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

All Rtfs packages - 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	

All Rtfs packages – 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_sector s	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_sector s	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	

All Rtfs packages – 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).

<u>**RETURNS**</u>

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_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

All Rtfs packages - 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)

RETURNS

>= 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

EXAMPLE

```
#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

po_ulseek

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

All Rtfs packages - 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.

All Rtfs packages - 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()

All Rtfs packages - 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.

RtfsProPlus - Real time and direct disk management API

pc_diskio_runtime_stats

Basic		ProPlus	x
Pro		ProPlus DVR	x

FUNCTION

Return buffering and block transfer usage patterns for a drive.

SUMMARY

BOOLEAN **pc_diskio_runtime_stats** (driveid, pstats, clear)

byte *driveid	Name of a mounted volume "A," "B," etc.
DRIVE_RUNTIME_STATS *pstats	Usage statistics are placed into this structure.
BOOLEAN clear	If this argument is **TRUE** the internal statistics are all set to zero after they are copied to the pstats buffer. This option is useful if you wish to monitor the usage patterns of a single operation or a group of operations. You can clear the statistics, then perform your operations and then retrieve the statistics. The statistics will contain only the accesses made while performing the operations under investigation.

DESCRIPTION

*Note: **pc_diskio_runtime_stats()** returns usage statistics that are acquired while Rtfs is running. To use this feature the Rtfs library must be compiled with the **INCLUDE_DEBUG_RUNTIME_STATS** define set to one (rtfsconf.h). If it is not enabled the function will zero fill the **DRIVE_RUNTIME_STATS** structure and return.*

These statistics are a tool for determining if you have optimally configured Rtfs for your application and how efficiently your application is using the library. Please see the description of the pstats structure below to learn how to interpret these statistics.

Detailed description of the stats structure fields. All fields are of type **dword**.	
Asynchronous statistics	
async_steps	Total number of steps made to complete async operations
Fat table access statistics for the buffered region used for directory clusters. Note that the ratio of fat_blocks_read to fat_reads and fat_blocks_written to fat_writes gives an indication of how successfully the fat buffer paging algorithm and multiblock FAT access code is performing.	
fat_reads	Number of reads made to the FAT region of the disk.
fat_blocks_read	Total number of FAT blocks read.
fat_writes	Number of writes made to the FAT region of the disk.
fat_blocks_written	Total number of blocks written.
fat_buffer_swaps	Number of FAT buffer pool misses that required flushing of a FAT buffer block.
Subdirectory block buffer pool access statistics. *These fields indicate how many cache hits, reads and writes have occurred. The number of hits is not a huge factor in performance. This is because Rtfs uses large multi-block reads whenever possible to quickly scan directory blocks*	
dir_buff_hits	Number of directory block read accesses that were fulfilled by data cached from previous reads or buffer initializations.
dir_buff_reads	Number of directory block read accesses that required reading from the disk.
dir_buff_writes	Number of buffered directory block writes.
Subdirectory block direct access statistics. Rtfs scans subdirectories using multi-block transfers whenever possible. The ratio of dir_direct_blocks_read to dir_direct_reads gives an indication of the amount this is occurring.	
dir_direct_reads	Number of un-buffered directory block read calls.
dir_direct_blocks_read	Number of un-buffered directory blocks read.
dir_direct_writes	Number of un-buffered directory block write calls.
dir_direct_blocks_written	Number of un-buffered directory blocks written.
File access statistics. *Application controlled read, write, and seek patterns can have a large impact on system performance. For optimal performance, applications should read and write data in chunks that are as large as possible and are always aligned on 512 byte file pointer boundaries. If this is not your application's access pattern then some degree of data copying and additional disk reads will occur.*	

If the values file_buff_reads *or* file_buff_writes *are large, the application is not behaving optimally and an attempt should be made to align the file data. If this is not possible then the file, or files, with unaligned accesses should be opened in buffered mode. When files are opened in buffered mode the values of both* **file_buff_reads** *and* file_buff_writes *should decrease and you should see and increase in the value of* file_buff_hits*.*	
file_buff_hits	Number of unaligned file block read accesses that were fulfilled by data cached from previous unaligned reads.
file_buff_reads	Number of unaligned file block read accesses that required reading from the disk.
file_buff_writes	Number of unaligned file block writes to the disk.
file_direct_reads	Number of block read calls of file data directly to the application buffer.
file_direct_blocks_read	Number of blocks of file data read directly to the application buffer.
file_direct_writes	Number of block write calls of file data directly from the application buffer.
file_direct_blocks_written	Number of blocks of file data written directly from the application buffer.

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

pc_diskio_free_list

Basic		ProPlus	x
Pro		ProPlus DVR	x

FUNCTION

Query free cluster segments on the drive.

SUMMARY

#include <rtfs.h>
BOOLEAN **pc_diskio_free_list**(byte *driveid, int listsize, FREELISTINFO *plist, dword startcluster, dword endcluster, dword threshhold)

driveid	Name of the volume “A:” “B:” etc.
listsize	Must contain the number of FREELISTINFO elements in the array provided in plist.
plist	Must contain the address of an array of listsize FREELISTINFO elements The **FREELISTINFO** structure is defined as follows: **typedef struct freelistinfo {** dword cluster; Cluster where free region starts dword nclusters; Number of free clusters **} FREELISTINFO;**
startcluster	Must contain the start of the cluster range to scan for free clusters. If startcluster is zero it scans from the beginning of the FAT
endcluster	Must contain the end of the cluster range to scan for free clusters. If endcluster is zero it scans to the end of the FAT
threshold	Selects the minimum sized contiguous free region to report. This argument allows the caller to exclude free chains that are less than a certain number of contiguous clusters. Set threshhold to one to report every free cluster segment in the range. Set threshhold to a larger value to filter out free fragments that are less than some minimum size. *Note: The value of threshold must be at least 1.*

DESCRIPTION

This routine returns a list of currently free cluster segments and places the results in the **FREELISTINFO** structure array.

The results may be used to analyze disk fragmentation patterns and for allocating specific clusters to individual files using **pc_efilio_setalloc()** and **pc_cflio_setalloc()**.

*Note: If there are more free cluster extents than will fit in plist****,*** *as indicated by listsize, then the list is not updated beyond listsize elements. However, the count is updated and returned, so the list size may be adjusted and the routine may be called again.*

Note: When ***INCLUDE_RTFS_FREEMANAGER*** *is enabled this function executes quickly, accessing only ram based structure. When* ***INCLUDE_RTFS_FREEMANAGER*** *is disabled the function scans the disk based FAT table, taking a longer time to complete.*

RETURNS

>= 0	Returns the number of free extents in the file or -1 on error
-1	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

pc_efilio_setalloc

Basic		ProPlus	x
Pro		ProPlus DVR	x

FUNCTION

Specify and optionally reserve clusters for a file.

SUMMARY

BOOLEAN **pc_efilio_setalloc(**int fd, dword cluster, dword ntoreserve)

int fd	A file descriptor that was returned from a successful call to pc_efilio_open.
dword cluster	Hint for the next cluster to allocate, or start of clusters to reserve
dword ntoreserve	If ntoreserve is non-zero then clusters in the range cluster to cluster + ntoreserve -1 are removed from free space and added to the file's reserved cluster list. When the file is expanded, these clusters are used. When the file is closed, any unused clusters in the reserve list are released. If all of the specified clusters are not currently free then **pc_efilio_setalloc()** fails and sets errno to **PEINVALIDPARMS**. If ntoreserve is zero, the clusters are not pre-allocated but when the file is next expanded, Rtfs tries to allocate clusters starting at cluster. If cluster is already in use, it allocates starting at the next free cluser beyond cluster.

DESCRIPTION

pc_efilio_setalloc() allows the programmer to either specify a hint where the next cluster should be allocated from or to specify a group of clusters to be pre-allocated to this file for its exclusive use.
*Note: **pc_diskio_free_list()** may be used in conjunction with **pc_efilio_setalloc()** to retrieve a free cluster map and assign specific clusters from that map to files.*

RETURNS

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

Application Level Error Return Codes

PEBADF	Invalid file descriptor
PECLOSED	File is no longer available. Call pc_efilio_close().
PEEFIOILLEGALFD	The file not open in extended IO mode
PEINVALIDPARMS	Missing or invalid parameters
PEEINPROGRESS	Asynchronous operation already in progress
An Rtfs system error	See Appendix for a description

pc_efilio_get_file_extents

Basic		ProPlus	x
Pro		ProPlus DVR	x

FUNCTION

Get the list of segments that make up a file.

SUMMARY

int **pc_efilio_get_file_extents**(int fd, int infolistsize,
FILESEGINFO *plist, BOOLEAN report_clusters, BOOLEAN raw)

int fd	A file descriptor returned from **po_open()** or **pc_efilio_open()**
infolistsize	The number of elements in the storage pointed to by plist.
plist	A pointer to a buffer or array of FILESEGINFO structures. The buffer must contain space for at least infolistsize FILESEGINFO structures The FILESEGINFO structure is defined as follows: **typedef struct fileseginfo {** dword block; Block or cluster number of the extent dword nblocks; Number of blocks or clusters in the extent **} FILESEGINFO;**
report_clusters	If report_clusters is **TRUE** the file segments in plist are reported in clusters. If report_clusters is **FALSE** the file segments in plist are reported in blocks.
raw	If report_clusters is **TRUE** this argument is ignored. If report_clusters is **FALSE** this argument does the following: If raw is **FALSE** blocks are reported as block offsets from the start of the partition. If raw is **TRUE** blocks are reported as block offsets from the start of the device. *Note: Set raw to **TRUE** if you will be using the resultant list to set up DMA transfers to or from the disk.*

DESCRIPTION

This routine traverses the cluster chain of the open file, fd, and logs into the list at plist the block location and length in blocks of each segment of the file. The block numbers and block length information can then be used to read and write the file directly using **pc_raw_read()** and **pc_raw_write()** or the information may be used to set up DMA transfers to or from the raw block locations. If the file contains more extents than will fit in plist as indicated by infolistsize then the list is not updated beyond infolistsize elements but the count is updated and returned so the list size may be adjusted and the routine may be called again.

Note: Rtfs provides an alternative way to inspect an open file's cluster chain. Please See ***pc_efilio_fstat().***

RETURNS

>= 0	The operation was a success. The return value is the number of extents in the file.
-1	The operation failed consult errno

Application Level Error Return Codes

PEINVALIDPARMS	Missing or invalid parameters
PEBADF	Invalid file descriptor
An Rtfs system error	See Appendix for a description

pc_get_dirent_info

Basic		ProPlus	X
Pro		ProPlus DVR	X

FUNCTION

Retrieve low level directory entry information.

SUMMARY

BOOLEAN **pc_get_dirent_info**(path, pinfo)

byte *path	File or directory name

DIRENT_INFO *pinfo	Retrieves the following information:
	
	typedef struct dirent_info {
	    byte fattribute;
	    dword fcluster;
	    word ftime;
	    word fdate;
	    dword fsize;
	    dword my_block;
	    int my_index;
	} DIRENT_INFO;
	
	Note: my_block and my_index are the block number and directory entry index within the block (0 to 16 with 512 byte sectors). They cannot be changed. The other elements may be changed and passed to pc_set_dirent_info.

DESCRIPTION

Given a file or directory name and a dirent_info buffer fill the buffer with low level directory entry information. This structure can be examined and it can be modified and passed to **pc_set_dirent_info()** to change the entry.

RETURNS

TRUE	If no errors were encountered.
FALSE	An error occurred

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

pc_set_dirent_info

Basic		ProPlus	x
Pro		ProPlus DVR	x

FUNCTION

Change low level directory entry information.

SUMMARY

BOOLEAN **pc_set_dirent_info**(path, pinfo)

byte *path	File or directory name

DIRENT_INFO *pinfo	An info structure returned from pc_get_dirent_info.
	
	typedef struct dirent_info {
	    byte fattribute;
	    dword fcluster;
	    word ftime;
	    word fdate;
	    dword fsize;
	    dword my_block;
	    int my_index;
	} DIRENT_INFO;
	
	The entries in the dirent_info structure that may be changed by pc_set_dirent_info are:
	fattribute,fcluster,ftime, fsize and fdate

DESCRIPTION

To use this function you should first call **pc_get_dirent_info()** to retrieve low level directory entry information and then change the fields you wish to modify and then call the function to apply the changes.

Note: This is a very low level function that could cause serious problems if used incorrectly.

For example, to move the contents from "File A" to "File B" you could perform the following.

```
DIRENT_INFO fileainfo, filebinfo;

    pc_get_dirent_info("File A", &fileainfo);
    pc_get_dirent_info("File B", &filebinfo);
    filebinfo.fcluster =  fileainfo.fcluster;
    filebinfo.fsize    =  fileainfo.fcfsize;
    fileainfo.fcluster =  0;
```

```
        fileainfo.fsize     =  0;
        pc_set_dirent_info("File A", &fileainfo);
        pc_set_dirent_info("File B", &filebinfo);
```

RETURNS

TRUE	If no errors were encountered.
FALSE	An error occurred

If an error occurred: errno is set to one of the following:
Application Level Error Return Codes

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

pc_efilio_fpos_sector

Basic		ProPlus	x
Pro		ProPlus DVR	x

FUNCTION

Retrieve the sector number and count of contiguous sectors at the current file pointer

SUMMARY

BOOLEAN **pc_efilio_fpos_sector**(int fd, BOOLEAN isreadfp, BOOLEAN raw,
dword *psectorno, dword *psectorcount)

int fd	A file descriptor that was returned from a successful call to pc_efilio_open or pc_cfilio_open
BOOLEAN isreadfp	Set this to TRUE if the intention is to read the sectors from the disk. Set it to FALSE if the intention is to write. Proper use of this argument is important for the following reasons: 1. If fd is a circular file the isreadfd argument is used to select which file pointer (read or write) to use when calculating *psectorno. 2. If isreadfp is **TRUE**, *psectorcount will contain the number of contiguous sectors starting at the file pointer up to the file size. Sectors from pre-allocated clusters that are not yet accounted for in the file's size will not be included. 3. If isreadfp is **FALSE**, *psectorcount will return the number of contiguous sectors starting at the file pointer and include sectors from pre-allocated clusters that are not yet accounted for in the file's size.
BOOLEAN raw	If raw is **TRUE**, the returned sector number will be the actual sector number on the device. If raw is **FALSE**, the returned sector number will be the sector number offset within the partition For DMA enabling raw should always be set to **TRUE**.
dword *psectorno	The current sector number is returned through this pointer.
dword *psectorcount	The number of contiguous sectors starting at this location is returned through this pointer. This value may be zero if this is a read request and you are at end of file or if it is a write request on a circular file opened in **PCE_CIRCULAR_FILE** mode and the write pointer has caught the read pointer.

DESCRIPTION

This function may be used to enable your application to DMA data directly to and from files. Special programming techniques allow DMA reading and writing disk sectors that already exist in the file as well as extents that are to be appended to the file.

Please study the application notes for an in-depth discussion on how to use this function along with other Rtfs functions to achieve these results.

RETURNS

TRUE	If no errors were encountered. *psectorno is set to the sector number at the file pointer. *psectorcount is set to the number of contiguous sectors at the file pointer.
FALSE	An error occurred

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

PEBADF	Invalid file descriptor
PECLOSED	File is no longer available. Call close.
An Rtfs system error	See Appendix for a description

pc_fd_to_driveid

Basic		ProPlus	x
Pro		ProPlus DVR	x

FUNCTION

Get the drive identifier associated with an open file.

SUMMARY

#include <rtfs.h>
int **pc_fd_to_driveid** (int fd, byte* pdrive_name)

DESCRIPTION

Use this function to get the drive number and drive name of the disk that the file associated with fd resides on.

This function populates the argument pdrive_name with the drive name and returns the drive number.

Note: It is legal to pass a NULL pointer instead of a pointer for the pdrive_name argument but if pdrive_name is non-NULL the buffer pointed to it must be large enough to contain the NULL terminated drive identifier. This is 3 bytes in ASCII or JIS, 6 bytes in UNICODE.

Note: The file descriptor may be a file descriptor that was returned by **po_open()**, **pc_efilio_open()**, **pc_cfilio_open()** or **pc_async_unlink_start()**.

RETURNS

>= 0	Drive number
-1	The file descriptor is invalid consult errno

pc_cluster_to_sector

Basic		ProPlus	x
Pro		ProPlus DVR	x

FUNCTION

Convert a cluster number to a sector number.

SUMMARY

dword **pc_cluster_to_sector** (driveno, cluster, raw)

int driveno	The drive number where the cluster resides
dword cluster	Cluster number to map to a sector number
BOOLEAN raw	If raw is **TRUE** then the return value will be the offset from the beginning of the physical device. If raw is **FALSE** then the return value will be the offset from the beginning of the partition.

DESCRIPTION

pc_cluster_to_sector() takes a cluster number and converts it to a sector number. This can be a useful informational tool and may be used in conjunction with other functions to implement specific block placement and DMA schemes.

RETURNS

>0	The sector number of the cluster.
0	An error occurred

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

PEINVALIDDRIVEID	Driveno is incorrect
PEINVALIDPARMS	Invalid arguments
An Rtfs system error	See Appendix for a description

pc_sector_to_cluster

Basic		ProPlus	x
Pro		ProPlus DVR	x

FUNCTION

Convert a sector number to a cluster number.

SUMMARY

dword **pc_sector_to_cluster** (driveno, sector, raw)

int driveno	The drive number
dword sector	Sector number to map to a cluster value
BOOLEAN raw	If raw is **TRUE** then sector must be the offset from the beginning of the physical device. If raw is **FALSE** then sector must be the offset from the beginning of the partition.

DESCRIPTION

pc_sector_to_cluster() Takes a sector number and converts it to a cluster number. This can be a useful informational tool and it may be used in conjunction with other functions to implement specific block placement and DMA schemes

RETURNS

>0	The cluster number of the sector.
0	An error occurred

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

PEINVALIDDRIVEID	Driveno is incorrect
PEINVALIDPARMS	Invalid arguments
An Rtfs system error	See Appendix for a description

pc_raw_read

Basic		ProPlus	x
Pro		ProPlus DVR	x

FUNCTION

Perform direct block read

SUMMARY

BOOLEAN **pc_raw_read** (driveno, buf, blockno, nblocks, raw)

int driveno	The drive number to read from
byte *buf	Buffer where data is to be read
dword blockno	Sector number to read
dword nblocks	Number of sectors to read
BOOLEAN raw	If raw is **TRUE** then blockno is the offset from the beginning of the physical device. If raw is **FALSE** then blockno is the offset from the beginning of the partition. If raw is **FALSE** and the drive volume is not currently mounted Rtfs will attempt to mount the device. If raw is **TRUE** the device may be accessed even if the volume is not currently mounted.

DESCRIPTION

pc_raw_read() bypasses normal file IO and reads block oriented data directly from the disk.

RETURNS

TRUE	If no errors were encountered
FALSE	An error occurred

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

PEINVALIDDRIVEID	Driveno is incorrect
PEINVALIDPARMS	Invalid arguments
PEIOERRORREAD	Error performing read
An Rtfs system error	See Appendix for a description

pc_raw_write

Basic		ProPlus	x
Pro		ProPlus DVR	x

FUNCTION

Perform direct block write

SUMMARY

BOOLEAN **pc_raw_write**(driveno, buf, blockno, nblocks, raw)

int driveno	The drive number to write to
byte *buf	Buffer where data is to be written from
dword blockno	Sector number to write
dword nblocks	Number of sectors to write
BOOLEAN raw	If raw is **TRUE** then blockno is the offset from the beginning of the physical device. If raw is **FALSE** then blockno is the offset from the beginning of the partition. If raw is **FALSE** and the drive volume is not currently mounted Rtfs will attempt to mount the device. If raw is **TRUE** the device may be accessed even if the volume is not currently mounted.

DESCRIPTION

pc_raw_write() bypasses normal file IO and writes block oriented data directly to the disk.

RETURNS

TRUE	If no errors were encountered
FALSE	An error occurred

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

PEINVALIDDRIVEID	Driveno is incorrect
PEINVALIDPARMS	Invalid arguments
PEIOERRORWRITE	Error performing write
An Rtfs system error	See Appendix for a description

pc_bytes_to_clusters

Basic		ProPlus	x
Pro		ProPlus DVR	x

FUNCTION

Calculate the minimum number of clusters to contain the number of bytes.

SUMMARY

BOOLEAN **pc_bytes_to_clusters** (driveno, bytes_hi, bytes_lo, *presult)

int driveno	The drive number
dword bytes_hi	High 32 bits of 64 bit byte count. (0 for a 32 bit byte count)
dword bytes_lo	32 bit byte count or low 32 bits of 64 bit byte count.
dword *presult	Returns the minimum number of clusters to contain the byte count.

DESCRIPTION

pc_bytes_to_clusters() takes a byte count and calculates the minimum number of clusters required to contain the number of bytes

RETURNS

TRUE	Success
FALSE	An error occurred

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

PEINVALIDDRIVEID	Driveno is incorrect
PEINVALIDPARMS	Invalid arguments
An Rtfs system error	See Appendix for a description

pc_clusters_to_bytes

Basic		ProPlus	x
Pro		ProPlus DVR	x

FUNCTION

Calculate the number of bytes contained in the number of clusters.

SUMMARY

BOOLEAN **pc_clusters_to_bytes** (driveno, n_clusters, *pbytes_hi, *pbytes_lo)

int driveno	The drive number
dword n_clusters	The number of clusters to convert
dword *pbytes_hi	High 32 bits of 64 bit result.
dword *pbytes_lo	32 bit result or low 32 bits of 64 bit result.

DESCRIPTION

pc_clusters_to_bytes() Calculates the number of bytes in n_clusters.

RETURNS

TRUE	Success
FALSE	An error occurred

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

PEINVALIDDRIVEID	Driveno is incorrect
PEINVALIDPARMS	Invalid arguments
An Rtfs system error	See Appendix for a description

pc_subtract_64

Basic		ProPlus	x
Pro		ProPlus DVR	x

FUNCTION

Subtract two sixty four bit numbers using hi, lo 32 bit format.

SUMMARY

void **pc_subtract_64** (hi_1, lo_1, hi_0, lo_0, *pres_hi, *pres_lo)

dword hi_1	High 32 bits of first number.
dword lo_1	Low 32 bits of first number.
dword hi_0	High 32 bits of second number.
dword lo_0	Low 32 bits of second number
dword *pres_hi	High 32 bits of result
dword *pres_lo	Low 32 bits of result

DESCRIPTION

With this subroutine you can subtract two 64 bit numbers that are already represented in 32 bit hi:lo format. This is the format that is used by seek and extract API routines.

The routine performs the following operation:

```
(*pres_hi:*pres_lo) = (hi_1:lo_1) – (hi_0:lo_0);
```

RETURNS

Nothing

pc_add_64

Basic		ProPlus	x
Pro		ProPlus DVR	x

FUNCTION

Add two sixty four bit numbers using hi, lo 32 bit format.

SUMMARY

void **pc_add_64** (hi_1, lo_1, hi_0, lo_0, *pres_hi, *pres_lo)

dword hi_1	High 32 bits of first number.
dword lo_1	Low 32 bits of first number.
dword hi_0	High 32 bits of second number.
dword lo_0	Low 32 bits of second number
dword *pres_hi	High 32 bits of result
dword *pres_lo	Low 32 bits of result

DESCRIPTION

With this subroutine you can add two 64 bit numbers that are already represented in 32 bit hi:lo format. This is the format that is used by seek and extract API routines.

The routine performs the following operation:

```
        (*pres_hi:*pres_lo) = (hi_1:lo_1) + (hi_0:lo_0);
```

RETURNS

Nothing

RtfsProPlus - Extended file IO API

pc_efilio_open

pc_efilio_open_uc

Basic		ProPlus	x
Pro		ProPlus DVR	x

FUNCTION

Open a file for extended IO operations.

*Note: **pc_efilio_open()** is highly configurable. This manual section provides a comprehensive reference for the function but you may also wish to visit the rtfsproplus/apps subdirectory and view the source code, which makes multiple calls to **pc_efilio_open()**.*

SUMMARY

int **pc_efilio_open** (name, flag, mode, poptions)

byte *name	File name
word flag	Flag values
word mode	Mode values
EFILEOPTIONS *poptions	Extended options, if NULL no extended options are used.

DESCRIPTION

Open the file for access as specified in flag with additional options specified in the poptions structure.

Flag values are:

PO_BINARY	Ignored. All file access is binary
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. Use mode to specify the permission on the file
PO_EXCL	If flag contains (**PO_CREAT** | **PO_EXCL**) and the file already exists fail and set errno to **PEEXIST**
PO_TRUNC	Truncate the file if it already exists
PO_NOSHAREANY	Fail if the file is already open. If the open succeeds, no other opens will succeed until it is closed
PO_NOSHAREWRITE	Fail if the file is already open for write. If the open succeeds no other opens for write will succeed until it is closed

PO_AFLUSH	Flush the file after each write
PO_APPEND	Always seek to the end of the file before writing
PO_BUFFERED	Use persistent buffers to improve performance of non-block aligned reads and writes

Mode values are:

PS_IWRITE	Write permitted
PS_IREAD	Read permitted. (Always true anyway)

Extended Options

If the options argument is zero no extended options are used, otherwise the options structure must be zeroed and its fields must be initialized properly before they are passed.

The options structure is:

```
typedef struct efileoptions {
      dword allocation_policy;
      dword min_clusters_per_allocation;
      dword allocation_hint;
      byte  *transaction_buffer;
      dword transaction_buffer_size;
      dword circular_file_size_hi;
      dword circular_file_size_lo;
      int   n_remap_records;
      REMAP_RECORD *remap_records;
} EFILEOPTIONS;
```

EFILEOPTIONS:

Allocation_policy – This field contains bit flags that may be set by the user to modify the behavior of the extended file IO routines. See the Allocation policy reference section below for options.

min_clusters_per_allocation	Set this value to a value greater than one to force **pc_efilio_write()** to allocate a minimum number of clusters each time it needs to extend the file. When the file closes any clusters that were pre-allocated but not used return to the disk's free space. (This behavior may be overridden by using the option **PCE_KEEP_PREALLOC**). Cluster pre-allocation is useful for minimizing disk fragmentation and for creating files that are contiguous. *Note: min_clusters_per_allocation plays a key role in DMA enabling applications. Please consult application notes for more information.*

	Note: Cluster pre-allocation features are only available it the memory based free space manager is active.

For example, a high-speed video capture and playback application requires file extents to be contiguous in order to play back the video in real time. If the worst-case file size is, 1000 clusters then set min_clusters_per_allocation to 1000. The first byte that is written will cause the file to be extended by 1000 clusters (if **PCE_FORCE_CONTIGOUS** is set then these clusters will all be contiguous). Then up to 1000 clusters of data may be written to the file without incurring additional overhead. When the file is closed, those clusters that were not consumed are returned to free space.

allocation_hint	Set this value to a specific cluster number if you would like **pc_efilio_write()** to first attempt to allocate file extents from this cluster. This option can be useful if you would like to precisely control the locations on the disk where files data will be placed. This option may be used in conjunction with pre-allocation methods for example to open a file and then pre-allocate some number of clusters to it at a fixed location. Note: The allocation hint may be changed explicitly once the file is opened by calling **pc_efilio_setalloc()** or **pc_cfilio_setalloc()**.

The following options are reserved for calls to pc_cfilio_open() and they must be set to zero when calling pc_efilio_open().

circular_file_size_hi	Set this value to zero
circular_file_size_lo	Set this value to zero
n_remap_records	Set this value to zero
remap_records	Set this value to zero

The following fields must be initialized if the PCE_TRANSACTION_FILE option is selected:

transaction_buffer	This field must contain the address of a memory buffer that is large enough to hold one cluster of data
transaction_buffer_size	This field must contain the size of the transaction_buffer in blocks. It must be greater than or equal to the volume's cluster size

Allocation Policy Reference:

PCE_LOAD_AS_NEEDED - Select this option to disable loading all file cluster chains when the file is opened, and instead, load them as they are needed. File re-opens complete faster with this option enabled, but some small delays are introduced when reads and seeks are performed.

PCE_TEMP_FILE - Select this option to force Rtfs to consider this to be a termporary file and release the directory entry and free all clusters when the file is closed. If the file already exists the open will fail and errno will be set to **PEEXIST**.

*Note: Since the cluster chains of files opened with the **PCE_TEMP_FILE** option are never actually commited to the disk based FAT table, opening with the **PCE_TEMP_FILE** option is more efficient than creating a normal file and deleting it after it is closed.*

PCE_REMAP_FILE- Select this option if the file is to be used as a linear extract file and an argument to **pc_cfilio_extract()**. Data in the file may be read or over-written, but it can't be extended by writing past EOF. Only **pc_cfilio_extract()** may assign file extents to file.

PCE_64BIT_META_FILE - Open the file as a 64-bit metafile. If this option is true and the file already exists as either a 32 bit or 64 bit file, re-open in the correct mode. If the file does not already exist, a 64-bit metafile will be created.
*(**PCE_64BIT_META_FILE** is available for ProPlus64 and ProPlusDvr only)*

PCE_FIRST_FIT - Select this option to give precedence to allocating file extent from the beginning of the file area. Otherwise, the default behavior allocates space near the file's currently allocated extents.

Note: The meaning of the **PCE_FIRST_FIT** *is different when using a dynamic device driver with erase block support (NAND for example). Normally, precedence is given to allocating clusters from empty erase block, but the* **PCE_FIRST_FIT** *option may be used to change the precedence so clusters are allocated from partially full erase blocks instead. This provides a garbage collection method in which certain files with lower performance requirements can scavange free clusters in partially filled erase blocks*

PCE_FORCE_FIRST - Select this option to give precedence to allocating the first free clusters in the range that fulfills the request. If **PCE_FORCE_FIRST** is not enabled, precedence goes to allocating the first contiguous group of free clusters in the range that can fulfill the request. If that fails then the same algorithm as **PCE_FORCE_FIRST** is used.

*Note: If you set both **PCE_FIRST_FIT** and **PCE_FORCE_FIRST** then free clusters are allocated sequentially from the beginning of the FAT. Using this option will help reduce disk fragmentation if it is used on transient files, small files and other files for which a higher amount of fragmentation is acceptable.*

PCE_FORCE_CONTIGUOUS - Select this option to force write calls to fail if the whole request can not be fulfilled in one contiguous extent.

PCE_KEEP_PREALLOC - Select this option to force excess pre-allocated clusters (see: min_clusters_per_allocation) to be incorporated into the file when it is closed. If this option is not selected, then excess pre-allocated clusters are returned to free space when the file is closed.

PCE_ASYNC_OPEN - Perform an asynchronous open of the file. If **PCE_ASYNC_**

OPEN is enabled **pc_efilio_open()** returns quickly after it has determined that the arguments are valid and it has created the directory entry on a file create or for a file re-open, after it has loaded the directory entry contents. If this option is not enabled then **pc_efilio_open()** will make the necessary disk accesses required to load the file's FAT based extent maps. On small files and even files up to several megabytes in size this will not be noticeable, but on very large files this may introduce a perceptible delay. The application must call **pc_async_continue()** to complete the open operation so the file descriptor may be used by the API.

PCE_TRANSACTION_FILE - Open the file in transaction mode. In transaction mode, when Rtfs returns from **pc_efilio_write()**, it guarantees that the data is written to the volume and will survive power loss. If power is interrupted before **pc_efilio_write()** returns then it is guaranteed that the file is unchanged.

If the write operation overwrites an existing region, Rtfs insures that the overwrite may be rewound if a power outage occurs before it completes. If the data is cluster aligned this is done with no copying, and Rtfs achieves similar overwrite performance in transactional mode as it does in normal mode. When the data is not cluster aligned, Rtfs still performs similarly by doing minimal copying (one cluster or less) and using a special buffering scheme.

In transaction mode, Rtfs automatically flushes Failsafe buffers to disk before **pc_efilio_write()** returns. This additional step adds typically one or two additional block writes per write call. This reduces performance somewhat over non transactional files but still provides reasonably high performance.

Notes: Failsafe must be enabled to use this option. To use this option the fields in the options structure named transaction_buffer and transaction_buffer_size must also be intialized.

RETURNS

> = 0	The operation was a success the return value is a valid file descriptor
- 1	The operation failed consult errrno

Application Level Error Return Codes:

PEACCES	Deleting an in-use object or writing to read only object
PEEXIST	Creating an object that already exists
PENOENT	File or directory not found
PENOSPC	Out of space to perform the operation
PESHARE	Sharing violation
PEINVALIDPARMS	Missing or invalid parameters
PEINVALIDPATH	Invalid path name used as an argument
PEEINPROGRESS	Asynchronous operation already in progress
PEEFIOILLEGALFD	File already opened in non extended mode
PE64NOT64BITFILE	Attempt to open a sub-directory with 64 bit file API
An Rtfs system error	See Appendix for a description

pc_efilio_close

Basic		ProPlus	x
Pro		ProPlus DVR	x

FUNCTION

Close an extended 32-bit file or 64-bit metafile.

SUMMARY

BOOLEAN **pc_efilio_close** (int fd)

int fd	A file descriptor that was returned from a successful call to **pc_efilio_open()**

NOTE: An asynchronous file close routine is also available, see **pc_efilio_async_close_start()**

DESCRIPTION

Flush the directory entry and flush the fat chain to disk. Process any deferred cluster chain linking and free all memory associated with the file descriptor.

RETURNS

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

Application Level Error Return Codes

PEBADF	Invalid file descriptor
PEEFIOILLEGALFD	The file not open in extended IO mode
PEINVALIDPARMS	Missing or invalid parameters
PEEINPROGRESS	Asynchronous operation already in progress
An Rtfs system error	See Appendix for a description

pc_efilio_read

Basic		ProPlus	x
Pro		ProPlus DVR	x

FUNCTION

Read from an extended 32 bit file or 64 bit metafile

SUMMARY

BOOLEAN **pc_efilio_read** (fd, buf, count, nread)

int fd	A file descriptor that was returned from a successful call to pc_efilio_open
dword count	The length of the read request, (0 to 0xffffffff)
byte *buf	Buffer where data is to be placed. *NOTE: If buf is a null pointer pc_efilio_read will proceed as usual but it will not transfer bytes to the buffer*
dword *nread	Returns the number of bytes read.

DESCRIPTION

pc_efilio_read() takes advantage of the extended file I/O subsystem to perform file reads. There is no disk latency required for mapping file extents to cluster regions, so reads may be performed at very near the bandwidth of the underlying device.

pc_efilio_read() attempts to read count bytes or to the end of file, whichever is less, from the current file pointer. The value of count may be up 0xffffffff. If the read count plus the current file pointer exceeds the end of file, the read count is truncated to the end of file.

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.

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

Application Level Error Return Codes

PEBADF	Invalid file descriptor
PECLOSED	File is no longer available. Call **pc_efilio_close()**.
PEEFIOILLEGALFD	The file not open in extended IO mode
PEINVALIDPARMS	Missing or invalid parameters
PEEINPROGRESS	Asynchronous operation already in progress
An Rtfs system error	See Appendix for a description

pc_efilio_write

Basic		ProPlus	x
Pro		ProPlus DVR	x

FUNCTION

Write to an extended 32 file or 64 bit metafile

SUMMARY

BOOLEAN **pc_efilio_write** (fd, buf, count, nwritten)

int fd	A file descriptor that was returned from a successful call to pc_efilio_open
dword count	The length of the write request, (0 to 0xffffffff) *NOTE: If count is zero no data is transferred and the file pointer is not moved, but the write routine verifies that clusters are allocated at the file pointer This feature plays a key role in DMA enabling applications. Please consult application notes for more information.*
byte *buf	Buffer containing data is to be written. *NOTE: If buf is a null pointer **pc_efilio_write()** will proceed as usual but it will not transfer bytes to the buffer. This feature may be used to quickly expand a file or to move the file pointer.*
dword *nwritten	Returns the number of bytes written.

DESCRIPTION

pc_efilio_write() takes advantage of the extended file I/O subsystem to perform file writes**.** There is guaranteed no disk latency required for mapping file extents to cluster regions, so writes may be performed at very near the bandwidth of the underlying device.

pc_efilio_write() attempts to write count bytes to the file at the current file pointer. The value of count may be up 0xffffffff.

The behavior of **pc_efilio_write()** is affected by the following options and extended options that were established in the open call.

PO_APPEND	Always seek to end of file before writing.
PCE_FIRST_FIT	Allocate from beginning of file data area
PCE_FORCE_FIRST	Precedence to small free disk fragments over contiguous fragments
PCE_FORCE_CONTIGUOUS	Force contiguous allocation or fail
PCE_TRANSACTION_FILE	When Rtfs returns from **pc_efilio_write()**, it guarantees that the data is written to the volume and will survive power loss. If power is

	interrupted before **pc_efilio_write()** returns then it is guaranteed that the file is unchanged. If the write operation overwrites an existing region, Rtfs insures that the overwritten bytes may be rewound if a power outage occurs before it completes.
min_clusters_per_allocation	If **pc_efilio_write()** needs to allocate clusters during a file extend operation and the **min_clusters_per_allocation** field was established when the file was opened, the **pc_efilio_write()** pre-allocates **min_clusters_per_allocation** clusters.

RETURNS

TRUE	If no errors were encountered. *nwriiten is set to the number of bytes successfully written**.**
FALSE	An error occurred

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

PEBADF	Invalid file descriptor
PECLOSED	File is no longer available. Call **pc_efilio_close().**
PEINVALIDPARMS	Bad or missing argument
PEACCES	File is read only
PEIOERRORWRITE	Error performing write
PEIOERRORREADBLOCK	Error reading block for merge and write
PENOSPC	Disk to full to allocate file minimum allocation size.
PEEFIOILLEGALFD	The file not open in extended IO mode.
PETOOLARGE	Attempt to extend a 32-bit file beyond 4 gigabytes
PERESOURCEREGION	Ran out of region structures while performing operation
An Rtfs system error	See Appendix for a description

pc_efilio_lseek

Basic		ProPlus	x
Pro		ProPlus DVR	x

FUNCTION

Move the file pointer of an extended 32-bit file or 64-bit metafile

SUMMARY

BOOLEAN **pc_efilio_lseek** (fd, offset_hi, offset_lo, origin, *poffset_hi, *poffset_lo)

int fd	A file descriptor that was returned from a successful call to **pc_efilio_open()**
dword offset_hi	High 32-bit word of the 64-bit offset from the beginning of the file. For 32-bit files, this argument must be zero
dword offset_lo	Low 32-bit word of the 64-bit offset from the beginning of the file
int origin	Origin and direction of the request (see below)
dword *poffset_hi	The high dword of the new 64-bit offset from the beginning of the file is returned in - *poffset_hi. For 32-bit files this argument is ignored
dword *poffset_lo	The low dword of the offset from the beginning of the linear file is returned in *poffset_lo

DESCRIPTION

pc_efilio_lseek() takes advantage of the extended file IO subsystem to perform file seeks with zero disk latency. A seek may be performed from anywhere to anywhere in a file almost instantaneously. The offset field is an unsigned long 64-bit value so a single seek may move the file pointer up to 0xffffffffffffffff bytes if the file is an exFat file or 0xffffffff bytes for a 32 bit file.

*Note: The previous statement is not true if the file was opened with the **PCE_LOAD_AS_NEEDEDED** option. In this case, a small delay is incurred when the file pointer is first moved to a region of the file. After that initial seek the extent map is cached and seek behaves as above.*

If a negative offset from the current file pointer is required the origin value **PSEEK_CUR_NEG** may be used. Seeks from **PSEEK_END** are always made in the negative direction from the end of file.

The file pointer is set according to the following rules:

ORIGIN	RULE
PSEEK_SET	Positive offset from beginning of file
PSEEK_CUR	Positive offset from current file pointer
PSEEK_CUR_NEG	Negative offset from current file pointer
PSEEK_END	Negative offset from end of file
PSEEK_SET_RAW	Positive offset from beginning of reserved area that precedes the data. (*see*

	pc_cfilio_extract).

If a **PSEEK_CUR** or **PSEEK_SET** operation attempts to move the file pointer beyond the end of file, the pointer is moved to the end of file.

If a **PSEEK_CUR_NEG** or **PSEEK_END**, operation tries to place the file pointer before zero the file pointer is placed at zero.

To query the current file pointer call:

pc_efilio_lseek(fd, 0, 0, PSEEK_CUR, &offset_hi, &offset_lo)

To report the file size in end_hi:end_lo without moving the file pointer:

pc_efilio_lseek(fd, 0, 0, PSEEK_CUR, &temp_hi, &temp_lo);
pc_efilio_lseek(fd, 0, 0, PSEEK_END, &end_hi, &end_lo);
pc_efilio_lseek(fd, temp_hi, temp_lo, PSEEK_SET, &temp_hi, &temp_lo);

RETURNS

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

Application Level Error Return Codes

PEBADF	Invalid file descriptor
PECLOSED	File is no longer available. Call pc_efilio_close().
PEEFIOILLEGALFD	The file not open in extended IO mode
PEINVALIDPARMS	Missing or invalid parameters
PEEINPROGRESS	Asynchronous operation already in progress
An Rtfs system error	See Appendix for a description

pc_efilio_chsize

Basic		ProPlus	x
Pro		ProPlus DVR	x

FUNCTION

Expand or truncate an extended 32-bit file or 64-bit metafile

SUMMARY

BOOLEAN **pc_efilio_chsize** (fd, newsize_hi, newsize_lo)

int fd	A file descriptor that was returned from a successful call to **pc_efilio_open()**
dword newsize_hi	High 32-bit word of the 64-bit offset from the beginning of the file. For 32-bit files, this argument must be zero
dword newsize_lo	Low 32-bit word of the 64-bit offset from the beginning of the file

DESCRIPTION

pc_efilio_chsize() Changes the file to the new size requested in the pair newsize_hi, newsize_lo. The file size is changed immediately and operations on the file may continue. The operation occurs without making any disk accesses. To update the disk the file must be closed or flushed. If the file is truncated it must be closed or flushed to return the freed clusters to the pool of available clusters.

RETURNS

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

Application Level Error Return Codes

PEBADF	Invalid file descriptor
PECLOSED	File is no longer available. Call pc_efilio_close().
PEEFIOILLEGALFD	The file not open in extended IO mode
PEINVALIDPARMS	Missing or invalid parameters
PEEINPROGRESS	Asynchronous operation already in progress
An Rtfs system error	See Appendix for a description

pc_efilio_extract

Basic		ProPlus	x
Pro		ProPlus DVR	x

FUNCTION

Extract clusters from one 32-bit or 64-bit file and insert them into another.

SUMMARY

BOOLEAN **pc_efilio_extract** (fd1, fd2, n_clusters)

int fd1	A file descriptor that was returned from a successful call to **pc_efilio_open()**
int fd2	A file descriptor that was returned from a successful call to **pc_efilio_open()**
dword n_clusters	Number of clusters to extract. Use zero to extract to the end of file 1.

DESCRIPTION

pc_efilio_extract() removes the clusters from the file at fd1, inserts them into the file at fd2 and adjusts both file sizes. This operation performs no disk accesses so it performs in real time.

- fd1 may be either a 32 bit or 64 bit file
- fd2 may be either a 32 bit or 64 bit file
- If the current file pointer for fd1 is zero the clusters are removed from the beginning of the file.
- If the current file pointer for fd1 is non-zero the clusters are removed starting with the first cluster beyond the cluster containing the current file pointer.
- If the current file pointer for fd2 is zero the clusters are inserted at the beginning of file two.
- If the current file pointer for fd2 is non-zero the clusters are inserted starting with the first cluster beyond the cluster containing the current file pointer.
- If the current file pointer for fd2 is the last cluster in the file, the clusters are appended to the end of file two.
- The file size of fd1 is reduced.
- The file size of fd2 is increased.
- Both files must be closed or flushed for the changes to be commited to the volume

RETURNS

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

Application Level Error Return Codes

PEBADF	Invalid file descriptor
PECLOSED	File is no longer available. Call **pc_efilio_close()**.
PEEFIOILLEGALFD	The file not open in extended IO mode
PEINVALIDPARMS	Missing or invalid parameters
PETOOLARGE	The result of the extract would make the file at fd2 too large. If fd2 is a 32 bit file, this means the result would exceed 4 gigabytes. If fd2 is a 64 bit file, this means the result would exceed the configured maximimum 64 bit file size.
PEEINPROGRESS	Asynchronous operation already in progress
An Rtfs system error	See Appendix for a description

pc_efilio_swap

Basic		ProPlus	x
Pro		ProPlus DVR	x

FUNCTION
Swap clusters between one 32-bit or 64-bit file and another.

SUMMARY

BOOLEAN **pc_efilio_swap** (fd1, fd2, n_clusters)

int fd1	A file descriptor that was returned from a successful call to **pc_efilio_open()**
int fd2	A file descriptor that was returned from a successful call to **pc_efilio_open()**
dword n_clusters	Number of clusters to swap. Use zero to swap to the end of file 1.

DESCRIPTION

pc_efilio_swap() exchanges clusters from the file at fd1 with clusters in the file at fd2. The file sizes are unchanged. This operation performs no disk accesses so it performs in real time.

- fd1 may be either a 32 bit or a 64 bit file
- fd2 may be either a 32 bit or a 64 bit file
- If the current file pointer for fd1 is zero the clusters are swapped from the beginning of the file.
- If the current file pointer for fd1 is non-zero the clusters are swapped starting with the first cluster beyond the cluster containing the current file pointer.
- If the current file pointer for fd2 is zero the clusters are swapped from the beginning of file two.
- If the current file pointer for fd2 is non-zero the clusters are swapped starting with the first cluster beyond the cluster containing the current file pointer.
- Both files must be closed or flushed for the changes to be commited to the volume

RETURNS

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

Application Level Error Return Codes

PEBADF	Invalid file descriptor
PECLOSED	File is no longer available. Call pc_efilio_close().
PEEFIOILLEGALFD	The file not open in extended IO mode
PEINVALIDPARMS	Missing or invalid parameters
PETOOLARGE	The number of clusters to swap is larger than the number of clusters available at the current file pointer of fd1 or fd2.
PEEINPROGRESS	Asynchronous operation already in progress
An Rtfs system error	See Appendix for a description

pc_efilio_remove

Basic		ProPlus	x
Pro		ProPlus DVR	x

FUNCTION

Extract clusters from a 32 or 64 bit file and free them.

SUMMARY

BOOLEAN **pc_efilio_remove** (fd, n_clusters)

int fd	A file descriptor that was returned from a successful call to pc_efilio_open
dword n_clusters	Number of clusters to remove. Use zero to remove to the end of the file.

DESCRIPTION

pc_efilio_remove() removes the clusters from the file at fd and queues them to be freed when the file is flushed or closed. This operation performs no disk accesses so it performs in real time.

- If the current file pointer for fd is zero the clusters are removed from the beginning of the file.
- If the current file pointer for fd is non-zero the clusters are removed starting with the first cluster beyond the cluster containing the current file pointer.
- The file size of fd is reduced.
- The file must be closed or flushed for the clusters to be available for re-use and for the changes to be commited to the volume

RETURNS

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

Application Level Error Return Codes

PEBADF	Invalid file descriptor
PECLOSED	File is no longer available. Call **pc_efilio_close()**.
PEEFIOILLEGALFD	The file not open in extended IO mode
PEINVALIDPARMS	Missing or invalid parameters
PETOOLARGE	The number of cluters to remove is larger than the number of clusters available at the current file pointer of fd1.
PEEINPROGRESS	Asynchronous operation already in progress
An Rtfs system error	See Appendix for a description

pc_efilio_flush

Basic		ProPlus	x
Pro		ProPlus DVR	x

FUNCTION

Flush an extended 32 bit file or a 64 bit metafile

SUMMARY

BOOLEAN **pc_efilio_flush** (fd)

int fd	A file descriptor that was returned from a successful call to **pc_efilio_open()**

DESCRIPTION

Flush the directory entry and flush the fat chain to disk. Process any deferred cluster chain linking and free all memory associated with the file descriptor.

NOTE: An asynchronous file flush routine is also available, see ***pc_efilio_async_flush_start()***

NOTE: As a rule for very large files, both synchronous and asynchronous file re-opens, file flushes, file truncates and file deletes complete faster as user buffer space is increased. **pc_efilio_setbuf()** *is available to temporarily increase buffer space to speed up this operation.*

RETURNS

TRUE	The operation was a success
FALSE	Bad drive id requested or bad parameters

Application Level Error Return Codes:

PEBADF	Invalid file descriptor
PEEFIOILLEGALFD	The file not open in extended IO mode
PEINVALIDPARMS	Missing or invalid parameters
PEEINPROGRESS	Asynchronous operation already in progress
An Rtfs system error	See Appendix for a description

pc_efilio_fstat

Basic		ProPlus	x
Pro		ProPlus DVR	x

FUNCTION

Report statistics on an extended 32-bit file or 64-bit metafile**.**

SUMMARY

BOOLEAN **pc_efilio_fstat** (fd, pestat)

int fd	A file descriptor that was returned from a successful call to **pc_efilio_open()**
RTFS_EFILIO_STAT *pestat	The address of an RTFS_EFILIO_STAT structure that will be filled in by this function

DESCRIPTION

pc_efilio_fstat() fills in the extended stat structure with information about the open file.

The extended stat structure contains the following fields:

dword minimum_allocation_size	Minimum number of bytes that will be pre-allocated at one time when the file is extended, by default this is the cluster size of the volume but it may be affected by the extended file open option “min_clusters_per_allocation”.
dword allocation_policy	These are the policy bits that were set in the allocation_policy field of the extended file open call. Please see the documentation for **pc_efilio_open()** and **pc_cfilio_open()** for a description of the allocation policy bits.
dword fragments_in_file	The numbers of separate disjoint fragments in the file.
dword first_cluster	First cluster in the file.
dword allocated_clusters	Clusters used for data in the file.
dword preallocated_clusters	Cluster used for data plus clusters pre-allocated for data in the file.
dword clusters_to_link	Count of clusters to be linked into a chain when the file is closed or flushed.
dword file_size_hi	The current file size in bytes. file_size_hi and file_size_lo are the high and low 32-bit words of the 64-bit file length. If the file is, a 32-bit file file_size_hi will always be zero. For a 64-bit metafile, file_size_hi will be non-zero if the file is over four gigabytes in length.

dword file_size_lo	
dword allocated_size_hi	The number of bytes currently allocated to the file including the file contents (current_file_size) and any additional blocks that were pre allocated due to minimum allocation guidelines. allocated_size_hi and allocated_size_lo are the high and low 32-bit words of the 64 bit allocated file size. If the file is a 32-bit file, allocated_size_hi will always be zero. For a 64-bit metafile, allocated_size_hi will be non-zero if the file is over four gigabytes in length.
dword allocated_size_lo	
dword file_pointer_hi	High and low, 32-bit words of the current file pointer.
dword file_pointer_lo	

REGION_FRAGMENT *pfirst_fragment[]	The list of fragments that make up the file. For 32 bit files there is only one valid element, pfirst_fragment[0], which contains a linked list of the file's disjoint cluster fragments. For 64 bit files there may be up to MAX_SEGMENTS_64 valid elements pfirst_fragment[0] to pfirst_fragment[MAX_SEGMENTS_64-1]. Each element contains a linked list of the individual file segments that make up the 64 bit metafile. It is useful to use for test and diagnostic procedures and to study file allocation patterns. These lists should not be manipulated.
	
	The region fragment structure is declared as follows:
	
	typedef struct region_fragment
	{
	 unsigned long start_location;
	 unsigned long end_location;
	  struct region_fragment *pnext;
	}
	REGION_FRAGMENT;

ERTFS_STAT stat_struct	This is a standard stat structure used by pc_stat and pc_fstat. The stat structure is documented in the table below.

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 exFAT, the high 32 bits of the file size

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

Application Level Error Return Codes

PEINVALIDPARMS	Missing or invalid parameters
PEEINPROGRESS	Asynchronous operation already in progress
PEEFIOILLEGALFD	API call not compatible file descriptor open method
An Rtfs system error	See Appendix for a description

RtfsProPlus - Asynchronous operations API

pc_async_continue

Basic		ProPlus	X
Pro		ProPlus DVR	X

FUNCTION

HEREHERE

Process functions queued for asynchronous completion.

SUMMARY

int **pc_async_continue**(int driveno, int target_state, int steps)

int driveno	Drive number (A: == 0, B: == 1, etc
int target_state	Level of processing to complete (see below)
int steps	Number of iterations, 0 == finish

DESCRIPTION

This routine must be called by applications to complete one or more passes on asynchronous routines that have been queued for completion by the subroutines described in this section.

pc_async_continue() may be used as a background asynchronous process manager when called by a background thread on a periodic basis or when called from a foreground thread periodically to process outstanding asynchronous operations.

pc_async_continue() may also be used to precisely manage the completion of asynchronous operations. **pc_async_continue()** may be used in this mode even if it is also being used in a periodic scheme. This usage of **pc_async_continue()** is useful in many circumstances, including:

An application normally calls **pc_async_continue()** periodically from a background thread, but it knows it is shutting down and must complete all asynchronous operations now.

An application normally calls **pc_async_continue()** periodically from a background thread, but it is critical now for the application to ensure that the session's view of the volume is committed to the Journal file and is persistent.

An application normally calls **pc_async_continue()** periodically from a background thread, but it is currently performing an asynchronous file operation that it needs to complete before it can proceed.

How to use **pc_async_continue()**:

Select the step count - The step count tells **pc_async_continue()** how many iterations to loop for before returning. If the step count is zero the routine loops until the target state is reached. Each loop will execute at most one disk read or one disk write, so if **pc_async_continue()** is called with a step count of one, the application can be sure that only one disk access will occur per call.

Select the target_state - When **pc_async_continue()** is called it executes up to the number of iterations specified by the step

Possible target states:

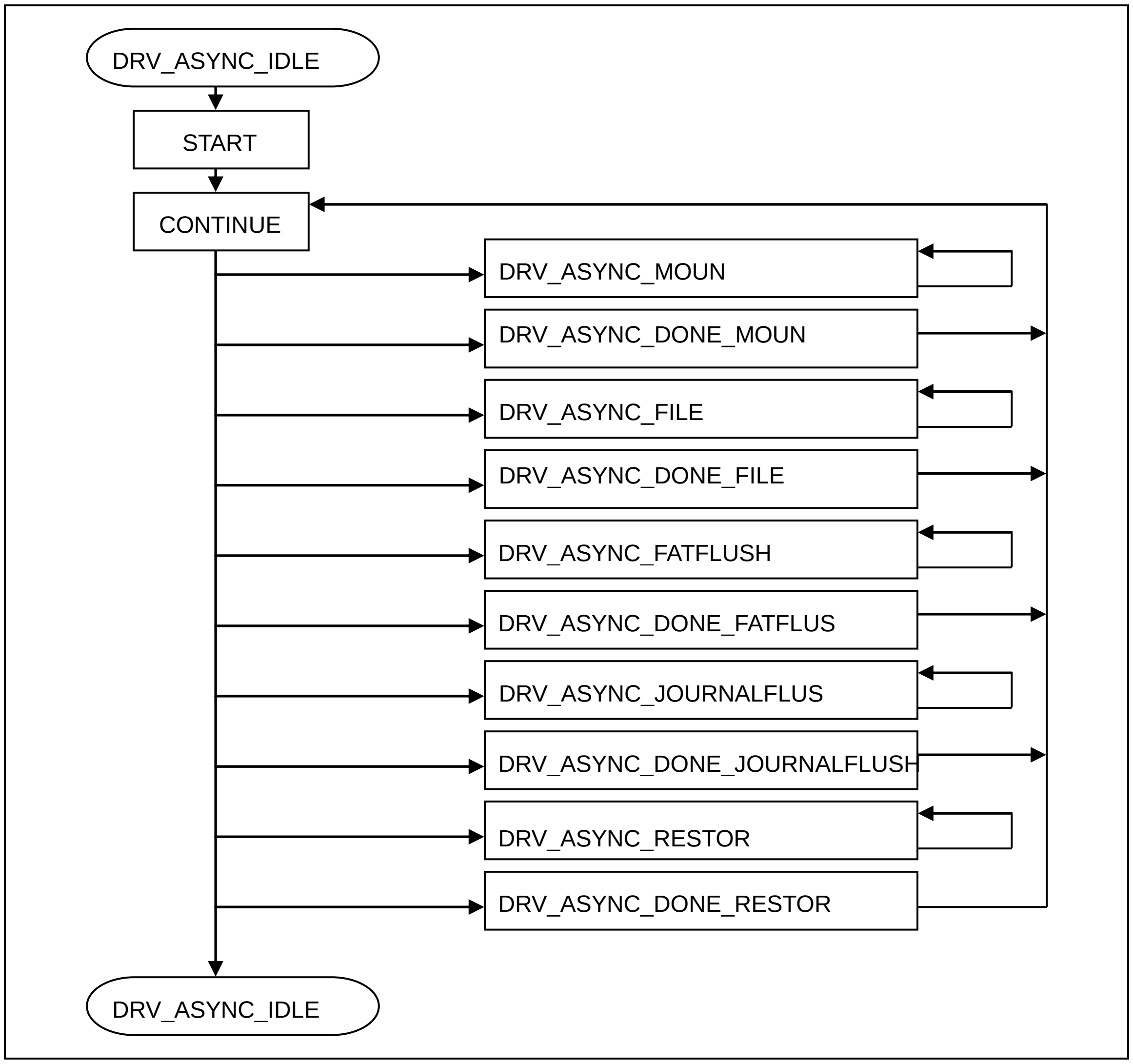
DRV_ASYNC_DONE_FILES	Process until all outstanding asynchronous file operations complete.
DRV_ASYNC_DONE_FATFLUSH	Process until all outstanding asynchronous file operations complete, and FATS are flushed.
DRV_ASYNC_DONE_JOURNALFLUSH	Process until all outstanding asynchronous file operations complete, FATS are flushed and the Journal file is flushed.
DRV_ASYNC_DONE_RESTORE	Process until all outstanding asynchronous file operations complete, FATS are flushed, the Journal file is flushed and the FAT volume is synchronized with the journal.
DRV_ASYNC_IDLE	Process until all outstanding asynchronous operations complete

Note: By default **pc_async_continue()** *does not enter any state greater than or equal to* **DRV_ASYNC_FATFLUSH***. In other words FAT flushing, Journal flushing and FAT synchronization are not automatically part of the asynchronous processing. To enable these states you must set the following policy bits in* **device_configure_*volume***.

DRVPOL_ASYNC_AFFLUSH	Enable background FAT flushing
DRVPOL_ASYNC_AJFLUSH	Enable background Journal flushing
DRVPOL_ASYNC_AJRESTORE	Enable background volume synchronization

Which steps to be completed can still be controlled when these bits are set by varying the target_state variable that is passed to **pc_async_continue()**. For example even if these bits are set, calling **pc_async_continue()** with target_state equal to **DRV_ASYNC_DONE_FILES** will result in file operations being completed but will not flush the FAT. Calling the routine later with **target_state** set to **DRV_ASYNC_DONE_FATFLUSH** will flush the FAT.

pc_async_continue: State Diagram



Example:

```
/* Execute one iteration per 100 Miliseconds */
for (;;) {
    pc_async_continue(0, DRV_ASYNC_IDLE, 1);
    Sleep(100);
}
/* Execute all iterations needed to complete all outstanding file operations */
pc_async_continue(0, DRV_ASYNC_DONE_FILES, 0);

/* Execute all iterations needed to complete all outstanding file operations */
/* and Flush the FAT buffers */
pc_async_continue(0, DRV_ASYNC_DONE_FATFLUSH, 0);

/* Execute all iterations needed to complete all outstanding file operations */
/* and Flush the FAT buffers */
/* and Flush the Journal file */
pc_async_continue(0, DRV_ASYNC_DONE_JOURNALFLUSH, 0);

/* Execute all iterations needed to complete all outstanding file operations */
/* and Flush the FAT buffers */
/* and Flush the Journal file */
/* and synchronize the volume with the Journal file */
pc_async_continue(0, DRV_ASYNC_DONE_RESTORE, 0);
```

RETURNS

PC_ASYNC_COMPLETE	Target state succesfully reached
PC_ASYNC_CONTINUE	Target state not reached in step_count iterations, continue calling pc_async_continue()
PC_ASYNC_ERROR	An error Target state not reached in step_count iterations, continue calling pc_async_continue()

pc_diskio_async_flush_start

Basic		ProPlus	x
Pro		ProPlus DVR	x

FUNCTION

HEREHERE

Non-blocking disk flush.

SUMMARY

int **pc_diskio_async_flush_start** (driveid**)**

byte *driveid	Name of a mounted volume “A:”, “B:” etc.

DESCRIPTION

Schedule modified blocks in the FAT buffer pool to be flushed by **pc_async_continue()** when it is next executed. Each pass through **pc_async_continue()** causes at most one write to occur. The number of passes required and the blocks written per pass depend on user buffer size and drive configuration options. See application notes and the manual pages for **pc_diskio_config()** and **pc_async_continue()** for more information of FAT buffering strategies.

Notes:

- Use of this function is not recommended. Instead it is recommended that you configure the disk with the policy **DRVPOL_ASYNC_AFFLUSH** enabled (see **device_configure_*volume***). Then asynchronous FAT flushing is enabled by default and if you wish to control when flushing occurs you may still do so by controlling the arguments to **pc_async_continue()**.
- **rtfs_app_callback**(**RTFS_CBA_ASYNC_FILE_COMPLETE**) is called when the flush operation completes or if it fails. The first argument to the callback contains **DRV_ASYNC_FATFLUSH** and the second argument contains 1 if successful, zero otherwise.
-

RETURNS

PC_ASYNC_CONTINUE	One or more blocks were successfully flushed but more calls are needed to complete the flush.
PC_ASYNC_COMPLETE	The operation was a success, no flush needed.
PC_ASYNC_ERROR	The operation failed. errno reflects the error condition**.**

Application Level Error Return Codes

PEINVALIDDRIVEID	Not a valid drive identifier
PEEINPROGRESS	Asynchronous operation already in progress
PEIOERRORWRITEFAT	IO error writing to the FAT
PEIOERRORREADINFO32	IO error reading the info block (FAT32 only)
PEIOERRORWRITEINFO32	IO error writing the info block (FAT32 only)

An Rtfs system error	See Appendix for a description

pc_diskio_async_mount_start

Basic		ProPlus	x
Pro		ProPlus DVR	x

FUNCTION

HEREHERE

Start a non-blocking disk mount

SUMMARY

int **pc_diskio_async_mount_start** (driveid)

byte *driveid	Name of the volume "A:", "B:" etc.

DESCRIPTION

A normal synchronous disk mount performs three initial disk accesses to read the master boot record, bios parameter block and, for FAT32 volumes, the FAT32 info sector. It then scans the file allocation table building a free map. On larger volumes, the FAT may contain thousands of disk blocks so the mount procedure may not return for several seconds while it scans the FAT. The asynchronous disk mount separates the three initial reads into one call and the subsequent FAT scan operation into multiple calls that are completed by **pc_async_continue()**. With this method, the system can continue to perform useful work while the disk is being mounted.

Note: You can eliminate the need to call **pc_diskio_async_mount_start()** *and simplify using asynchronous mounts with removable media by configuring Rtfs to perform asynchronous mounts automatically by returning the value 1 when* **rtfs_app_callback**(**RTFS_CBA_ASYNC_MOUNT_CHECK**) *is called. Then Rtfs call errno is set to* **PENOTMOUNTED** *and the API call fails. Before returning Rtfs calls* **pc_diskio_async_mount_start()** *to start a mount and then calls* **rtfs_app_callback**(**RTFS_CBA_ASYNC_START**) *to inform the application it should be certain the foreground process or a background process is cycling* **pc_async_continue()***. The drive will not be accessible to the API until* **pc_async_continue()** *completes the mount processing. Rtfs calls* **rtfs_app_callback**(**RTFS_CBA_ASYNC_DRIVE_COMPLETE**) *when the mount process is complete*

Notes: If any other API call is attempted on the drive before the asynchronous mount is completed the call will fail and errno will be set to **PEEINPROGRES***S.*

Example:

```
if (pc_diskio_async_mount_start ((byte *) "A:")  != PC_ASYNC_ERROR)
{
int driveno = pc_drname_to_drno((byte *) "A:");
do {
        /* Process other application needs here */

        /* Process one iteration of the mount process */
        rval = pc_async_continue(driveno, DRV_ASYNC_IDLE, 1);
} while (rval == PC_ASYNC_CONTINUE);
```

RETURNS

PC_ASYNC_CONTINUE	Mount start completed. Now call pc_async_continue.
PC_ASYNC_ERROR	The operation failed. errno reflects the error condition**.**

Application Level Error Return Codes

PEINVALIDDRIVEID	Not a valid drive identifier
PEEINPROGRESS	Asynchronous operation already in progress
PEINVALIDBPB	No signature found in BPB (please format)
PEINVALIDMBR	No signature found in MBR (please write partition)
PEINVALIDMBROFFSET	Partition requested but none at that offset
PEIOERRORREADMBR	IO error reading MBR (MBR read is the first access, indicates device not ready)
PEIOERRORREADBPB	IO error reading BPB (block 0)
PEIOERRORREADINFO32	IO error reading fat32 INFO structure (BPB extension)
An Rtfs system error	See Appendix for a description

pc_efilio_async_open_start

pc_efilio_async_open_start_uc

Basic		ProPlus	x
Pro		ProPlus DVR	x

FUNCTION

Start a non-blocking file open

SUMMARY

See the manual page for **pc_efilio_open()**.

DESCRIPTION

There really is no real **pc_efilio_async_open_start()** subroutine. This manual page is here to remind you that an asynchronous open operation is available. To open a file asynchronously call **pc_efilio_open()** and set the **PCE_ASYNC_OPEN** bit in the allocation policy field.
Note: As a rule for very large files, both synchronous and asynchronous file re-open's, file flushes, file truncates and file deletes will complete faster as user buffer space is increased.

Example:

```
PCFD fd;
EFILEOPTIONS my_options;

rtfs_memset(&my_options, 0, sizeof(my_options));
my_options.allocation_policy |= option;
my_options.allocation_policy |= PCE_ASYNC_OPEN;
fd = pc_efilio_open((byte *)"MYFILE",
                 (word)(PO_BINARY|PO_RDWR|PO_TRUNC|PO_CREAT),
                  ,(word)(PS_IWRITE | PS_IREAD)
                  ,&my_options);

/* Complete asynchronously */
if (fd>=0)
{
int rval,my_driveno;
byte my_drivename[8];

   my_driveno = pc_fd_to_driveid(fd, my_drivename);
    do {
        rval = pc_async_continue(my_driveno, DRV_ASYNC_DONE_FILES, 1);
        /* Do other useful application work here */
    } while (rval == PC_ASYNC_CONTINUE);
}
```

pc_efilio_async_close_start

Basic		ProPlus	x
Pro		ProPlus DVR	x

FUNCTION

Start a non-blocking file close.

SUMMARY

int **pc_efilio_async_close_start** (int fd)

int fd	A file descriptor that was returned from a successful call to **pc_efilio_open().**

DESCRIPTION

Initiate flush of a file's cluster map to the file allocation table followed by a flush of file's directory entry, followed by a release of the file descriptor. **pc_efilio_async_close_start()** behaves almost identically to **pc_efilio_async_flush_start()** except that when the asynchronous close completes the file descriptor is released.

Proper use of user buffering to optimize the asynchronous flush procedure improves overall system performance. So please note the following:

pc_async_continue() performs only one write operation per iteration. Either the number of contiguous clusters in the chain or the size of the assigned user buffer, whichever is less, limits the number of cluster entries written per iteration. Since Rtfs files are typically contiguous, the size of the user buffer is usually the limiting factor in maximizing the performance of the flush operation.

Note: As a rule for very large files, both synchronous and asynchronous file re-open's, file flushes, file truncates and file deletes will proceed and will complete faster as user buffer space is increased

For a FAT32 volume with a cluster size of 32768 bytes, 256 Kbytes of FAT table space are occupied per 1 gigabyte of data. A flush of a one gigabyte file requires one iteration if 256 K of user buffering space is provided, 2 iterations if 128 K is provided, or 4 iterations if 64 K is provided and so on. On most media, larger user buffers during asynchronous flushes will improve performance.

Example:

```
/* Sample function to close a file asynchronously */
void do_asy_close(int fd)
{
   In my_driveno;
   byte my_drivename[8];

   my_driveno = pc_fd_to_driveid(fd, my_drivename);

   if (pc_efilio_async_close_start(fd) == PC_ASYNC_CONTINUE)
  {
   int rval;
      do {
         rval = pc_async_continue(my_driveno, DRV_ASYNC_DONE_FILES, 1);
         /* Do other useful application work here */
      } while (rval == PC_ASYNC_CONTINUE);
   }
}
```

RETURNS

PC_ASYNC_COMPLETE	Asynchronous operation completed
PC_ASYNC_CONTINUE	No failure has occurred. Continue calling **pc_async_continue().**
PC_ASYNC_ERROR	Could not initiate the flush operation because of some sort of parameter problem. Consult errno for the cause. A call to **pc_async_continue()** will fail

Application Level Error Return Codes

PEEINPROGRESS	Asynchronous operation already in progress
PEEFIOILLEGALFD	Not opened as an extended file
An Rtfs system error	See Appendix for a description

pc_efilio_async_flush_start

Basic		ProPlus	x
Pro		ProPlus DVR	x

FUNCTION

Start a non-blocking file flush

SUMMARY

int **pc_efilio_async_flush_start** (int fd)

int fd	A file descriptor that was returned from a successful call to **pc_efilio_open()**.

DESCRIPTION

Initiate flush of a file's newly allocated clusters to the file allocation table. After the file has been been flushed, it may continue to be used or it may be closed.

After **pc_efilio_async_flush_start()** has been called you must complete the flush operation by calling **pc_async_continue()** until it no longer returns **PC_ASYNC_CONTINUE**. While the flush is in progress no API calls for this file descriptor will succeed.

Proper use of user buffering to optimize the asynchronous flush procedure improves overall system performance. So please note the following:

pc_async_continue() performs only one write operation per iteration. Either the number of contiguous clusters in the chain or the size of the assigned user buffer, whichever is less, limits the number of cluster entries written per iteration. Since Rtfs files are typically contiguous, the size of the user buffer is usually the limiting factor in maximizing the performance of the flush operation.

Note: As a rule for very large files, both synchronous and asynchronous file re-open's, file flushes, file truncates and file deletes will complete faster as user buffer space is increased.

For a FAT32 volume with a cluster size of 32768 bytes, 256 Kbytes of FAT table space are occupied per 1 gigabyte of data. A flush of a one gigabyte file requires one iteration if 256 K of user buffering space is provided, 2 iterations if 128 K is provided, or 4 iterations if 64 K is provided and so on. On most media, larger user buffers during asynchronous flushes will improve performance.

Example:

```
 /* Sample function to flush a file asynchronously */
void do_asy_flush(int fd)
{
   int my_driveno;
   byte my_drivename[8];

   my_driveno = pc_fd_to_driveid(fd, my_drivename);
   if (pc_efilio_async_flush_start(fd) == PC_ASYNC_CONTINUE)
  {
   int rval;
      do {
         rval = pc_async_continue(my_driveno, DRV_ASYNC_DONE_FILES, 1);
         /* Do other useful application work here */
      } while (rval == PC_ASYNC_CONTINUE);
   }
}
```

RETURNS

PC_ASYNC_COMPLETE	Asynchronous operation completed
PC_ASYNC_CONTINUE	No failure has occurred. Continue calling **pc_efilio_async_continue().**
PC_ASYNC_ERROR	Could not initiate the flush operation because of some sort of parameter problem. Consult errno for the cause. A call to **pc_async_continue()** will fail.

Application Level Error Return Codes

PEEINPROGRESS	Asynchronous operation already in progress
PEEFIOILLEGALFD	Not opened as an extended file
An Rtfs system error	See Appendix for a description

pc_efilio_async_unlink_start

pc_efilio_async_unlink_start_uc

Basic		ProPlus	x
Pro		ProPlus DVR	x

FUNCTION

Start a non-blocking file delete.

SUMMARY

int **pc_efilio_async_unlink_start** (byte *filename)

byte *filename	The name of the file to delete.

DESCRIPTION

Initiate deleting a file. The whole process involves loading the file's chain representation from the FAT into an internal form and then generating the disk write operations necessary to overwrite the on-disk cluster chains with zeroes, and finally, clearing the directory entry.

Note: After **pc_efilio_async_unlink_start()** *has been called you must complete the flush operation by calling* **pc_async_continue()** *until it no longer returns* **PC_ASYNC_CONTINUE***. While the flush is in progress no API calls other than* **pc_async_continue()** *will succeed on the file.*

Proper use of user buffering to optimize the asynchronous flush procedure improves overall system performance. So please note the following:

pc_async_continue() performs only one read or write operation per iteration. Either the number of contiguous clusters in the chain or the size of the assigned user buffer, whichever is less, limits the number of cluster entries transferred per iteration. Since Rtfs files are typically contiguous, the size of the user buffer is usually the limiting factor in maximizing the performance of the flush operation.

For a typical FAT32 volume that consumes 128 Kbytes of FAT table space per 1 gigabyte of data, a flush of a one gigabyte file requires only two iterations (one read/one write) if 128 K of user buffering space is provided, or 4 iterations if 64 K is provided, 8 iterations if 32 K is provided and so on. On most media, larger user buffers during asynchronous deletes will improve performance.

Note: As a rule for very large files, both synchronous and asynchronous file re-opens, file flushes, file truncates, and file deletes will complete faster as user buffer space is increased.

Example:

```
Void do_async_delete(byte *filename)
{
Int fd,rval,my_driveno;
byte my_drivename[8];

    fd = pc_efilio_async_unlink_start();
    if (fd >= 0)
    {
      my_driveno = pc_fd_to_driveid(fd, my_drivename);
      do {
         rval = pc_async_continue(my_driveno, DRV_ASYNC_DONE_FILES, 1);
         /* Do other useful application work here */
      } while (rval == PC_ASYNC_CONTINUE);
   }
}
```

RETURNS

A file descriptor to be passed to **pc_async_continue(),** -1 on an error

Application Level Error Return Codes:

PEACCES	Deleting an in-use object or writing to read only object
PENOENT	File or directory not found
PESHARE	Sharing violation
PEINVALIDPARMS	Missing or invalid parameters
PEINVALIDPATH	Invalid path name used as an argument
PEEINPROGRESS	Asynchronous operation already in progress
An Rtfs system error	See Appendix for a description

RtfsProPlusDVR - Circular File IO API

Introduction to circular files

Circular files are useful for Video and Data acquisition systems. This manual describes each circular file function and how to use it. Since circular files are slightly different from linear files, please note the following concepts before using the API:

- **Circular files act as ring buffers of data blocks.** When the file is read and written the file pointer advances until it reaches a user defined file wrap point and then wraps back to the beginning of the file.
- **Separate read and write file pointers are maintained.** These separate pointers act similarly to the separate input and output pointers presented in traditional memory based ring buffer schemes. The circular file presents a file based FIFO (first in first out) interface to the application.
- **Traditional file pointer view**. APIs are provided to seek the read and write pointers within the physical view of the file. In this view the file pointers are constrained to be between zero and the wrap point.
- **Stream file pointer view.** APIs are provided to seek the read and write pointers within a virtual or "stream" view of the file. In this view the file pointers are 64 bit offsets of the read and write file pointers from the origin. The first byte written to the file is at location zero and the last byte written to the file is at a location we will call END for this discussion. The file can only contain a fixed number of bytes determined by the wrap point, which we will call FILESIZE for this discussion. In stream view mode, the virtual end of the file corresponds to location END, the virtual beginning of the file is zero. The oldest data in the file corresponds to the virtual pointer at location END-FILESIZE. In this way stream view of the data file is a sliding-window that is FILESIZE bytes wide, starting at the 64 bit index of the oldest byte in the file. Stream view is useful for coordinating the current view of the file with the actual sequences that are being recorded. For example: A video recorder application may have a circular buffer that is only capable of holding one hour of video at a time, but in stream view when the file wraps, the stream view pointer does not. So for example in hour zero to one the stream view pointer appears as zero to (framesize * frames_per_hour), but in hour three to four the stream view pointer appears as (3 X framesize * frames_per_hour) to (4 X framesize * frames_per_hour). The file pointers may only reside within the current sliding window and stream view seek functions are provided to move both the read pointer and the write pointer within this window.
- **Circular buffer mode.** By default files are opened with the **PCE_CIRCULAR_BUFFER** attribute set. In this mode if the write file pointer catches and laps the read file pointer no exceptional processing occurs and the write operation continues, allowing data that was written but never read to be overwritten. This is useful for applications like PVRs where not all data is permanently archived.
- **Circular file mode.** Files may be opened with the **PCE_CIRCULAR_FILE** attribute set. In this mode if the write file pointer catches the read file pointer the write operation stops, writing only as many bytes that can fit in the buffer without overwriting data that has not been read. This is useful for traditional data acquisition applications where buffer overflows are

considered an error and indicate that the buffer must be made larger. The data processing or archiving rate must be increased or the rate of data acquisition must be reduced.

- **Determining the amount of unread data in the buffer.** This may be done by using the stream view seek functions to ascertain the current stream write pointer and the current stream read pointer. The amount of unread data in the buffer is the write pointer minus the read pointer.
- **Extracting linear portions from the file.** An extract function is provided to allow extracting regions from the circular file into permanent linear files.
    - The linear extract files must be opened first with **pc_efilio_open()**, specifying the **PCE_REMAP_FILE** file option.
    - The extract process is typically a zero copy operation, whenever possible using cluster "soft linking" rather than data copies.
    - The extracted data exists in both the linear file and the circular file until the extract file is closed or the circular file write pointer overtakes the extract region.
    - The application is notified via a callback mechanism when the extract region has been overtaken, so it may close the extract file if it desires.
    - The extract file must be flushed or closed before becoming part of the disk image. They may be flushed immediately after the extract has occurred but they should not be closed (typically) until the region is overwritten.
    - If an extract file is closed before the write pointer overtakes the region of the buffer it was associated with, the data in that region of the circular file will be random, containing the contents of un-initialized or re-used disk data blocks.
    - Extracts fragment the circular file – The linear extract file occupies the original clusters from the circular file and the circular file extents in that regions are linked to newly allocated clusters.
- **The circular buffer may be a temporary file.** If the circular buffer is purely temporary it may be opened the **PCE_TEMP_FILE** attribute. This way when the file is closed its contents are quickly returned to free space and no flushing occurs.
- **Reopening circular files.** When circular files are reopened the stream view read and write pointers are both at zero. So there appears to be no data in the file. The write pointer may be advanced by either writing to the file passing a NULL pointer as the data pointer or by **pc_cfilio_lseek()**. This has the affect of making the stream view write pointer non-zero while leaving the read pointer at zero. This same scheme may be used to restore the stream view pointers to values larger than the physical size of the file by "writing", using a NULL pointer, as many bytes as necessary to restore the write stream view pointer to its previous value. Then seek on the read pointer to place it within the sliding window

pc_cfilio_open

pc_cfilio_open_uc

Basic		ProPlus	
Pro		ProPlus DVR	x

FUNCTION

Open a file for circular file IO operations

SUMMARY

int **pc_cfilio_open** (name, flag, poptions)

byte *name	File name
word flag	Flag values. Same as for po_open
EFILEOPTIONS *poptions	Extended options

DESCRIPTION

Open the file for access as a circular file with options specified in flag, with additional options specified in the poptions structure.

Note: **pc_cfilio_open()** *accepts many of the same options as* **pc_efilio_open()**. *Please also consult the documentation for* **pc_efilio_open()**.

Flag values are:

PO_BINARY	Ignored. All file access is binary
PO_TEXT	Ignored
PO_RDONLY	Open for read only
PO_RDWR	Read/write access allowed
PO_WRONLY	Open for write only
PO_CREAT	Creates the file if it does not exist. Use mode to specify the permission on the file.
PO_EXCL	If flag contains (PO_CREAT | PO_EXCL) and the file already exists fail and set errno() to EXIST
PO_TRUNC	Truncate the file if it already exists
PO_NOSHAREANY	Fail if the file is already open. If the open succeeds, no other opens will succeed until it is closed.
PO_NOSHAREWRITE	Fail if the file is already open for write. If the open succeeds, no other opens for write will succeed until it is closed.
PO_AFLUSH	Flush the file after each write
PO_APPEND	Always seek to end of file before writing.
PO_BUFFERED	Use persistent buffers to improve performance of non-block aligned reads and writes.

Extended Options

Unlike **pc_efilio_open()** the options argument is required for **pc_cfilio_open()**. The options structure must be zeroed and its fields must be initialized properly before they are passed.

The options structure is:

typedef struct efileoptions {

dword allocation_policy;	
dword min_clusters_per_allocation;	*see pc_efilio_open*
dword allocation_hint;	*see pc_efilio_open*
byte *transaction_buffer;	*see pc_efilio_open*
dword transaction_buffer_size;	*see pc_efilio_open*
dword circular_file_size_hi;	
dword circular_file_size_lo;	
int n_remap_records;	
REMAP_RECORD *remap_records;	

} **EFILEOPTIONS;**

EFILEOPTIONS;

allocation_policy	This field contains bit flags that may be set by the user to modify the behavior of the extended file IO routines.

The following option, specific to circular files may be used:

PCE_CIRCULAR_FILE	Select this option to force **pc_cfilio_write()** calls to truncate the write operation rather than allow the write file pointer to overtake the read file pointer.
PCE_CIRCULAR_BUFFER	Select this option to allow **pc_cfilio_write()** calls to proceed even when the write file pointer overtakes the read file pointer. (If neither **PCE_CIRCULAR_FILE** or **PCE_CIRCULAR_BUFFER** are selected this is the default behavior)

The following allocation options (see* pc_efilio_open()*) may be used

PCE_TEMP_FILE
PCE_64BIT_META_FILE
PCE_SMALL_FILE
PCE_FIRST_FIT
PCE_FORCE_FIRST
PCE_FORCE_CONTIGUOUS
PCE_KEEP_PREALLOC

The following allocation options (see* pc_efilio_open()*) may not be used

PCE_ASYNC_OPEN
PCE_TRANSACTION_FILE
PCE_REMAP_FILE

These two fields, which must be zero for* pc_efilio_open()*, must be set for* pc_cfilio_open()*.

circular_file_size_hi	Set this value to the high 32 bits of the 64-bit file offset that defines the circular file wrap point. If the wrap point is, less than 4 gigabytes set this to zero.

Note: If circular_file_size_hi *is non- zero, the circular file operations require underlying 64-bit metafiles support. Therefore,* **pc_cfilio_open()** *sets the* **PCE_64BIT_META_FILE** *option bit automatically.*

circular_file_size_lo	Set this value to the low 32 bits of the 64-bit file offset that defines the circular file wrap point.

These two fields, which must be zero for* pc_efilio_open()*, must be set for* pc_cfilio_open()*, if* pc_cfilio_extract() *is going to be used.

If the application will call **pc_cfilio_extract()** to extract linear sections from the circular file then these two fields must provide storage to contain the file's remap records. One remap record is required per active extract file (an active extract file is a file descriptor that was used as an argument to **pc_cfilio_extract()** but has not yet been closed or overtaken by the write pointer). (See the Rtfs-Pro Plus application notes for a description and tutorial on using extract files).

n_remap_records	The number of remap records being provided in the space pointed to by remap_records
remap_records	Pointer to a user supplied buffer or array of structures of type REMAP_RECORD large enough to contain n_remap_records structures. The **REMAP_RECORD** is a small structure (approximately 32 bytes).

<u>RETURNS</u>

>= 0	The operation was a success the return value is a valid file descriptor
-1	The operation failed consult errno

Application Level Error Return Codes:

PEACCES	Deleting an in-use object or writing to read only object
PEEXIST	Creating an object that already exists
PENOENT	File or directory not found

PENOSPC	Out of space to perform the operation
PESHARE	Sharing violation
PEINVALIDPARMS	Missing or invalid parameters
PEINVALIDPATH	Invalid path name used as an argument
PEEINPROGRESS	Asynchronous operation already in progress
PEEFIOILLEGALFD	API call not compatible file descriptor open method
PE64NOT64BITFILE	Attempt to open a directory with 64 bit file API
An Rtfs system error	See Appendix for a description

pc_cfilio_setalloc

Basic		ProPlus	
Pro		ProPlus DVR	x

FUNCTION

Specify and optionally reserve clusters for a file.

SUMMARY

BOOLEAN **pc_cfilio_setalloc(**int fd, dword cluster, dword ntoreserve)

int fd	A file descriptor that was returned from a successful call to pc_cfilio_open
dword cluster	Hint for the next cluster to allocate, or start of clusters to reserve
int fd	Reserve this many clusters, or zero

DESCRIPTION

pc_cfilio_setalloc() allows the programmer either specify a hint where the next cluster should be allocated from or to specify a group of clusters to be pre-allocated to this file for its exclusive use.
If ntoreserve is non-zero then clusters in the range cluster to cluster + ntoreserve - 1 are removed from free space and added to the file's reserved cluster list. When the file is expanded, these clusters are used. When the file is closed, any unused clusters in the reserve list are released. If all of the specified clusters are not currently free then **pc_cfilio_setalloc()** fails and sets errno to **PEINVALIDPARMS**.

If ntoreserve is zero, the clusters are not pre-allocated but when the file is next expanded, Rtfs tries to allocate clusters starting at cluster. If cluster is already in use, it allocates starting at the next free cluser beyond cluster.

*Note: **pc_diskio_free_list()** may be used in conjunction with **pc_cfilio_setalloc()** to retrieve a free cluster map and assign specific clusters from that map to files.*

RETURNS

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

Application Level Error Return Codes

PEBADF	Invalid file descriptor
PECLOSED	File is no longer available. Call pc_cfilio_close().
PEEFIOILLEGALFD	The file not open in extended IO mode
PEINVALIDPARMS	Missing or invalid parameters
PEEINPROGRESS	Asynchronous operation already in progress
An Rtfs system error	See Appendix for a description

pc_cfilio_close

Basic		ProPlus	
Pro		ProPlus DVR	x

FUNCTION

Close an open circular file

SUMMARY

BOOLEAN **pc_cfilio_close** (int fd)

int fd	A file descriptor that was returned from a successful call to **pc_cfilio_open()**.

DESCRIPTION

Flush the directory and FAT to disk, process any deferred cluster maintenance operations and free all memory associated with the file descriptor.

Note: If there are still any remap regions attached to the circular file when it is closed they are released and the release callback function is called, as if they were released because of a region overwrite.

RETURNS

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

Application Level Error Return Codes

PEBADF	Invalid file descriptor
PEEFIOILLEGALFD	The file not open in circular IO mode
PEINVALIDPARMS	Missing or invalid parameters
PEEINPROGRESS	Asynchronous operation already in progress
An Rtfs system error	See Appendix for a description

pc_cfilio_read

Basic		ProPlus	
Pro		ProPlus DVR	x

FUNCTION

Read from a circular file

SUMMARY

BOOLEAN **pc_cfilio_read** (fd, buf, count, nread)

int fd	A file descriptor that was returned from a successful call to **pc_cfilio_open()**
dword count	The length of the read request, (0 to 0xffffffff)
byte *buf	Buffer where data is to be placed. If buf is a null pointer **pc_cfilio_read()** will proceed as usual but it will not transfer bytes to the buffer
dword *nread	Returns the number of bytes read

DESCRIPTION

Read forward from the current pointer count bytes or until the last valid data byte is encountered, whichever is less.

If the region of the circular file to be read contains remapped sections the data is transferred from the remap file rather than the blocks of the circular file itself.

If the read pointer crosses (laps) the file wrap point during the read operation then the underlying physical linear file pointer as reported by **pc_cfilio_lseek()** is reset to zero. The logical stream pointer as reported by **pc_cstreamio_ lseek()** continues to increase. **pc_efilio_read()** uses underlying extended file read functions so it has the same zero disk latency file extent tracking behavior.

RETURNS

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

Error Return Codes:

PEBADF	Invalid file descriptor
PECLOSED	File is no longer available. Call pc_cfilio_close().
PEEFIOILLEGALFD	The file was not opened as a circular file.
PEINVALIDPARMS	Missing or invalid parameters.
PEEINPROGRESS	Asynchronous operation already in progress.
An Rtfs system error	See Appendix for a description

pc_cfilio_write

Basic		ProPlus	
Pro		ProPlus DVR	x

FUNCTION

Write to a circular file

SUMMARY

BOOLEAN **pc_cfilio_write**(fd, buf, count, nwrite)

int fd	A file descriptor that was returned from a successful call to pc_cfilio_open.
dword count	The length of the write request, (0 to 0xffffffff) *NOTE: If count is zero no data is transferred and the file pointer is not moved, but the write routine verifies that clusters are allocated at the file pointer This feature plays a key role in DMA enabling applications. Please consult section 5 and the application notes for more* **information.**
byte *buf	Buffer containing data is to be written. NOTE: If buf is a null pointer **pc_cfilio_write()** will proceed as usual but it will not transfer bytes to the buffer

DESCRIPTION

Write forward from the current write pointer, count bytes. If the circular file is not yet filled to the wrap point extend the underlying linear file using the allocation rules provided to **pc_cfilio_open()**. If the circular file is filled to the wrap point and the write file pointer reaches the wrap point operation then the underlying physical linear file pointer as reported by **pc_cfilio_lseek()** is reset to zero. The logical stream pointer as reported by **pc_cstreamio_ lseek()** continues to increase. **pc_cfilio_write()** uses underlying extended file write functions so it has the same zero disk latency file extent tracking and allocation behavior.

If the region of the circular file to be written contains remapped sections, the remapped sections are reduced by the amount to be written before the write takes place so the newly written data is contained in the extents owned by the circular file, and not the remap file. Subsequent reads from this section of the circular file will return the bytes just written and not bytes in the remap region. If a remapped section becomes completely overwritten it is released and the callback mechanism alerts the user that the remap file is no longer associated with the circular file.

If the circulator file was opened with the **PCE_CIRCULAR_FILE** allocation strategy rather than **PCE_CIRCULAR_BUFFER**, a write call will write less bytes than requested if the write pointer catches the read pointer, a short write return value may be used to signal to your application that the buffer offload function is not keeping up with the buffer load functions.

RETURNS

TRUE	If no errors were encountered. *nwrite is set to the number of bytes successfully written**.**
FALSE	An error occurred

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

PEBADF	Invalid file descriptor
PECLOSED	File is no longer available. Call **pc_cfilio_close()**.
PEINVALIDPARMS	Bad or missing argument
PEACCES	File is read only
PEIOERRORWRITE	Error performing write
PEIOERRORREADBLOCK	Error reading block for merge and write
PENOSPC	Disk to full to allocate file minimum allocation size.
PEEFIOILLEGALFD	The file was not opened as a circular file.
PETOOLARGE	Attempt to extend a 32-bit file beyond 4 gigabytes
PERESOURCEREGION	Ran out of region structures while performing operation
An Rtfs system error	See Appendix

pc_cfilio_lseek

Basic		ProPlus	
Pro		ProPlus DVR	x

FUNCTION

Move linear read or write file pointer of a circular file

SUMMARY

BOOLEAN **pc_cfilio_lseek** (fd, which, off_hi, off_lo, origin, *poff_hi, *poff_lo)

int fd	A file descriptor that was returned from a successful call to **pc_cfilio_open()**
int which_pointer	Which file pointer read or write
CFREAD_POINTER	Move the read file pointer
CFWRITE_POINTER	Move the write file pointer
dword offset_hi	High 32-bit word of the 64 bit offset from the beginning of the file. For 32 bit, files must be zero.
dword offset_lo	Low 32 bit word of the 64 bit offset from the beginning of the file.
int origin	Origin and direction of the request (see below)
dword *poffset_hi	The high dword of the new 64 bit offset from the beginning of the underlying linear file is returned in *poffset_lo.
dword *poffset_lo	The low dword of the new 64 bit offset from the beginning of the underlying linear file is returned in *poffset_hi.

DESCRIPTION

pc_cfilio_lseek() moves the linear read or write file pointer of the linear file underlying the circular file. This function selects the appropriate read or write file structure and then calls the underlying **pc_efilio_lseek()** to move the file pointer according to the rules of **pc_efilio_lseek()**.

The file pointer is not the same as the stream pointer that is used by **pc_cstreamio_lseek()**. The file pointer may not exceed the current file length and at steady state, this maximum will be the circular file wrap point that was provided when the file was opened. The stream pointer is not bounded like the file pointer; it begins at zero and extends to the 64-bit value 0xffffffffffffffff. At a given time, valid values for the stream pointer are contained in a sliding window that extends backwards to the last logical offset written to, minus the size of the circular file. If the file has not yet reached the wrap point, the window extends from zero to the wrap point.

The file pointer is set according to the following rules:

PSEEK_SET	offset from beginning of file
PSEEK_CUR	positive offset from current file pointer

PSEEK_CUR_NEG	negative offset from current file pointer
PSEEK_END	0 or negative offset from end of file

If a **PSEEK_CUR** operation attempts to move the file pointer beyond the end of file, the pointer is moved to the end of file.

If a **PSEEK_CUR_NEG** or **PSEEK_END**, operation tries to place the file pointer before zero the file pointer is placed at zero.

To query the current file pointer call:

pc_cfilio_lseek(fd, CFREAD_POINTER,0, 0, PSEEK_CUR, &offset_hi, &offset_lo)

Or

pc_cfilio_lseek(fd, CFWRITE_POINTER,0, 0, PSEEK_CUR, &offset_hi, &offset_lo)

<u>RETURNS</u>

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

Application Level Error Return Codes

PEBADF	Invalid file descriptor
PECLOSED	File is no longer available. Call **pc_cfilio_close()**.
PEEFIOILLEGALFD	The file was not opened as a circular file.
PEINVALIDPARMS	Missing or invalid parameters.
PEEINPROGRESS	Asynchronous operation already in progress.
An Rtfs system error	See Appendix for a description

pc_cstreamio_lseek

Basic		ProPlus	
Pro		ProPlus DVR	x

FUNCTION

Move read or write stream pointer of a circular file

SUMMARY

BOOLEAN **pc_cstreamio_lseek** (fd, which, off_hi, off_lo, origin, *poff_hi, *poff_lo)

int fd	A file descriptor that was returned from a successful call to **pc_cfilio_open()**
int which_pointer	Which file pointer read or write
CFREAD_POINTER	Move the read file pointer
CFWRITE_POINTER	Move the write file pointer
dword offset_hi	high dword 64 bit offset.
dword offset_lo	lo, dword 64 bit offset to move the file pointer from the specified origin. *Note: the stream pointer is a 64-bit value regardless of whether the underlying circular file is a 32-bit file or a 64-bit Meta file.*
int origin	Origin and direction of the request (see below)
dword *poffset_hi	Returned hi dword 64 bit offset.
dword *poffset_lo	Returned lo dword 64 bit offset, contains the new file pointer value after seek.

DESCRIPTION

pc_cstreamio_lseek() moves the read or write stream pointer of a circular file. The stream file pointer is the offset in the data stream that has been written to the circular file.

The stream pointer is not the same as the file pointer that is used by **pc_cfilio_lseek()**. The stream pointer is not bounded like the file pointer; it begins at zero and extends to the 64-bit value 0xffffffffffffffff. At a given time, valid values for the stream pointer are contained in a sliding window that extends backwards from the last logical offset written to, to the last logical offset written to minus the size of the circular file. If the file has not yet reached the wrap point, the window extends from zero to the wrap point.

If a **PSEEK_SET** operation attempts to move the read stream pointer beyond the end of the sliding window or before the sliding window, the function returns **FALSE** and errno is set to **PEINVALIDPARMS**.

To clear this condition and place the stream pointer at the beginning of the sliding window call:

```
 pc_cstreamio_lseek(fd, CFREAD_POINTER, 0xffffffff, 0xffffffff,
                          PSEEK_END, &offset_hi, &offset_lo)
```

This will place the stream pointer at the beginning of the sliding window. If a **PSEEK_CUR** operation attempts to move the stream pointer beyond the end of the sliding window, the pointer moves to the end of sliding window, (the last byte written).

If a **PSEEK_CUR_NEG** or **PSEEK_END** operation tries to place the stream pointer, before sliding window, the stream pointer positions at the beginning of the sliding window (the oldest data in the stream).

To query the current stream pointer call:

```
pc_cstreamio_lseek (fd, CFREAD_POINTER, 0, 0,
                           PSEEK_CUR, &offset_hi, &offset_lo)
```

Or

```
pc_cstreamio_lseek (fd, CFWRITE_POINTER, 0, 0,
                           PSEEK_CUR, &offset_hi, &offset_lo)
```

The end of the data stream is numerically the furthest offset from the origin where a byte has been written.

It is an error to seek beyond the end of the data stream or before the beginning of the sliding window location.

The behavior of the function with each origin is provided here:

ORIGIN	RULE
PSEEK_SET	Seek to the absolute location in the data stream. If the provided location is outside of the sliding window, return FALSE and set errno to PEINVALIDPARMS
PSEEK_CUR	Seek foreword in the data stream. If the provided offset plus the current stream pointer resides outside of the sliding window, place the stream pointer at the end of the data stream.
PSEEK_CUR_NEG	Seek backward in the data stream. If current stream pointer minus the provided offset precedes the sliding window, place the stream pointer at the beginning of the sliding window.
PSEEK_END	Seek backward in the data stream. If the current ends of the data stream minus the provided offset precedes the sliding window place the stream pointer at the beginning of the sliding window.

RETURNS

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

Application Level Error Return Codes

PEBADF	Invalid file descriptor
PECLOSED	File is no longer available. Call pc_cfilio_close().
PEEFIOILLEGALFD	The file cannot open in circular IO mode.
PEINVALIDPARMS	Missing or invalid parameters.
PEEINPROGRESS	Asynchronous operation already in progress.
An Rtfs system error	See Appendix for a description

pc_cfilio_extract

Basic		ProPlus	
Pro		ProPlus DVR	x

FUNCTION

Extract a region from a circular file to a linear extract file.

SUMMARY

BOOLEAN **pc_cfilio_extract**(circ_fd, linear_fd, length_hi, length_lo, header_buffer, header_size)

int circ_fd	A file descriptor that was returned from a successful call to pc_cfilio_open.
int linear_fd	A file descriptor that was returned from a successful call to pc_cfilio_open. The file must have been opened with the PCE_REMAP_FILE allocation attribute.
dword length_hi	These two fields are the high and low values of the length of the region to extract from the current logical read pointer in the circular buffer.
dword length_lo	
byte *header_buffer	Data passed in this buffer will be placed in reserved bytes before the start of data in the extracted file. If no header is desired pass 0. This data section may be read or written to by calling the **pc_efilio_lseek** using the origin **PSEEK_SET_RAW**.
int header_size	Length of the data in **header_buffer**, or 0.

Note: **pc_cfilio_extract()** *must sometimes copy data. When it does it uses the user buffer. So the size of the user buffer may impact the performance of* ***pc_cfilio_extract()****. User buffer space at least the size of a cluster is required.*

See the documentation for *device_configure_volume in the media driver callback* section of the API reference Guide for instructions on providing Rtfs with user buffering.

DESCRIPTION

This function unlinks the cluster chains in the range bounded by the current read file pointer and the current read file pointer plus length_hi:length_lo. It moves these clusters to the linear file, allocates new clusters from free space, and links these replacement clusters into the circular file where the extracted file was removed.

If the extracted region does not begin on a cluster boundary, the extracted file will be assigned a start offset equal to the offset into the first cluster and the file size will be this offset plus the size of the extracted data.

Under most circumstances, clusters are linked not copied. However, sometimes clusters must be copied.

If the beginning of the extract region does not lie on a cluster boundary, that cluster must be copied from the circular file to the extract file.

If the end of the extract region does not lie on a cluster boundary, that cluster must be copied from the circular file to the extract file.

If the extract region spans previously extracted and remapped regions then these overlapping blocks are copied and not re-linked.

After **pc_cfilio_extract()** completes the extracted region is remapped in the circular file. What this means is that future reads from the circular file in this region will return bytes from the extract file. Future writes to this region write, not to the extract file, but to replacement clusters in the circular file. If an area of the remap region is overwritten, the remap region is split or shrunk so future reads from that area will be from the circular file and not the remap file.

When the linear extract file closes the remap region is removed from the circular file. In this event, the contents of the region may be read but the data will be un-initialized until the region is overwritten.

Note: the following preconditions for using **pc_cfilio_extract()**.

The linear extract file: the file must have been opened with **pc_efilio_open()** using the **PCE_REMAP_FILE** allocation attribute and it must be an empty file.

pc_cfilio_extract() requires at least one block of user supplied buffering for copying clusters that must be copied instead of linked. The amount of buffering must be at least one block but it is a good idea for the user buffer size to be large enough to hold at least one cluster. If larger regions are being copied because of overlapping extract regions then even larger buffer will improve performance by reducing seeks.

When the circular file was opened **pc_cfilio_open()** must have been provided with enough remap structures to hold this remap extract region, and all other active remap regions. The number of remap regions present depends on the number of remap regions that have been created by calls to **pc_cfilio_extract()** and have not yet been released. Remap regions are purged when the write file pointer overwrites the region, when a new remap region covers all of the current remap region or when the file descriptor of the extract file is closed with **pc_efilio_close()**.

A call to **pc_cfilio_extract()** will consume one remap record if it either does not overlap an existing region or if it intersects or overlaps an existing region on one side. If the new region resides completely inside of another remap region then the existing region is split and two instead of one remap structures are consumed. If one new remap region completely overlaps one or more existing remap regions then those regions are released resulting in one or more remap records being made available again for reuse.

Remap records are relatively small (around 32 bytes) so make sure enough of them are provided.

See the description of **pc_cfilio_open()** for a description of the configuration values n_remap_records and remap_records.

Note: While the extract file is sharing a data region with the circular file, both files share a common view of the region when they read from it. The write behavior is different though. When the extract file is written to, the data is visible through the read call to both the circular file and the linear file. When the region is written to by the circular file the new data is visible only in the circular file, not in the linear file. If the linear file is closed before the region is fully overwritten by the circular file then the data regions are no longer shared. The read view from the linear file will be what was extracted, but the circular file's view of that region will be un-initialized data on disk blocks. For this reason it is not a good idea to close extract files until they have been overwritten. To aid in this process rtfs_app_callback(RTFS_CBA_DVR_EXTRACT_RELEASE) is called everytime an extract file is overwritten. This callback may be used to signal that the extract file may now be closed.

See the Application callbacks section of the manual for an explanation of user callbacks.

RETURNS

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

Application Level Error Return Codes

PEINVALIDPARMS	Missing or invalid parameters
PETOOLARGE	Extract File is not exFat but region size exceeds
PEEFIOILLEGALFD	API call not compatible file descriptor open method
PECFIONOMAPREGIONS	More open remap regions than provided for in **pc_cfilio_open()**
An Rtfs system error	See Appendix for a description