if they run out of region buffers, so it is a good idea to make sure you have enough of them. You should inspect region_buffers_low_water after running your application at steady state or worst case conditions to determine if NREGIONS is correct.
The elements that follow are provided only if the extended argument is **TRUE**, if the extended argument is **FALSE** they will be zero.	
free_fragments	The free clusters are in this many fragments that are separated by allocated space. *Note: free_fragments is calculated and thus require some additional processing time. If a memory based free manager is operational the free_fragments calculation is ram based only and will complete quickly. If the free manager is disabled as indicated by* free_manager_enabled is FALSE, *the disk will be scanned for free fragment, which will take significantly longer.*

RETURNS

TRUE	The operation was a success
FALSE	The operation failed consult errno

Application Level Error Return Codes

PEINVALIDPARMS	Missing or invalid parameters
PEINVALIDDRIVEID	Invalid drive specified in an argument
An Rtfs system error	See Appendix for a description

get_errno

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Get the last Rtfs assigned errno value for the calling task

SUMMARY

int **get_errno** (void)

DESCRIPTION

This function retrieves the last ERRNO value set by Ertfs for this task.

EXAMPLE

```
If (!pc_mkdir(“Test”))
    printf(“mk_dir failed: ERRNO == %d\n”, get_errno());
```

get_errno_location

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Get the current errno value and the source filename and source line number that last set the errno value for the calling task.

SUMMARY

int get_errno_location (char **filename, long *linenumber)

DESCRIPTION

If **INCLUDE_DEBUG_VERBOSE_ERRNO** is enabled **rtfs_set_errno()** prints the file name and line number that called it through the user supplied terminal IO output handler.

get_errno_location () may be called to retrieve the last filename and line number that were printed along with the last errno value. The application may retrieve these values even when the target system does not have console IO support.

This function retrieves the last ERRNO value set by Rtfs for this task and the source file name and line number that set errno.

If **INCLUDE_DEBUG_VERBOSE_ERRNO** is not enabled
    *filename and *linenumber are not set.

If **INCLUDE_DEBUG_VERBOSE_ERRNO** is enabled and the current errno is non-zero
    *filename points to the read-only file name that last called rtfs_set_errno.
    *linenumber contains the line number that last called rtfs_set_errno.

If **INCLUDE_DEBUG_VERBOSE_ERRNO** is enabled and the current errno is zero
    *filename and *linenumber are not set.

EXAMPLE

```
If (!pc_mkdir(“Test”))
{
long linenumber = 0;
char *filename = “unknown”;
int errno;
   errno = get_errno_location (&filename, &linenumber);
   printf(“mk_dir failed: ERRNO == %d, FILE == %s, LINE = %d\n”,
                 errno,  filename, linenumber);
}
```

pc_gfirst

pc_gfirst_uc

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Return the first entry in a directory.

SUMMARY

BOOLEAN **pc_gfirst** (DSTAT *statobj, byte *pattern)

DESCRIPTION

Given a pattern which contains both a path specifier and a search pattern, fill in the structure at statobj with information about the file and set up internal parts of statobj to supply appropriate information for calls to **pc_gnext()**.

Examples of patterns are:
"D:\USR\RELEASE\NETWORK\*.C"
"BIN\UU*.*"
"MEMO_?.*"
"*.*"

Note: If **pc_gfirst()** *succeeds you may call* **pc_gnext()** *to get the next directory entry that matches the criteria. When you are done you must call* **pc_gdone()** *to free internal resources. If* **pc_gfirst()** *does not succeed it is not necessary to call* **pc_gdone()**.

RETURNS

TRUE	The operation was a success and a match was found
FALSE	Operation failed or no match found. consult errno

errno is set to one of the following:

0	No error
PEINVALIDDRIVEID	Drive component is invalid
PEINVALIDPATH	Path specified badly formed
PENOENT	Not found, no match
An Rtfs system error	See Appendix for a description of system errors

SEE ALSO:

pc_gnext(), pc_gdone(), and **pc_seedir()** in appcmdsh.c

pc_gnext

pc_gnext_uc

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Return next entry in a directory.

SUMMARY

BOOLEAN **pc_gnext** (DSTAT *statobj)

DESCRIPTION

Continue with the directory scan started by a call to **pc_gfirst()**.

RETURNS

TRUE	The operation was a success and a match was found
FALSE	The operation failed or no match found. consult errno

errno is set to one of the following:

0	No error
PEINVALIDPARMS	statobj argument is not valid
PENOENT	Not found, no match (normal termination of scan)
PEINVALIDDRIVEID	Drive was removed or closed since **pc_gfirst()** call.
An Rtfs system error	See Appendix for a description of system errors

EXAMPLE

```
#include <rtfs.h>
if (pc_gfirst(&statobj,"A:\\dev\\*.c"))
{
    do
    {
        /* print file name, extension and size */
        printf("%-8s.%-3s %7ld \n",statobj.fname,
        statobj.fext,statobj.fsize);
    }
    while (pc_gnext(&statobj));
    /* Call gdone to free up internal resources */
    pc_gdone(&statobj);
}
```

pc_glast

pc_glast_uc

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Return the last entry in a directory.

SUMMARY

BOOLEAN **pc_glast** (DSTAT *statobj, byte *pattern)

DESCRIPTION

Pc_glast behaves similarly to pc_gfirst except it finds the last entry in a directory to match a pattern. Given a pattern which contains both a path specifier and a search pattern, fill in the structure at statobj with information about the file and set up internal parts of statobj to supply appropriate information for calls to **pc_prev().**

Examples of patterns are:
“D:\USR\RELEASE\NETWORK\*.C”
“BIN\UU*.*”
“MEMO_?.*”
“*.*”

Note: If **pc_glast()** *succeeds you may call* **pc_gprev()** *to get the next directory entry that matches the criteria. When you are done you must call* **pc_gdone()** *to free internal resources. If* **pc_glast()** *does not succeed it is not necessary to call* **pc_gdone()***.*

RETURNS

TRUE	The operation was a success and a match was found
FALSE	Operation failed or no match found. consult errno

errno is set to one of the following:

0	No error
PEINVALIDDRIVEID	Drive component is invalid
PEINVALIDPATH	Path specified badly formed
PENOENT	Not found, no match
An Rtfs system error	See Appendix for a description of system errors

SEE ALSO:

pc_gprev(), pc_gdone(), and **pc_seedir()** in appcmdsh.c

pc_gprev

pc_gprev_uc

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Return previous entry in a directory.

SUMMARY

BOOLEAN **pc_gprev** (DSTAT *statobj)

DESCRIPTION

Continue with the directory scan started by a call to **pc_glast()**.

RETURNS

TRUE	The operation was a success and a match was found
FALSE	The operation failed or no match found. consult errno

errno is set to one of the following:

0	No error
PEINVALIDPARMS	statobj argument is not valid
PENOENT	Not found, no match (normal termination of scan)
PEINVALIDDRIVEID	Drive was removed or closed since **pc_gfirst()** call.
An Rtfs system error	See Appendix for a description of system errors

EXAMPLE

```
#include <rtfs.h>
if (pc_glast(&statobj,”A:\\dev\\*.c”))
{
    do
    {
        /* print file name, extension and size */
        printf(“%-8s.%-3s %7ld \n”,statobj.fname,
        statobj.fext,statobj.fsize);
    }
    while (pc_gprev(&statobj));
    /* Call gdone to free up internal resources */
    pc_gdone(&statobj);
}
```

pc_gdone

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Free directory scan resources originally allocated by **pc_first() or pc_gprev()**.

SUMMARY

void **pc_gdone** (DSTAT *statobj)

DESCRIPTION

Given a pointer to a DSTAT structure that was set up by a call to **pc_gfirst()** free internal elements used by statobj.

Note: You must call this function after you have finished calling **pc_gfirst()** *and* **pc_gnext()** *or calling* **pc_gprev()** *and* **pc_gprev()** *or a memory leak will occur.*

RETURNS

Nothing

Does not set errno.

EXAMPLE

See **pc_gnext()**

pc_gread

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Read data from a directory scan result.

SUMMARY

```
BOOLEAN pc_gread (DSTAT *statobj,
                  int blocks_to_read,
                  byte *buffer,
                   int *blocks_read)
```

DESCRIPTION

Read data from the **DSTAT** structure returned from a successful call to **pc_gfirst()** or **pc_gnext().** This function can be used to implement efficient file enumeration procedures for media player devices by eliminating the need to open files to read header information.

Note: This function is intended for reading file header information but the ability to read blocks from a subdirectory is also provided.

*Note: This function is block oriented and ignores the directory entry's file size attribute, so if (blocks_to_read*512) is larger than the file's size, it will read up to the last cluster boundary.*

statobj	DSTAT structure previously filled by **pc_gfirst()** or **pc_gnext()**
blocks_to_read	The number of blocks you would like to read from the beginning of the file or subdirectory.
buffer	Buffer that pc_gread should read data to. *Note: buffer must be at least large enough to hold blocks_to_read sectors.This is typically 512 * blocks_to_read bytes, but the buffer must be larger if the media has a larger sector size.*
blocks_read	Pointer to an integer that returns the number of blocks that were successfully transferred to the buffer. *Note: If the file or subdirectory contains less than to blocks_to_read blocks, data will be read up to the boundary of the last cluster in the file.*

RETURNS

TRUE	The operation was a success. Blocks_read contains the number of blocks transferred to buffer.
FALSE	The operation failed consult errno

errno is set to one of the following:

0	No error
PEINVALIDDRIVEID	Drive was removed or closed since **pc_gfirst()** call
PEINVALIDPARMS	Invalid arguments
An Rtfs system error	See Appendix for a description of system errors

pc_get_attributes

pc_get_attributes_uc

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Get File Attributes of the named file

SUMMARY

BOOLEAN **pc_get_attributes**(byte *path, byte *p_return);

DESCRIPTION

Given a file or directory name, return the directory entry attributes associated with the entry. One or more of the following values will be or'ed together:

BIT	Mnemonic
0	ARDONLY
1	AHIDDEN
2	ASYSTEM
3	AVOLUME
4	ADIRENT
5	ARCHIV

RETURNS

TRUE	The operation was a success
FALSE	The operation failed consult errno

errno is set to one of the following:

0	No error
PENOENT	Path not found
An Rtfs system error	See Appendix for a description of system errors

pc_set_attributes

pc_set_attributes_uc

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Set File Attributes

SUMMARY

BOOLEAN **pc_set_attributes** (byte *path, byte attributes)

DESCRIPTION

Given a file or directory name set the directory entry attributes associated with the entry. One or more of the following values may be or'ed together.

BIT	Mnemonic
0	ARDONLY
1	AHIDDEN
2	ASYSTEM
3	ARCHIVE
4	ADIRENT
5	ARCHIVE

RETURNS

TRUE	The operation was a success
FALSE	The operation failed consult errno

errno is set to one of the following:

0	No error
PEINVALIDPARMS	Attribute argument is invalid
PEINVALIDDRIVEID	Drive component is invalid
PEINVALIDPATH	Path specified badly formed
PENOENT	Path not found
PEACCESS	Object is read only
An Rtfs system error	See Appendix for a description of system errors

EXAMPLE

```
#include <rtfs.h>
byte attribs;
if (pc_get_attributes("A:\\COMMAND.COM", &attribs)
{
    attribs |= ARDONLY|AHIDDEN
    pc_set_attributes("A:\\COMMAND.COM", attribs);
}
```

pc_isdir

pc_isdir_uc

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Test if a path is a directory.

SUMMARY

BOOLEAN **pc_isdir** (byte *path)

DESCRIPTION

This is a simple routine that opens a path and checks if it is a directory, then closes the path. The same functionality can be had by calling **pc_gfirst()** and testing the DSTAT structure.

RETURNS

TRUE	The operation was a success and it is a directory
FALSE	The operation failed consult errno

errno is set to one of the following:

0	No error
PENOENT	Path not found
An Rtfs system error	See Appendix for a description of system errors

pc_isvol

pc_isvol_uc

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Test if a path name is a volume label.

SUMMARY

BOOLEAN **pc_isvol**(byte *path)

DESCRIPTION

Tests to see if a path specification is a volume label specifier.

RETURNS

TRUE	The operation was a success and it is a volume
FALSE	The operation failed consult errno

errno is set to one of the following:

0	No error
PENOENT	Path not found
An Rtfs system error	See Appendix for a description of system errors

pc_stat

pc_stat_uc

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Return properties of a named file or directory.

SUMMARY

int **pc_stat** (byte *name,ERTFS_STAT *pstat)

DESCRIPTION

This routine searches for the file or directory provided in the first argument. If found, it fills in the stat structure as described here:

The ERTFS_STAT structure:

st_dev	the entry's drive number
st_mode	Contains one or more of the following bits: **S_IFMT** - type of file mask **S_IFCHR** - char special (unused) **S_IFDIR** - directory **S_IFBLK** - block special (unused) **S_IFREG** - regular (a "file") **S_IWRITE** - Write permitted **S_IREAD** - Read permitted
st_rdev	the entry's drive number
st_size	file size
st_atime	Last modified date in DATESTR format
st_mtime	Last modified date in DATESTR format
st_ctime	Last modified date in DATESTR format
t_blksize	optimal blocksize for I/O (cluster size)
t_blocks	blocks allocated for file
The following fields are extensions to the standard stat structure	
fattributes	The DOS attributes. This is non-standard but supplied if you wish to look at them.
st_size_hi	If the file is an exFAT file, the high 32 bits of the file size

NOTE: ERTFS_STAT structure is equivalent to the STAT structure available with most posix like run time environments. Unfortunately certain run time environments like uITRON also use a structure named STAT so in order to avoid namespace collisions Rtfs uses the proprietary name ERTFS_STAT. If you are porting an application that uses STAT you may put the following preprocessor macro in rtfs.H just below the declaration of ERTFS_STAT: #define STAT ERTFS_STAT

RETURNS

0	The operation was a success
-1	The operation failed consult errno

errno is set to one of the following:

0	No error
PEINVALIDDRIVEID	Drive component is invalid
PENOENT	File or directory not found
An Rtfs system error	See Appendix for a description

EXAMPLE

```
#include <rtfs.h>
struct ERTFS_stat st;

 if (pc_stat("A:\\MYFILE.TXT", &st)==0)
{
        printf("DRIVENO: %d\n", st.st_dev);
        printf("SIZE: %d\n" st.st_size);    /* in bytes */
        printf("Month: %d\n", (st.st_atime.date >> 5 ) & 0xf,);
        printf("Day: %d\n", (st.st_atime.date ) & 0x1f,);
        printf("Year: %d\n", (st.st_atime.date >> 9 ) & 0xf,);
        printf("Hour: %d\n", (st.st_atime.time >> 11) & 0x1f);
        printf("Minute: %d\n", (st.st_atime.time >> 5) & 0x3f);
        printf("OPT BLOCK SIZE:%d\n",
                                st.st_blksize,st.st_blocks);
        printf("FILE size (BLOCKS): %d\n",  st.st_blocks);
                printf("MODE BITS :");

        if (st.st_mode&S_IFDIR)
                printf("S_IFDIR|");
        if (st.st_mode&S_IFREG)
                printf("S_IFREG|");
        if (st.st_mode&S_IWRITE)
                printf("S_IWRITE|");
        if (st.st_mode&S_IREAD)
                printf("S_IREAD\n");
                printf("\n");
```

pc_blocks_free

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Return disk free space statistics

SUMMARY

```
BOOLEAN pc_block_free (byte *drive,
               dword *total blocks,
               dword *free blocks);
```

DESCRIPTION

Given a drive ID, return the total number of blocks on the drive in the dword pointed to by total_blocks, return the number of blocks free in the dword pointed to by free_blocks.

RETURNS

TRUE	The operation was a success
FALSE	The operation failed consult errno

errno is set to one of the following:

0	No error
PEINVALIDDRIVEID	Driveno is incorrect
An Rtfs system error	See Appendix for a description of system errors

EXAMPLE

```
#include <rtfs.h>
  If (pc_blocks_free ("A:", & total_blocks, & free_blocks))
   printf ("%d blocks free out of %d blocks total  \n:",
          free_blocks, total_blocks);
```

pc_mkdir

pc_mkdir_uc

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Create a subdirectory.

SUMMARY

BOOLEAN **pc_mkdir** (byte *path)

DESCRIPTION

Create a subdirectory in the path specified by path. Fails if a file or directory of the same name already exists or if the directory component (if there is one) of path is not found.

RETURNS

TRUE	The subdirectory was created
FALSE	The operation failed consult errno

errno is set to one of the following:

0	No error
PEINVALIDDRIVEID	Drive component is invalid
PEINVALIDPATH	Path specified badly formed.
PENOENT	Path to new directory not found
PEEXIST	File or directory of this name already exists
An Rtfs system error	See Appendix for a description of system errors

EXAMPLE

```
pc_mkdir("\\USR\\LIB\\HEADER\\SYS");
```

pc_rmdir

pc_rmdir_uc

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Delete a directory

SUMMARY

BOOLEAN **pc_rmdir** (byte *path)

DESCRIPTION

Delete the directory specified in path. Fails if path is not a directory, is read only or is not empty.

RETURNS

TRUE	The directory was successfully removed.
FALSE	The operation failed consult errno

errno is set to one of the following:

0	No error
PEINVALIDDRIVEID	Drive component is invalid
PEINVALIDPATH	Path specified badly formed.
PENOENT	Directory not found
PEACCESS	Directory is in use or is read only
An Rtfs system error	See Appendix for a description of system errors

EXAMPLE

```
#include <rtfs.h>
if (!pc_rmdir("D:\\USR\\TEMP")
  printf("Can't delete directory\n");
```

pc_mv

pc_mv_uc

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Rename files and directories

SUMMARY

BOOLEAN **pc_mv** (char *oldpath, char *newpath)

DESCRIPTION

Moves the file or subdirectory named oldpath to the new name specified in newpath. oldpath and newpath must be on the same drive but they may be in different sub-directories. Both names must be fully qualified (see examples). Fails if newpath is invalid or already exists or if oldpath is not found.

RETURNS

TRUE	The file or subdirectory was moved
FALSE	The operation failed consult errno

errno is set to one of the following:

0	No error
PEINVALIDDRIVEID	Drive component is invalid or they are not the same
PEINVALIDPATH	Path specified by old_name or new_name is badly formed.
PEACCESS	File or directory in use, or old_name is read only
PEEXIST	new_name already exists
An Rtfs system error	See Appendix for a description of system errors

EXAMPLE

```
#include <rtfs.h>

if (!pc_mv("\\USR\\TXT\\LETTER.TXT", "LETTER.OLD"))
    printf("Can't move the file\n");


if (!pc_mv("\\employeefolders\\joe",  "\\ex-employeefolders\\joe")
  printf("Can't move the subdirectory \n");
```

pc_unlink

pc_unlink_uc

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Delete a file.

SUMMARY

BOOLEAN **pc_unlink** (byte *path)

DESCRIPTION

Delete the filename pointed to by path. Fail if it is not a simple file, if it is open, if it does not exist, or it is read only.

RETURNS

TRUE	It successfully deleted the file.
FALSE	The operation failed consult errno

errno is set to one of the following:

0	No error
PEINVALIDDRIVEID	Drive component is invalid
PEINVALIDPATH	Path specified badly formed.
PENOENT	Can't find file to delete
PEACCESS	File in use, is read only or is not a simple file.
An Rtfs system error	See Appendix for a description of system errors

EXAMPLE

```
if (!pc_unlink("B:\\USR\\TEMP\\TMP001.PRN") )
    printf("Can't delete file \n")
```

pc_async_unlink_start() is also available

Basic File IO API

po_open

po_open_uc

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Open a file.

SUMMARY

int **po_open** (byte *path, word flag, word mode)

DESCRIPTION

Open the file for access as specified in flag. If creating use mode to set the access permissions.

Flag values are:	
PO_APPEND	All writes will be appended to the file
PO_BINARY	Ignored
PO_TEXT	Ignored
PO_RDONLY	Open for read only
PO_RDWR	Read/write access allowed
PO_WRONLY	Open for write only
PO_CREAT	Create the file if it does not exist
PO_EXCL	If flag has (**PO_CREAT|PO_EXCL**) and the file already exists, fail and set errno to **EEXIST**
PO_TRUNC	Truncate the file if it already exists
PO_BUFFERED	If this is set, reads and writes of less than 512 bytes and operations that do not start or end on block boundaries are buffered. The buffer is flushed when **po_close()** is called, when **po_flush()** is called or if a buffered IO request is made to a different block number. Using the **PO_BUFFERED** flag increases performance of

	applications performing reads and writes of small or un aligned data buffers.
PO_AFLUSH	Enable auto flush mode. The file is flushed automatically by **po_write()** whenever the file length changes.
PO_NOSHAREANY	Fail if already open, fail if another open is tried
PO_NOSHAREWRITE	Fail if already open for write and fail if another open for write is tried

Mode values are:	
PS_IWRITE	Write permitted
PS_IREAD	Read permitted (Always true anyway)

<u>RETURNS</u>

>= 0	to be used as a file descriptor for calling **po_read()**, **po_write()**, **po_lseek()**, **po_flush()**, **po_truncate()**, and **po_close()**
-1	The operation failed consult errno

errno is set to one of the following:

0	No error
PENOENT	Not creating a file and file not found
PEMFILE	Out of file descriptors
PEINVALIDPATH	Invalid pathname
PENOSPC	No space left on disk to create the file
PEACCES	Is a directory or opening a read only file for write
PESHARE	Sharing violation on file opened in exclusive mode
PEEXIST	Opening for exclusive create but file already exists
An Rtfs system error	See Appendix for a description of system errors

<u>EXAMPLE</u>

```
#include <rtfs.h>
int fd;
if(fd=po_open(“\\USR\\MYFILE”,(PO_CREAT|PO_EXCL|PO_WRONLY)
                        ,P S_IWRITE)<0))
        printf(“Can’t create file error:%i\n” ,get_errno())
```

pc_efilio_open is also available

po_close

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Close a file that was opened with po_open

SUMMARY

int **po_close** (int fd)

DESCRIPTION

Close the file and update the disk by flushing the directory entry and file allocation table. Free all core associated with fd.

RETURNS

0	The operation was a success
-1	The operation failed consult errno

errno is set to one of the following:

0	No error
PEBADF	Invalid file descriptor
PECLOSED	Invalid file descriptor because a removal or media failure asynchronously closed the volume. **po_close()** must be called to clear this condition.
An Rtfs system error	See Appendix for a description of system errors

SEE ALSO

po_flush

EXAMPLE

```
#include <rtfs.h>
if (po_close(fd) < 0)
   printf("Error closing file:%i\n",rtfs_get_errno());
```

pc_efilio_close is also available

po_read

Basic	x	ProPlus	X
Pro	x	ProPlus DVR	X

FUNCTION

Read from a file.

SUMMARY

int **po_read** (int fd, byte *buf, int count)

DESCRIPTION

Attempt to read count bytes from the current file pointer of file at fd and place the data in buf. The file pointer is updated.

Note: If buf is 0 (the null pointer) then the operation is performed identically to a normal read except no data transfers are performed. This may be used to quickly advance the file pointer.

RETURNS

>= 0	The actual number of bytes
-1	The operation failed consult errno

errno is set to one of the following:

0	No error
PEBADF	Invalid file descriptor
PECLOSED	Invalid file descriptor because a removal or media failure asynchronously closed the volume. **po_close** must be called to clear this condition.
PEIOERRORREAD	Read error
An Rtfs system error	See Appendix for a description of system errors

EXAMPLE

```
int fd;
int fd2;
fd = po_open("FROM.FIL",PO_RDONLY,0);
fd2 =po_open("TO.FIL",PO_CREAT|PO_WRONLY,PS_IWRITE)
if (fd >= 0 && fd2 >= 0)
  while (po_read(fd, buff, 512) ==512)
      po_write(fd2, buff, 512);
```

pc_efilio_read is also available

po_write

Basic	x	ProPlus	X
Pro	x	ProPlus DVR	X

FUNCTION

Write to a file.

SUMMARY

int **po_write** (int fd, byte *buf, int count)

DESCRIPTION

Attempt to write count bytes from buf to the current file pointer of file at fd. The file pointer is updated.
Note: If buf is 0 (the null pointer) then the operation is performed identically to a normal write, the file pointer is moved and as the file cluster chain is extended if needed but no data is transferred. This may be used to quickly expand a file or to move the file pointer.

RETURNS

>= 0	The actual number of bytes written
-1	The operation failed consult errno

errno is set to one of the following:

0	No error
PEBADF	Invalid file descriptor
PECLOSED	Invalid file descriptor because a removal or media failure asynchronously closed the volume. **po_close** must be called to clear this condition.
PEACCES	File is read only
PEIOERRORWRITE	Error performing write
PEIOERRORREAD	Error reading block for merge and write
PENOSPC	Disk full
An Rtfs system error	See Appendix for a description of system errors

EXAMPLE

```
int fd, fd2;

fd = po_open(“FROM.FIL”,PO_RDONLY,0);
fd2 =po_open(“TO.FIL”,PO_CREAT|PO_WRONLY,PS_IWRITE)
if (fd >= 0 && fd2 >= 0)
        while (po_read(fd, buff, 512) ==512)
            po_write(fd2, buff, 512);
```

pc_efilio_write is also available

po_lseek64

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

64 bit move file pointer

SUMMARY

ddword **po_lseek64** (int fd, ddword offset, int origin)

DESCRIPTION

Move the file pointer offset bytes from the origin described by origin. Origin may have the following values:

PSEEK_SET	Seek from beginning of file
PSEEK_CUR	Seek from the current file pointer
PSEEK_CUR_NEG	Seek backward from the current file pointer
PSEEK_END	Seek from end of file

Attempting to seek beyond end of file puts the file pointer one byte past end of file. Seeking zero bytes from origin **PSEEK_END** returns the file length.

Note: for exFAT true 64 bit seeks are supported. For FAT, po_lseek64() operates on the lower 32 bits but still reports error as **(**0xffffffffffffffff).

RETURNS

M64SET32(0xffffffff, 0xffffffff) or (0xffffffffffffffff)	The operation failed consult errno.
(!0xffffffffffffffff)	The new offset

errno is set to one of the following:

0	No error
PEBADF	Invalid file descriptor
PECLOSED	Invalid file descriptor because a removal or media failure asynchronously closed the volume. **po_close()** must be called to clear this condition.
PEINVALIDPARMS	Attempt to seek past EOF or to a negative offset
PEINVALIDCLUSTER	Files contains a bad cluster chain
An Rtfs system error	See Appendix for a description of system errors

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Move file pointer, unsigned

SUMMARY

```
BOOLEAN po_ulseek (int fd, unsigned long offset,
                   unsigned long *pnew_offset, int origin)
```

DESCRIPTION

Move the file pointer offset bytes from the origin described by origin. origin may have the following values:

PSEEK_SET	Seek from beginning of file
PSEEK_CUR	Seek from the current file pointer
PSEEK_CUR_NEG	Seek backward from the current file pointer
PSEEK_END	Seek from end of file

The new file pointer is returned in *pnew_offset

Attempting to seek beyond end of file puts the file pointer one byte past end of file. Seeking zero bytes from **PSEEK_END** returns the file length.

RETURNS

TRUE	The operation succeeded
FALSE	The operation failed consult errno

errno is set to one of the following:

0	No error
PEBADF	Invalid file descriptor
PECLOSED	Invalid file descriptor because a removal or media failure asynchronously closed the volume. **po_close()** must be called to clear this condition.
PEINVALIDPARMS	Attempt to seek past EOF or to a negative offset
PEINVALIDCLUSTER	Files contains a bad cluster chain
An Rtfs system error	See Appendix for a description of system errors

pc_efilio_lseek is also available

po_chsize

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Truncate or extend an open file.

SUMMARY

int **po_chsize** (int fd, unsigned long newfilesize)

DESCRIPTION

Given a file handle and a new file size, either extend the file or truncate it. If the current file pointer is still within the range of the file, it is not moved, otherwise it is moved to the end of file. This function uses other API calls and does not set errno itself.

RETURNS

0	The operation succeeded
-1	The operation failed consult errno

errno is set to one of the following:

0	No error
PEBADF	Invalid file descriptor
PECLOSED	Invalid file descriptor because a removal or media failure asynchronously closed the volume. **po_close()** must be called to clear this condition.
PEACCES	File is read only
PEINVALIDPARMS	Invalid or inconsistent arguments
An Rtfs system error	See Appendix for a description of system errors

pc_efilio_chsize() is also available

po_flush

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Flush a file to disk.

SUMMARY

BOOLEAN **po_flush** (int fd)

DESCRIPTION

Flush file buffers, flush directory entry changes to disk, and flush the FAT. After this call completes, the on disk view of the file is completely consistent with the in memory view. It is a good idea to call this function periodically if a file is being extended. If failsafe is not running and a file is not flushed or closed when a power down occurs, the file size will be wrong on disk and the FAT chains will be lost.

RETURNS

TRUE	The flush was successful
FALSE	The operation failed consult errno

errno is set to one of the following:

0	No error
PEBADF	Invalid file descriptor
PECLOSED	Invalid file descriptor because a removal or media failure asynchronously closed the volume. **po_close()** must be called to clear this condition.
PEACCES	File is read only
An Rtfs system error	See Appendix for a description of system errors Directory is in use or is read only

SEE ALSO

pc_dskflush()

EXAMPLE

```
#include <rtfs.h>
if (po_flush(fd) < 0)
   printf("Error flushing file:%i\n",rtfs_get_errno());
```

pc_efilio_chsize is also available

pc_fstat

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Return properties of a file associated with a file descriptor.

SUMMARY

int **pc_fstat** (int file_descriptor, ERTFS_STAT *pstat)

DESCRIPTION

For the provided file descriptor this routine fills in the stat structure as described here:

The ERTFS_STAT structure:

st_dev	the entry's drive number
st_mode	Contains one or more of the following bits: **S_IFMT** - type of file mask **S_IFCHR** - char special (unused) **S_IFDIR** - directory **S_IFBLK** - block special (unused) **S_IFREG** - regular (a "file") **S_IWRITE** - Write permitted **S_IREAD** - Read permitted
st_rdev	the entry's drive number
st_size	file size
st_atime	Last modified date in DATESTR format
st_mtime	Last modified date in DATESTR format
st_ctime	Last modified date in DATESTR format
t_blksize	optimal blocksize for I/O (cluster size)
t_blocks	blocks allocated for file
The following fields are extensions to the standard stat structure	
fattributes	The DOS attributes. This is non-standard but supplied if you wish to look at them.
st_size_hi	If the file is an exFAT file, the high 32 bits of the file size

NOTE: ERTFS_STAT structure is equivalent to the STAT structure available with most posix like run time environments. Certain run time environments like uITRON also use a structure named STAT so to avoid namespace collisions Rtfs uses the proprietary name ERTFS_STAT

RETURNS

0	The operation succeeded
-1	The operation failed consult errno

errno is set to one of the following:

0	No error
PEBADF	Invalid file descriptor
PECLOSED	Invalid file descriptor because a removal or media failure asynchronously closed the volume. **po_close()** must be called to clear this condition.

EXAMPLE

```
#include <rtfs.h>
struct ERTFS_stat st;
int fd;
fd = po_open(“A:\\MYFILE.TXT”,(PO_BINARY|PO_RDONLY),0);
if (pc_fstat(fd, &st)==0)
{
{
        printf(“DRIVENO: %d\n”, st.st_dev);
        printf(“SIZE: %d\n” st.st_size);    /* in bytes */
        printf(“Month: %d\n”, (st.st_atime.date >> 5 ) & 0xf,);
        printf(“Day: %d\n”, (st.st_atime.date ) & 0x1f,);
        printf(“Year: %d\n”, (st.st_atime.date >> 9 ) & 0xf,);
        printf(“Hour: %d\n”, (st.st_atime.time >> 11) & 0x1f);
        printf(“Minute: %d\n”, (st.st_atime.time >> 5) & 0x3f);
        printf(“OPT BLOCK SIZE:%d\n”,
                                    st.st_blksize,st.st_blocks);
        printf(“FILE size (BLOCKS): %d\n”,  st.st_blocks);
printf(“MODE BITS :”);

if (st.st_mode&S_IFDIR)
        printf(“S_IFDIR|”);
if (st.st_mode&S_IFREG)
printf(“S_IFREG|”);
if (st.st_mode&S_IWRITE)
        printf(“S_IWRITE|”);
if (st.st_mode&S_IREAD)
        printf(“S_IREAD\n”);
printf(“\n”);
}
}
```

pc_efilio_fstat() is also available

Format and partition management API

pc_get_media_parms

pc_get_media_parms_uc

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Get device geometry for a named device.

SUMMARY

```
BOOLEAN pc_get_media_parms (
                byte *path,
                PDEV_GEOMETRY pgeometry)
```

DESCRIPTION

Query the drive's associated device driver for a description of the installed media. This information is used by the command shell when performing the FDISK command to prompt the user for the sizes required for each partition. **pc_partition_media()** and **pc_format_volume()** require geometry information but they call the device driver themselves to retrieve it.

Note: The floppy device driver uses a "back door" to communicate with the format routine through the geometry structure. This allows us to not have floppy specific code in the format routine but still use the exact format parameters that DOS uses when it formats a floppy.

See the following definition of the geometry structure:

typedef struct dev_geometry {

```
int   bytespsector;              - 0 or 512 for 512 byte sectors, 1024, 2048, 4096
int dev_geometry_heads;       - Must be < 256
int dev_geometry_cylinders;  - Must be < 1024
int dev_geometry_secptrack; - Must be < 64
dword dev_geometry_lbas;     - For oversized media that
                                supports logical block ad dressing. If this is non-zero
                                dev_geometry_cylinders
                                is ignored but dev_geometry_heads and
                                dev_geometry_secptrack must still be valid.
BOOLEAN fmt_parms_valid;    - If the device I/O control call
                                sets this TRUE, then it tells the
                                applications layer that these
                                format parameters should be used. This is a way to
                                format floppy disks exactly as they are
                                formatted by DOS.
```

FMTPARMS fmt;
} DEV_GEOMETRY;

RETURNS

TRUE	The operation succeeded
FALSE	The operation failed consult errno

errno is set to one of the following:

0	No error
PEINVALIDDRIVEID	Drive component is invalid
PEDEVICEFAILURE	Device driver get device geometry request failed
PEINVALIDPARMS	Device driver returned bad values

SEE ALSO

pc_format_media(), **pc_partition_media()**,
pc_format_volume()

EXAMPLE

Note: This routine is designed to work in a specific context. See the source code of appcmdsh.c *and the documentation for* **pc_format_volume()** *for example usage.*

pc_partition_media

pc_partition_media_uc

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Partition a disk

SUMMARY

BOOLEAN **pc_partition_media** (byte *path, struct mbr_specification *pmbrspec)

DESCRIPTION

Write a partition table onto the disk at path, according to the specification provided in pmbrspec.

Note: If the underlying device driver is dynamic, it will provide dynamic partitioning instructions and 0 may be passed for pmbrspec, since it is ignored.

Note: If extended partitions are desired then one additional mbr_specification structure is required per virtual volume in the extended partition. The specifications must be provided in a contiguous array pointed to by pmbrspec.

The MBR specification structure

Typically one specification structure is provided. This is used to initialize the primary boot record.

```
struct mbr_specification {
   int    device_mbr_count;
   dword  mbr_sector_location;
   struct mbr_entry_specification entry_specifications[4];
};
```

device_mbr_count	Only used in the first specification. This must contain 1 if there is only one partition table. If extended partitions are required this must be 1 plus the number of EBR (extended boot record) specifications to follow.
mbr_sector_location	Location of this primary or extended boot record. Will contain 0 for the primary MBR. For extended boot records this will contain the absolute sector address where the record will reside.
entry_specifications[4]	Contains four partition table entries. If an entry is not used it should be zero filled.

```
struct mbr_entry_specification {
   dword partition_start;
   dword partition_size;
   byte  partition_type;
   byte  partition_boot

};
```

partition_start	Sector number where the volume BPB resides.
partition_size	Number of sectors in the partition.
partition_type	0x0c; - Fat 32 0x06; - Huge Fat 16 0x04; - Fat 16 0x01; - Fat 12
partition_boot	use 0x80 for bootable, 0x00 otherwise (ignored by Rtfs)

If the device driver is dynamically providing the specifications, it will be called once for each specification it needs, passing the index number as an argument.

Note: The source of appcmdshformat.c *contains source code with example usage, including how to create extended partitions.* appcmdshformat.c *is intentionally partitioned to be easy to cut and paste sections, excluding user interface code into your own project.*

RETURNS

TRUE	The operation succeeded
FALSE	The operation failed consult errno

errno is set to one of the following:

0	No error
PEINVALIDDRIVEID	Drive component is invalid
PEINVALIDPARMS	Inconsistent or missing parameters
PEIOERRORWRITE	Error writing partition table
An Rtfs system error	See Appendix for a description of system errors

SEE ALSO

pc_get_media_parms(), **pc_format_media()**, **pc_format_volume()**

pc_format_media

pc_format_media_uc

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Perform a device level format

Note: The source of appcmdshformat.c *contains source code with example usage.* appcmdshformat.c *is intentionally partitioned to be easy to cut and paste sections, excluding user interface code into your own project.*

Note: Format media requests are passed to the device driver which to format the device. Most devices do not require formatting. If the devices supported by your application never require formatting you may omit this call. Alternatively you may call pc_format_media which will have no effect. Devices for which device format may be necessary are floppy disks, and some flash drivers that may wish to erase sectors and possibly internal formatting hidden FTL control block.

SUMMARY

BOOLEAN **pc_format_media** (byte *path)

path is the device's drive id (A:, B: etc).

DESCRIPTION

This routine performs a device level format on the specified device.

RETURNS

TRUE	The operation succeeded
FALSE	The operation failed consult errno

errno is set to one of the following:

0	No error
PEINVALIDDRIVEID	Drive component is invalid
PEDEVICEFAILURE	Device driver format request failed
PEDYNAMIC	A dynamic device driver is present but it returned invalid parameters

SEE ALSO

pc_get_media_parms(), **pc_partition_media()**, **pc_format_volume()**

pc_format_volume

pc_format_volume_uc

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Perform a volume format

SUMMARY

BOOLEAN **pc_format_volume** (byte *path)

DESCRIPTION

This routine formats the volume referred to by drive letter. If the device is partitioned, the partition table is read and the volume within the partition is formatted. If it is a non-partitioned device, the device is formatted according to the geometry parameters returned by the device driver

Note: The source of appcmdshformat.c *contains source code with example usage.* **appcmdshformat.c** *is intentionally partitioned to be easy to cut and paste sections, excluding user interface code into your own project.*

RETURNS

TRUE	The operation succeeded
FALSE	The operation failed consult errno

errno is set to one of the following:

0	No error
PEINVALIDDRIVEID	Drive component is invalid
PEIOERRORREADMBR	Partitioned device. I/O error reading
PEINVALIDMBR	Partitioned device has no master boot record
PEINVALIDMBROFFSET	Requested partition has no entry in master boot record
PEINVALIDPARMS	Inconsistent or missing parameters
PEIOERRORWRITE	Error writing during format
PEDYNAMIC	A dynamic device driver is present but it returned invlid parameters
An Rtfs system error	See Appendix for a description of system errors

SEE ALSO

pc_get_media_parms(), **pc_partition_media()**, **pc_format_media()**

EXAMPLE

See the routine **doformat()** in appcmdshformat.c.

pc_format_volume_ex

pc_format_volume_ex_uc

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Perform a volume format

SUMMARY

BOOLEAN **pc_format_volume_ex** (byte *path, struct rtfsfmtparmsex *pfmtparms)

DESCRIPTION

This routine formats the volume referred to by drive letter. If the device is partitioned, the partition table is read and the volume within the partition is formatted. If it is a non-partitioned device, the device is formatted according to the geometry parameters returned by the device driver

struct rtfsfmtparmsex {

BOOLEAN	scrub_volume
unsigned char	bits_per_cluster
unsigned short	numroot
unsigned char	numfats
unsigned char	secpalloc
unsigned short	secreserved

};

struct rtfsfmtparmsex	
scrub_volume	If TRUE erase the section of media containing the volume. For NAND the device driver erase routine will be called, for other devices all sectors will be written with zeroes.
bits_per_cluster	Select file system type, 12, 16, 32 for FAT12, FAT16, FAT32 respectively
numroot	Number of root directory entries to reserve. Normally 512 for FAT12 and FAT16, must be 0 for FAT32.
numfats	Number of FATS on the disk, Must be 2 if using Failsafe
secpalloc	Sectors per cluster
secreserved	Number of reserved sectors. usually 32 for FAT32, 1 for not FAT32

Note: The source of appcmdshformat.c *contains source code with example usage.* appcmdshformat.c *is intentionally partitioned to be easy to cut and paste sections, excluding user interface code into your own project.*

RETURNS

TRUE	The operation succeeded
FALSE	The operation failed consult errno

errno is set to one of the following:

0	No error
PEINVALIDDRIVEID	Drive component is invalid
PEINVALIDPARMS	Inconsistent or missing parameters
PEIOERRORWRITE	Error writing during format
An Rtfs system error	See Appendix for a description of system errors

SEE ALSO

pc_get_media_parms(), **pc_partition_media()**, **pc_format_media()**

EXAMPLE

See the routine **doformat()** in appcmdshformat.c.

pcexfat_format_volume

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Perform an exFAT volume format

SUMMARY

BOOLEAN **pcexfat_format_volume** (byte *path)

DESCRIPTION

This routine partitions and formats the drive referred to by drive letter. The device is partitioned and formatted according to rules in the SD card association exFAT file specification.

Note: The source of appcmdshformat.c *contains source code with example usage.* **appcmdshformat.c** *is intentionally partitioned to be easy to cut and paste sections, excluding user interface code into your own project.*

RETURNS

TRUE	The operation succeeded
FALSE	The operation failed consult errno

errno is set to one of the following:

0	No error
PEINVALIDDRIVEID	Drive component is invalid
PEIOERRORREADMBR	Partitioned device. I/O error reading
PEINVALIDMBR	Partitioned device has no master boot record
PEINVALIDMBROFFSET	Requested partition has no entry in master boot record
PEINVALIDPARMS	Inconsistent or missing parameters
PEIOERRORWRITE	Error writing during format
PEDYNAMIC	A dynamic device driver is present but it returned invlid parameters
An Rtfs system error	See Appendix for a description of system errors

SEE ALSO

pc_ format_volume()

EXAMPLE

See the routine **doexfatformat()** in appcmdshformat.c.

Utility API

pc_deltree

pc_deltree_uc

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Delete a directory tree

SUMMARY

BOOLEAN **pc_deltree** (byte *directory_name)

DESCRIPTION

Delete the directory specified in directory_name, deletes all subdirectories of that directory, and all files contained therein. Fail if directory_name is not a directory, is read only or is currently in use.

Note: If a portion of the tree being deleted is in use, either with an open file or directory traversal, then the deltree algorithm will abort leaving the tree partially removed.

RETURNS

TRUE	The directory was successfully removed
FALSE	consult errno

errno is set to one of the following:

0	No error
PEINVALIDDRIVEID	Drive name is invalid
PEINVALIDPATH	Path specified by name is badly formed.
PENOENT	Can't find path specified by name.
PEACCES	Directory or one of its subdirectories is read only or in use.
An Rtfs system error	See Appendix for a description of System Errors

pc_enumerate

pc_enumerate_uc

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Recursively process all directory entries that match a pattern.

SUMMARY

int **pc_enumerate**(
byte * from_path_buffer
- pointer to a scratch buffer of size EMAXPATH

byte * from_pattern_buffer
- pointer to a scratch buffer of size EMAXPATH

byte * spath_buffer
- pointer to a scratch buffer of size EMAXPATH

byte * dpath_buffer
- pointer to a scratch buffer of size EMAXPATH

byte * root_search
- Root of the search IE C:\ or C:\USR etc.

word match_flags
- Selection flags (see below)

byte match_pattern
- Match pattern (see below)

int maxdepth
- Maximum depth of the traversal.

PENUMCALLBACK pcallback
- User callback function (see below).

)

DESCRIPTION

This routine traverses a subdirectory tree and tests each directory entry to see if it matches user supplied selection criteria. If it does match the criteria, a user supplied callback function is called with the full path name of the directory entry and a pointer to a DSTAT structure that contains detailed information about the directory entry (see the **pc_gfirst()** manual page for a detailed description of the DSTAT structure).

Selection criteria: Two arguments are used to determine the selection criteria. One is a flags word that specifies attributes; the other is a pattern that specifies a wild card pattern.

The flags argument specifies what types of directory entries will be considered a match if the wildcard match succeeds. It must contain a bitwise oring together of one or more of the following:

MATCH_DIR	Select directory entries
MATCH_VOL	Select volume labels
MATCH_FILES	Select files
MATCH_DOT	Select '.' entry MACTH_DIR must be true too
MATCH_DOTDOT	Select '..' entry MATCH_DIR must be true too

*The selection pattern is a standard wildcard pattern such as *, '*.*' or *.txt*

Note: **pc_enumerate()** *requires a fair amount of buffer space to function. Instead of allocating the space internally, we require the application to pass three buffers of size EMAXPATH in to the function. See below.*
Note: to scan only one level set maxdepth to 1. For all levels set it to 99.

RETURNS

Returns 0 unless the callback function returns a non-zero value at any point. If the callback returns a non-zero value, the scan terminates immediately and returns the returned value to the application.

This function does not set errno.

About the callback:

The callback function returns an integer and is passed the fully qualified path to the current directory entry and a DSTAT structure. The callback function must return 0 if it wishes the scan to continue or any other integer value to stop the scan and return the callback's return value to the application layer.

EXAMPLE 1 - Print the name of every file and directory on a disk

```
byte buf0[EMAXPATH], buf1[EMAXPATH], buf2[EMAXPATH], buf3[EMAXPATH];
int rdir_callback(byte *path, DSTAT *d) {printf("%s\n", path);return(0);}

print_all()
{
    pc_enumerate(buf0,buf1,buf2,buf3,"\\",(MATCH_DIR|MATCH_FILES),
    "*",99,rdir_callback);
}
```

EXAMPLE 2 -Delete every file on a disk

```
int delfile_callback(byte *path, DSTAT *d) {pc_unlink(path); return(0);}
delete_all()
{
    pc_enumerate(buf0,buf1,buf2,buf3,"\\",(MATCH_DIR|MATCH_FILES),
    "*",99, delfile_callback);
}
```

Note appcmdsh.c provides source code for an example command "ENUMDIR" which uses **pc_enumerate()**.

pc_check_disk

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Check a volume's integrity

SUMMARY

BOOLEAN **pc_check_disk** (byte *drive_id, CHKDISK_STATS *pstat, int verbose, int fix_problems, int write_chains)

DESCRIPTION

This routine scans the disk searching for lost chains and crossed files and returns information about the scan in the structure at pstat. If fix_problems is non-zero it corrects file sizes if necessary. If fix_problems is non-zero and if write_chains is zero, it frees lost cluster chains; if write_chains is non-zero, it writes lost chains to files names FILE???.CHK in the root directory. If fix_problems is zero the write_chains argument is ignored.

pstat - a pointer to a structure of type CHKDISK_STATS. **pc_check_disk()** returns information about the disk in this structure.

```
typedef struct typedef struct chkdisk_stats {
dword n_user_files
dword n_hidden_files;
dword n_user_directories;
dword n_free_clusters;
dword n_bad_clusters;        /* # clusters marked bad */
dword n_file_clusters;       /* Clusters in non hidden files */
dword n_hidden_clusters;     /* Clusters in hidden files */
dword n_dir_clusters;        /* Clusters in directories */
dword n_crossed_points;      /* Number of crossed chains. */
dword n_lost_chains;         /* # lost chains */
dword n_lost_clusters;       /* # lost clusters */
dword n_bad_lfns;            /* # corrupt/disjoint lfns */
} CHKDISK_STATS;
```

verbose **-** If this parameter is 1 **pc_check_disk()** prints status information as it runs. If it is 0 **pc_check_disk()** runs silently.

fix_problems - If this parameter is 1 **pc_check_disk()** will make repairs to the volume, if it is zero, problems are reported but not fixed.

write_chains - If this parameter is 1 **pc_check_disk()** creates files from lost chains. If write_chains is 0 lost chains are automatically discarded and freed for reuse. If fix_problems is 0 then write_chains has no affect.

RETURNS

TRUE	The operation succeeded
FALSE	The operation failed consult errno

pc_check_disk() does not set errno.

EXAMPLE

```
CHKDISK_STATS chkstat;
pc_check_disk(“A:”, &chkstat, 1, 1, 0);
/* Check disk, be verbose, fix problems, free lost chains */
pc_check_disk(“A:”, &chkstat, 1, 1, 1);
/* Check disk, run quietly, fix problems, convert lost chains to files */
return(0);
```

Note:
Failsafe users should never require pc_check_disk()

Miscellaneous functions

tst_shell

Basic	X	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Interactive command Shell

SUMMARY

pc_tstsh(void)

DESCRIPTION

This subroutine provides an interactive command shell for controlling Rtfs. It provides a handy method for testing and exercising your port of Rtfs and it may be used to maintain the file system on your target system.

The test shell contains most basic file system maintenance commands like "mkdir", "rmdir" etc.

A command shell reference guide is included in the application notes.

Note: The source code for the command shell is provided in several files contained in rtfscommom/apps and rtfsproplus/apps this source code contains many examples of calling and using the Rtfs API.

EXAMPLE

```
main()
{
      pc_ertfs_run(); /* Don't forget to call the initialization  code */
      pc_tstsh(); /* Call the test shell. It will execute until
                       the user types QUIT */
      exit(0);
}
```

pc_free_user

Basic	x	ProPlus	x
Pro	x	ProPlus DVR	x

FUNCTION

Release this task's Rtfs user context block

SUMMARY

#include <rtfs.h>
void **pc_free_user()**
pc_tstsh

DESCRIPTION

NOTE: This routine should be called by all tasks that have used Rtfs before they exit.

When a task first uses the Rtfs API, a user context block is automatically created specifically for that task. Before the task exits it must release its context block, otherwise Rtfs will run out of context blocks and all new tasks will have to share the same context block.

Typical places to call to **pc_free_user()** are just prior to a task returning or exiting or your RTOS's task exit callback routine or in an "onexit" processing subroutine.

Please see the explanation for **RTFS_CFG_NUM_USERS** in the Configuration Guide for more information about this function.

RETURNS

Nothing

EXAMPLE

```
void my_ftp_server_task()
{
    do_server_session(); /* Call the ftp server function here */
    pc_free_user(); /* Free Rtfs resources for this thread */
    exit(0); /* Terminate the thread */
}
```

Sixty four bit math package

Basic		ProPlus	X
Pro		ProPlus DVR	x

A macro package is available to perform 64 bit arithmetic. This macro package works on processors with 64 bit native integer support and on processors that provide only 32 bit integers.

This macro package is useful for application programming with 64 bit files.

A synopsis of the available macros is provided here. Many sample uses of these macros may also be found in the source code for the test suite in the subdirectory rtfspackages/apps**.**

Mixed 64 bit 32 bit operators

dword **M64HIGHDW**(ddword A)	- Returns the high 32 bits of a 64 bit int.
dword **M64LOWDW**(ddword A)	- Returns the low 32 bits of a 64 bit int.
ddword **M64SET32**(dword HI, dword LO)	- Create a 64 bit int from 2 32 bit ints.
ddword **M64PLUS32**(ddword A, dword B)	- Add a 32 bit int to a 64 bit int.
ddword **M64MINUS32**(ddword A, dword B)	- Subtract a 32 bit int from a 64 bit int.

64 bit arithmetic operators

ddword **M64PLUS**(ddword A, ddword B)	- Add 2 64 bit ints.
ddword **M64MINUS**(ddword A, ddword B)	- Subtract a 64 bit int from a 64 bit int.
ddword **M64LSHIFT**(ddword A, int B)	- Left shift a 64 bit int by B.
ddword **M64RSHIFT**(ddword A,int B)	- Right shift a 64 bit int by B.

64 bit logical operators

BOOLEAN **M64IS64**(ddword A)	- TRUE if A > than the largest 32 bit int
BOOLEAN **M64EQ**(ddword A, ddword B)	- TRUE if A equals B
BOOLEAN **M64LT**(ddword A, ddword B)	- TRUE if A less than B
BOOLEAN **M64LTEQ**(ddword A, ddword B)	- TRUE if A less than or equal B
BOOLEAN **M64GT**(ddword A, ddword B))	- TRUE if A greater than B
BOOLEAN **M64GTEQ**(ddword A, ddword B)	- TRUE if A greater than or equal to B
BOOLEAN **M64NOTZERO**(ddword A)	- TRUE if A is not zero.
BOOLEAN **M64ISZERO**(ddword A)	- TRUE if A is equal to zero.