# Introduction

## File system filter driver

A file system filter driver intercepts requests targeted at a file system or another file system filter driver. By intercepting the request before it reaches its intended target, the filter driver can extend or replace functionality provided by the original target of the request. It is developed primarily to allow the addition of new functionality beyond what is currently available.

### File system monitor filter

File system monitor filter can monitor the file system activities on the fly. With file system monitor filter, you can monitor the file activities on file system level, capture file open/create/replace, read/write, query/set file attribute/size/time security information, rename/delete, directory browsing and file close request by which user and process name. You can develop the software for the following purposes:

* Continuous data protection (CDP).
* Auditing.
* Access log.
* Journaling.

### File system control filter

File system control filter can control the file activities, which you can intercept the file system call, modify its content before or after the request goes down to the file system, allow/deny/cancel its execution based on the filter rule. You can fully control file open/create/replace, read/write, query/set file attribute/size/time security information, rename/delete, directory browsing these Io requests. With file system control filter, you can developer these kinds of software:

* Data protection.
* File Screening
* Access Control.
* File Security.

### File system encryption filter

File system encryption filter provides a comprehensive solution for transparent file level encryption. It allows developers to create transparent, on-access, per-file encryption products which it can encrypt or decrypt file on-the-fly. Our encryption engine uses a strong cryptographic algorithm called Rijndael (256-bit key), it is a high security algorithm created by Joan Daemen and Vincent Rijmen (Belgium). Rijndael is the new Advanced Encryption Standard (AES) chosen by the National Institute of Standards and Technology (NIST).

### Process filter driver

Process filter driver is a kernel-mode driver that tracks and controls the process and thread operations, it provides you an easy way to develop Windows application for the Windows process monitoring and protection. Prevent the untrusted executable binaries (malwares) from being launched, protect your data being damaged by the untrusted processes.

### Registry filter driver

Registry filter driver is a kernel-mode driver that tracks and controls the registry access, it provides you an easy way to develop Windows application for registry monitoring and protection, it enables your application to prevent the Windows core registry keys and values from being damaged.

### The rules to use of file system control filter

To use the file system control filter, you need to follow the following rules, or might cause the system deadlock.

1. *Avoid the re-entrance issue, don’t generate any new I/O request which will cause the request comes to the control filter handler again.*
2. *Avoid using any file operations in buffered mode, open any file in the control filter handler with FILE\_FLAG\_NO\_BUFFERING flag set.*
3. *Avoid asynchronous procedure calls.*
4. *Avoid any GUI (user interface) operations.*

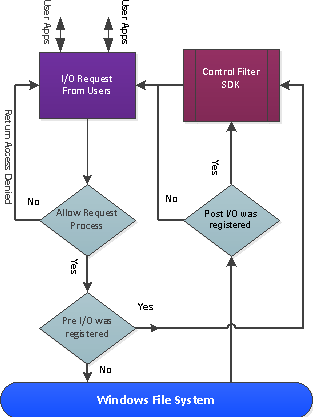
## Supported Platforms

* Windows 2016/Windows 2019/Windows 2022
* Windows 10/11 (32bit, 64bit)
* Windows 8 (32bit, 64bit)
* Windows 2012 Server R2
* Windows 2008 Server R2 (32bit, 64bit)
* Windows 7 (32bit,64bit)

# Overview and Basic Concepts

### The EaseFilter file system filter driver

EaseFilter file system filter driver is a kernel component module which is sitting on the layer between the I/O manager and the file system. When a user application invokes a win32 API to a file, the filter driver can intercept this I/O, based on the policies was set with the filter rule, the I/O information can be sent to the user, or be modified/blocked the access based on the setting as below figure.



### Filter Rule

A filter rule is the policy to monitor or control the files by the filter driver. To manage the files by filter driver, you need to create at least one filter rule, you can have multiple filter rules with different policies to manage different files to meet your requirement.

A filter rule has only one unique include file mask, when a new filter rule was added to the filter driver, if a filter rule with the same include file filter mask exists, the new filter rule will replace the older one.

### I/O registration

#### Pre IO and Post IO

A pre IO is the I/O operation which is not going down to the file system. A post IO is the I/O operation which was returned from the file system after it was processed.

To get the notification of an I/O for a monitor filter driver you need to register the specific post IO callback class. To control the file I/O, you can register the pre IO or post IO callback class for the control filter driver, you can modify the I/O data in your callback function.

# Symbol Reference

## Structures, Enums

### *Typedef enum FilterType*

{

*FILE\_SYSTEM\_CONTROL = 1,*

*FILE\_SYSTEM\_ENCRYPTION = 2,*

*FILE\_SYSTEM\_MONITOR = 4,*

*FILE\_SYSTEM\_REGISTRY = 8,*

*FILE\_SYSTEM\_PROCESS = 16,*

};

**Comments**

FILE\_SYSTEM\_CONTROL filter type is the file system filter driver which can control the file I/O request’s behaviour before it goes down to the file system or after it is completed by the file system.

FILE\_SYSTEM\_ENCRYPTION filter type is the file system filter driver which can encrypt and decrypt the files on-the-fly.

FILE\_SYSTEM\_MONITOR filter type is the file system filter driver which only intercept the file I/O notification after it was completed.

FILE\_SYSTEM\_REGISTRY filter type is the filter driver which can track and control the registry access.

FILE\_SYSTEM\_PROCESS filter type is the filter driver which can track and control the process and thread operations.

### *typedef enum IOCallbackClass*

{

*PRE\_CREATE = 0x00000001,*

*POST\_CREATE = 0x00000002,*

*PRE\_NEW\_FILE\_CREATED = 0x0000000100000000,*

*POST\_NEW\_FILE\_CREATED = 0x0000000200000000,*

*PRE\_FASTIO\_READ = 0x00000004,*

*POST\_FASTIO\_READ = 0x00000008,*

*PRE\_CACHE\_READ = 0x00000010,*

*POST\_CACHE\_READ = 0x00000020,*

*PRE\_NOCACHE\_READ = 0x00000040,*

*POST\_NOCACHE\_READ = 0x00000080,*

*PRE\_PAGING\_IO\_READ = 0x00000100,*

*POST\_PAGING\_IO\_READ = 0x00000200,*

*PRE\_FASTIO\_WRITE = 0x00000400,*

*POST\_FASTIO\_WRITE = 0x00000800,*

*PRE\_CACHE\_WRITE = 0x00001000,*

*POST\_CACHE\_WRITE = 0x00002000,*

*PRE\_NOCACHE\_WRITE = 0x00004000,*

*POST\_NOCACHE\_WRITE = 0x00008000,*

*PRE\_PAGING\_IO\_WRITE = 0x00010000,*

*POST\_PAGING\_IO\_WRITE = 0x00020000,*

*PRE\_QUERY\_INFORMATION = 0x00040000,*

*POST\_QUERY\_INFORMATION = 0x00080000,*

*PRE\_QUERY\_FILE\_SIZE = 0x0000000400000000,*

*POST\_QUERY\_FILE\_SIZE = 0x0000000800000000,*

*PRE\_QUERY\_FILE\_BASIC\_INFO = 0x0000001000000000,*

*POST\_QUERY\_FILE\_BASIC\_INFO = 0x0000002000000000,*

*PRE\_QUERY\_FILE\_STANDARD\_INFO = 0x0000004000000000,*

*POST\_QUERY\_FILE\_STANDARD\_INFO = 0x0000008000000000,*

*PRE\_QUERY\_FILE\_NETWORK\_INFO = 0x0000010000000000,*

*POST\_QUERY\_FILE\_NETWORK\_INFO = 0x0000020000000000,*

*PRE\_QUERY\_FILE\_ID = 0x0000040000000000,*

*POST\_QUERY\_FILE\_ID = 0x0000080000000000,*

*PRE\_SET\_INFORMATION = 0x00100000,*

*POST\_SET\_INFORMATION = 0x00200000,*

*PRE\_SET\_FILE\_SIZE = 0x0000400000000000,*

*POST\_SET\_FILE\_SIZE = 0x0000800000000000,*

*PRE\_SET\_FILE\_BASIC\_INFO = 0x0001000000000000,*

*POST\_SET\_FILE\_BASIC\_INFO = 0x0002000000000000,*

*PRE\_SET\_FILE\_STANDARD\_INFO = 0x0004000000000000,*

*POST\_SET\_FILE\_STANDARD\_INFO = 0x0008000000000000,*

*PRE\_SET\_FILE\_NETWORK\_INFO = 0x0010000000000000,*

*POST\_SET\_FILE\_NETWORK\_INFO = 0x0020000000000000,*

*PRE\_RENAME\_FILE = 0x0040000000000000,*

*POST\_RENAME\_FILE = 0x0080000000000000,*

*PRE\_DELETE\_FILE = 0x0100000000000000,*

*POST\_DELETE\_FILE = 0x0200000000000000,*

*PRE\_DIRECTORY = 0x00400000,*

*POST\_DIRECTORY = 0x00800000,*

*PRE\_QUERY\_SECURITY = 0x01000000,*

*POST\_QUERY\_SECURITY = 0x02000000,*

*PRE\_SET\_SECURITY = 0x04000000,*

*POST\_SET\_SECURITY = 0x08000000,*

*PRE\_CLEANUP = 0x10000000,*

*POST\_CLEANUP = 0x20000000,*

*PRE\_CLOSE = 0x40000000,*

*POST\_CLOSE = 0x80000000,*

};

**Members**

**PRE\_CREATE**

This is the IRP\_MJ\_CREATE request, the create I/O request before it goes down to the file system.

**POST\_CREATE**

This is the IRP\_MJ\_CREATE request, the create I/O request after it is completed by file system.

**PRE\_NEW\_FILE\_CREATED**

This is the IRP\_MJ\_CREATE request, if you want to filter the file going to be created operation only, you can register PRE\_NEW\_FILE\_CREATED.

**POST\_NEW\_FILE\_CREATED**

This is the IRP\_MJ\_CREATE request, if you want to filter the file being created operation only, you can register PRE\_NEW\_FILE\_CREATED.

**PRE\_FASTIO\_READ**

This is the fast I/O pre read request if the data is in the cache.

**POST\_FASTIO\_READ**

This is the fast I/O post read request if the data is in the cache. If the data is already in the Cache Manager,it will return true, or it will return false and the I/O Manager will reissue a new IRP cache read request to the file system.

**PRE\_CACHE\_READ**

This is the IRP\_MJ\_READ cache read pre request, before the data was read from the cache manager.

**POST\_CACHE\_READ**

This is the IRP\_MJ\_READ cache read post request, after the data was read from the cache manager. If the data is not in the Cache Manager, it will trigger a paging I/O read request and load the data from the storage to the Cache Manager. Normally you will see the paging I/O read request follows the cache read request.

**PRE\_NOCACHE\_READ**

This is the IRP\_MJ\_READ no cache read pre request, read the data from the disk directly, bypass the cache manager.

**POST\_NOCACHE\_READ**

This is the IRP\_MJ\_READ no cache read post request, read the data from the disk directly, bypass the cache manager. You will see the noncache read request if you open a file with the flag FILE\_NO\_INTERMEDIATE\_BUFFERING.

**PRE\_PAGING\_IO\_READ**

This is the IRP\_MJ\_READ paging read pre request, read the data from the disk to the cache.

**POST\_PAGING\_IO\_READ**

This is the IRP\_MJ\_READ paging read post request, read the data from the disk to the cache. It is initiated by the virtual memory system in order to satisfy the needs of the demand paging system.

**PRE\_FASTIO\_WRITE**

This is the fast I/O pre write request to the cache.

**POST\_FASTIO\_WRITE**

This is the fast I/O pre write request to the cache. The data was written to the cache if the request is satisfied immediately, or a IRP cache write will be issued.

**PRE\_CACHE\_WRITE**

This is the IRP\_MJ\_WRITE cache write pre request, write the data to the cache manager.

**POST\_CACHE\_WRITE**

This is the IRP\_MJ\_WRITE cache write post request, write the data to the cache manager. The paging I/O write request will be issued after the cache write.

**PRE\_NOCACHE\_WRITE**

This is the IRP\_MJ\_WRITE no cache write pre request, write the data to the disk directly, bypass the cache manager.

**POST\_NOCACHE\_WRITE**

This is the IRP\_MJ\_WRITE no cache write pre request, write the data to the disk directly, bypass the cache manager.You will see the noncache write request if you open a file with flag FILE\_NO\_INTERMEDIATE\_BUFFERING.

**PRE\_PAGING\_IO\_WRITE**

This is the IRP\_MJ\_WRITE paging write pre request, write the data from cache to the disk.

**POST\_PAGING\_IO\_WRITE**

This is the IRP\_MJ\_WRITE paging write post request, after the data was written from the cache to the disk.

**PRE\_QUERY\_INFORMATION**

This is the IRP\_MJ\_QUERY\_INFORMATION pre request to retrive the information for a given file before it goese down to the file system. The file information class tells the type of the information will be returned.

**POST\_QUERY\_INFORMATION**

This is the IRP\_MJ\_QUERY\_INFORMATION post request to retrive the information for a given file after it came back from the file system. The file information class tells the type of the information will be returned.

**PRE\_QUERY\_FILE\_SIZE**

This is the IRP\_MJ\_QUERY\_INFORMATION pre request with information class FileEndOfFileInformation, if you only want to get the callback for the file size query, you can register this class.

**POST\_QUERY\_FILE\_SIZE**

This is the IRP\_MJ\_QUERY\_INFORMATION post request with information class FileEndOfFileInformation.

**PRE\_QUERY\_FILE\_BASIC\_INFO**

This is the IRP\_MJ\_QUERY\_INFORMATION pre request with information class FileBasicInformation, if you only want to get the callback for the file basic information query, you can register this class.

**POST\_QUERY\_FILE\_BASIC\_INFO**

This is the IRP\_MJ\_QUERY\_INFORMATION post request with information class FileBasicInformation.

**PRE\_QUERY\_FILE\_STANDARD\_INFO**

This is the IRP\_MJ\_QUERY\_INFORMATION pre request with information class FileStandardInformation, if you only want to get the callback for the file standard information query, you can register this class.

**POST\_QUERY\_FILE\_STANDARD\_INFO**

This is the IRP\_MJ\_QUERY\_INFORMATION post request with information class FileStandardInformation.

**PRE\_QUERY\_FILE\_NETWORK\_INFO**

This is the IRP\_MJ\_QUERY\_INFORMATION pre request with information class FileNetworkOpenInformation, if you only want to get the callback for the file network information query, you can register this class.

**POST\_QUERY\_FILE\_NETWORK\_INFO**

This is the IRP\_MJ\_QUERY\_INFORMATION post request with information class FileNetworkOpenInformation.

**PRE\_QUERY\_FILE\_ID**

This is the IRP\_MJ\_QUERY\_INFORMATION pre request with information class FileInternalInformation, if you only want to get the callback for the file Id information query, you can register this class.

**POST\_QUERY\_FILE\_ID**

This is the IRP\_MJ\_QUERY\_INFORMATION post request with information class FileInternalInformation.

**PRE\_SET\_INFORMATION**

This is the IRP\_MJ\_SET\_INFORMATION pre request to set the information for a given file before it goese down to the file system. The file information class tells the type of the information will be set.

**POST\_SET\_INFORMATION**

This is the IRP\_MJ\_SET\_INFORMATION post request to set the information for a given file after it came back from the file system. The file information class tells the type of the information will be set.

**PRE\_SET\_FILE\_SIZE**

This is the IRP\_MJ\_SET\_INFORMATION pre request to set the information with class FileEndOfFileInformation for a given file before it goese down to the file system. You can register this I/O class if you want to filter the file size change.

**POST\_SET\_FILE\_SIZE**

This is the IRP\_MJ\_SET\_INFORMATION post request to set the information with class FileEndOfFileInformation for a given file after the I/O completed and came back from the file system.

**PRE\_SET\_FILE\_BASIC\_INFO**

This is the IRP\_MJ\_SET\_INFORMATION pre request to set the information with class FileBasicInformation for a given file before it goese down to the file system. You can register this I/O class if you want to filter the file basic information change.

**POST\_SET\_FILE\_BASIC\_INFO**

This is the IRP\_MJ\_SET\_INFORMATION post request to set the information with class FileBasicInformation for a given file after the I/O completed and came back from the file system.

**PRE\_SET\_FILE\_STANDARD\_INFO**

This is the IRP\_MJ\_SET\_INFORMATION pre request to set the information with class FileStandardInformation for a given file before it goese down to the file system. You can register this I/O class if you want to filter the file standard information change.

**POST\_SET\_FILE\_BASIC\_INFO**

This is the IRP\_MJ\_SET\_INFORMATION post request to set the information with class FileStandardInformation for a given file after the I/O completed and came back from the file system.

**PRE\_SET\_FILE\_NETWORK\_INFO**

This is the IRP\_MJ\_SET\_INFORMATION pre request to set the information with class FileNetworkOpenInformation for a given file before it goese down to the file system. You can register this I/O class if you want to filter the file network information change.

**POST\_SET\_FILE\_NETWORK\_INFO**

This is the IRP\_MJ\_SET\_INFORMATION post request to set the information with class FileNetworkOpenInformation for a given file after the I/O completed and came back from the file system.

**PRE\_RENAME\_FILE**

This is the IRP\_MJ\_SET\_INFORMATION pre request to set the information with class FileMoveOrRenameInformation for a given file before it goese down to the file system. You can register this I/O class if you want to filter the file move or rename operation.

**POST\_RENAME\_FILE**

This is the IRP\_MJ\_SET\_INFORMATION post request to set the information with class FileMoveOrRenameInformation for a given file after the I/O completed and came back from the file system.

**PRE\_DELETE\_FILE**

This is the IRP\_MJ\_SET\_INFORMATION pre request to set the information with class FileDispositionInformation for a given file before it goese down to the file system. You can register this I/O class if you want to filter the file delete operation.

**POST\_DELETE\_FILE**

This is the IRP\_MJ\_SET\_INFORMATION post request to set the information with class FileDispositionInformation for a given file after the I/O completed and came back from the file system.

**PRE\_DIRECTORY**

This the IRP\_MJ\_DIRECTORY\_CONTROL pre request for the folder browsing I/O request before it goese down to the file system. It retrive various kinds of information about files in the given directory. The information class tells the type of information will be returned.

**POST\_DIRECTORY**

This the IRP\_MJ\_DIRECTORY\_CONTROL post request for the folder browsing I/O request after the data came back from the file system. It retrive various kinds of information about files in the given directory. The information class tells the type of information will be returned.

**PRE\_QUERY\_SECURITY**

This is the IRP\_MJ\_QUERY\_SECURITY pre request to query the security information before it goes down to the file system. It will retrive the security descriptor for a given file. The security information tells the the type of the security descriptor.

**POST\_QUERY\_SECURITY**

This is the IRP\_MJ\_QUERY\_SECURITY post request to query the security information after the data came back from the file system. It will retrive the security descriptor for a given file. The security information tells the the type of the security descriptor.

**PRE\_SET\_SECURITY**

This is the IRP\_MJ\_SET\_SECURITY pre request to set the security information before it goes down to the file system. It will set the security state for a given file. The security information tells the the type of the security descriptor.

**POST\_SET\_SECURITY**

This is the IRP\_MJ\_SET\_SECURITY post request to set the security information after the data was written to the file system. It will set the security state for a given file. The security information tells the the type of the security descriptor.

**PRE\_CLEANUP**

This is the IRP\_MJ\_CLEANUP pre request before it goes down to the file system. It indicates that the handle reference count on a file object has reached zero. In other words, all handles to the file object have been closed. Often it is sent when a user-mode application has called the Microsoft Win32 CloseHandle function on the last outstanding handle to a file object.

**POST\_CLEANUP**

This is the IRP\_MJ\_CLEANUP post request after the I/O completed and came back from the file system.

**PRE\_CLOSE**

This is the IRP\_MJ\_CLOSE pre request before it goese down to the file system. It indicates that the reference count on a file object has reached zero, usually because a file system driver or other kernel-mode component has called ObDereferenceObject on the file object. This request normally follows a cleanup request. However, this does not necessarily mean that the close request will be received immediately after the cleanup request.

**POST\_CLOSE**

This is the IRP\_MJ\_CLOSE post request after the I/O completed and came back from the file system.

**Comments**

Register the I/O request with the combination of the request type you want to monitor. For file system monitor filter, only post requests are affected.

### *typedef enum FilterCommand*

{

*FILTER\_SEND\_FILE\_CHANGED\_EVENT = 0X00010001,*

*FILTER\_REQUEST\_USER\_PERMIT = 0X00010002,*

*FILTER\_REQUEST\_ENCRYPTION\_KEY = 0X00010003,*

*FILTER\_REQUEST\_ENCRYPTION\_IV\_AND\_KEY = 0X00010004,* *FILTER\_REQUEST\_ENCRYPTION\_IV\_AND\_KEY\_ACCESSFLAG=0X00010005,*

*FILTER\_REQUEST\_ENCRYPTION\_IV\_AND\_KEY\_TAGDATA =0X00010006,*

*FILTER\_SEND\_REG\_CALLBACK\_INFO = 0X00010007,*

*FILTER\_SEND\_PROCESS\_CREATION\_INFO = 0X00010008,*

*FILTER\_SEND\_PROCESS\_TERMINATION\_INFO = 0X00010009,*

*FILTER\_SEND\_THREAD\_CREATION\_INFO = 0X0001000A,*

*FILTER\_SEND\_THREAD\_TERMINATION\_INFO = 0X0001000B,*

*FILTER\_SEND\_PROCESS\_HANDLE\_INFO = 0X0001000C,*

*FILTER\_SEND\_THREAD\_HANDLE\_INFO = 0X0001000D*

*FILTER\_SEND\_ATTACHED\_VOLUME\_INFO = 0x0001000e,*

*FILTER\_SEND\_DETACHED\_VOLUME\_INFO = 0x0001000f,*

*FILTER\_SEND\_DENIED\_FILE\_IO\_EVENT = 0x00010010,*

*FILTER\_SEND\_DENIED\_VOLUME\_DISMOUNT\_EVENT = 0x00010011,*

*FILTER\_SEND\_DENIED\_PROCESS\_EVENT = 0x00010012,*

*FILTER\_SEND\_DENIED\_REGISTRY\_ACCESS\_EVENT = 0x00010013,*

*FILTER\_SEND\_DENIED\_PROCESS\_TERMINATED\_EVENT = 0x00010014,*

*FILTER\_SEND\_DENIED\_USB\_READ\_EVENT = 0x00010015,*

*FILTER\_SEND\_DENIED\_USB\_WRITE\_EVENT = 0x00010016,*

*FILTER\_SEND\_PRE\_TERMINATE\_PROCESS\_INFO = 0x00010017,*

}

**FILTER\_SEND\_FILE\_CHANGED\_EVENT**

If the monitor filter driver register the event type with API ‘RegisterEventTypeToFilterRule’,the notification will be sent when the file was changed.

**FILTER\_REQUEST\_USER\_PERMIT**

If the file has this control flag, the filter driver will request the permission to open this file.

**FILTER\_REQUEST\_ENCRYPTION\_KEY**

For encryption filter driver, if this control flag was enable for this file, the filter driver will request the encryption key for this file.

**FILTER\_REQUEST\_ENCRYPTION\_IV\_AND\_KEY**

For encryption filter driver, if this control flag was enable for this file, the filter driver will request the encryption key and IV for this file.

**FILTER\_REQUEST\_ENCRYPTION\_IV\_AND\_KEY\_AND\_ACCCESSFLAG**

For encryption filter driver, if this control flag was enable for this file, the filter driver will request the encryption key, IV and access flag for this file.

**FILTER\_REQUEST\_ENCRYPTION\_IV\_AND\_KEY\_AND\_TAGDATA**

For encryption filter driver, if this control flag was enable for this file, the filter driver will request the encryption key, IV and tag data for this file.

**FILTER\_SEND\_REG\_CALLBACK\_INFO**

For registry filter driver, if the registry callback class was registered, the registry access notification will be sent whent the registry was accessed.

**FILTER\_SEND\_PROCESS\_CREATION\_INFO**

For process filter driver, if the PROCESS\_CREATION\_NOTIFICATION

was registered, the process infomation will be sent when the new process was created.

**FILTER\_SEND\_PROCESS\_TERMINATION\_INFO**

For process filter driver, if the PROCESS\_TERMINATION\_NOTIFICATION

was registered, the process infomation will be sent when the process was terminated.

**FILTER\_SEND\_THREAD\_CREATION\_INFO**

For process filter driver, if the THREAD\_CREATION\_NOTIFICATION

was registered, the process infomation will be sent when the new thread was created.

**FILTER\_SEND\_THREAD\_TERMINATION\_INFO**

For process filter driver, if the THREAD\_TERMINATION\_NOTIFICATION

was registered, the process infomation will be sent when the thread was terminated.

**FILTER\_SEND\_PROCESS\_HANDLE\_INFO**

For process filter driver, if the PROCESS\_HANDLE\_OP\_NOTIFICATION

was registered, the process infomation will be sent when a handle for a process was created or duplicated.

**FILTER\_SEND\_THREAD\_HANDLE\_INFO**

For process filter driver, if the THREAD\_ HANDLE\_OP\_NOTIFICATION

was registered, the process infomation will be sent when a handle for a thread was created or duplicated.

**FILTER\_SEND\_ATTACHED\_VOLUME\_INFO**

Send the volume information if there a volume was attached to the filter driver if it was enabled in volume control flag.

**FILTER\_SEND\_DETTACHED\_VOLUME\_INFO**

Send the volume information if there a volume was dettached from the filter driver if it was enabled in volume control flag.

**FILTER\_SEND\_FILE\_IO\_DENIED\_EVENT**

Send file I/O information if the I/O was blocked by the filter driver with access flag setting if the flag ENABLE\_SEND\_DENIED\_EVENT was enabled in global boolean setting.

**FILTER\_SEND\_FILE\_IO\_DENIED\_EVENT**

Send file I/O information if the I/O was blocked by the filter driver with access flag setting if the flag ENABLE\_SEND\_DENIED\_EVENT was enabled in global boolean setting.

**FILTER\_SEND\_DENIED\_VOLUME\_DISMOUNT\_EVENT**

Send volume information if the volume dismount operation was blocked by the filter driver with the flag BLOCK\_VOLUME\_DISMOUNT enabled in the volume control flag if the ENABLE\_SEND\_DENIED\_EVENT was enabled in global boolean setting.

**FILTER\_SEND\_DENIED\_PROCESS\_EVENT**

Send the new process information if new process creation was blocked by the filter driver with flag DENY\_NEW\_PROCESS\_CREATION was enabled in the process control flag if the flag ENABLE\_SEND\_DENIED\_EVENT was enabled in global boolean setting.

**FILTER\_SEND\_DENIED\_REGISTRY\_ACCESS\_EVENT**

Send registry access information if the operation was blocked by the filter driver in registry control flag setting if the flag ENABLE\_SEND\_DENIED\_EVENT was enabled in global boolean setting.

**FILTER\_SEND\_DENIED\_PROCESS\_TERMINATION\_EVENT**

Send the process information if the protected process termination was blocked by the filter driver if the flag ENABLE\_SEND\_DENIED\_EVENT was enabled in global boolean setting.

**FILTER\_SEND\_DENIED\_USB\_READ\_EVENT**

Send file I/O information if the USB read was block by the filter driver with flag BLOCK\_USB\_READ enabled if the flag ENABLE\_SEND\_DENIED\_EVENT was enabled in global boolean setting.

**FILTER\_SEND\_DENIED\_USB\_WRITE\_EVENT**

Send file I/O information if the USB write was block by the filter driver with flag BLOCK\_USB\_WRITE enabled if the flag ENABLE\_SEND\_DENIED\_EVENT was enabled in global boolean setting.

**FILTER\_SEND\_PRE\_TEMINIATE\_PROCESS\_INFO**

Send the process information before it was going to be terminated if the flag PROCESS\_PRE\_TERMINATION\_REQUEST was enabled in process control setting.

**Comments**

This is the command which was sent from the filter driver with the information associated to the specfic command in structure ‘MESSAGE\_SEND\_DATA’.

### *typedef enum VolumeControlFlag*

{

*GET\_ATTACHED\_VOLUME\_INFORMATION = 0X00000001,*

*VOLUME\_ATTACHED\_NOTIFICATION = 0X00000002,*

*VOLUME\_DETACHED\_NOTIFICATION = 0X00000004,*

*BLOCK\_VOLUME\_DISMOUNT = 0X00000008,*

*BLOCK\_USB\_READ = 0x00000010,*

*BLOCK\_USB\_WRITE = 0x00000020,*

}

**GET\_ATTACHED\_VOLUME\_INFORMATION**

If this flag is enabled, the filter driver will send all the attached volume information to the user mode service.

**VOLUME\_ATTACHED\_NOTIFICATION**

If this flag is enabled, you will get the notification when the filter driver attached to a volume.

**VOLUME\_DETACHED\_NOTIFICATION**

If this flag is enabled, you will get the notification when the filter driver detached from a volume.

**BLOCK\_VOLUME\_DISMOUNT**

If this flag is enabled, it will prevent the attached volumes from being formatted or dismounted.

**BLOCK\_USB\_READ**

If this flag is enabled, it will prevent USB volume from being read.

**BLOCK\_USB\_WRITE**

If this flag is enabled, it will prevent USB volume from being written.

**Comments**

This is the control flag for volume information with API “SetVolumeControlFlag”, enable you to get the notification when the filter driver attached to a volume or detached a volume, or prevent the attached volumes from being formatted or dismounted.

### *typedef struct \_VOLUME\_INFO*

*{*

*ULONG VolumeNameLength;*

*WCHAR VolumeName[MAX\_FILE\_NAME\_LENGTH];*

*ULONG VolumeDosNameLength;*

*WCHAR VolumeDosName[MAX\_FILE\_NAME\_LENGTH];*

*ULONG VolumeFileSystemType;*

*ULONG DeviceCharacteristics;*

*}* ***VOLUME\_INFO****, \*****PVOLUME\_INFO****;*

**Members**

**VolumeNameLength**

This is the length of the volume name.

**VolumeName**

The volume name buffer.

**VolumeDosNameLength**

The length of the volume dos name.

**VolumeDosName**

The volume dos name buffer.

**VolumeFileSystemType**

The file system type of the attached volume.

**DeviceCharacteristics**

The volume attached device’s characteristics.

**Comments**

This is the structure of the volume’s information when the filter driver attached to a volume or detached a volume.

### *typedef enum ProcessControlFlag*

{

*DENY\_NEW\_PROCESS\_CREATION = 0X00000001,*

*ENABLE\_SEND\_DENIED\_PROCESS\_EVENT = 0X00000002,*

*PROCESS\_PRE\_TERMINATION\_REQUEST = 0X00000004,*

*PROCESS\_CREATION\_NOTIFICATION = 0X00000100,*

*PROCESS\_TERMINATION\_NOTIFICATION = 0X00000200,*

*PROCESS\_HANDLE\_OP\_NOTIFICATION = 0X00000400,*

*THREAD\_CREATION\_NOTIFICATION = 0X00000800,*

*THREAD\_TERMINATION\_NOTIFICATION = 0X00001000,*

*THREAD\_HANDLE\_OP\_NOTIFICATION = 0X00002000,*

}

**DENY\_NEW\_PROCESS\_CREATION**

If this flag is enabled, it will block the new process creation when the process name matchs the process name filter mask. It is especially good to prevent the untrusted process from being launched.

**ENABLE\_SEND\_DENIED\_PROCESS\_EVENT**

If this flag is enabled, a notification will be sent when a new process creation was blocked by the filter driver.

**PROCESS\_PRE\_TERMINATION\_REQUEST**

If this flag is enabled, a callback request will be sent before the process is going to be terminated ungratefully, you can block the termination in your callback function.

**PROCESS\_CREATION\_NOTIFICATION**

If this flag is enabled and was registered to the process filter rule, the process creation notification will be sent when the creating new process name matchs the process name filter mask.

**PROCESS\_TERMINATION\_NOTIFICATION**

If this flag is enabled and was registered to the process filter rule, the process termination notification will be sent when the terminating process name matchs the process name filter mask.

**PROCESS\_HANDLE\_OP\_NOTIFICATION**

If this flag is enabled and was registered to the process filter rule, the process operation information notification will be sent when a handle for a process is being created or duplicated.

**THREAD\_CREATION\_NOTIFICATION**

If this flag is enabled and was registered to the process filter rule, the thread creation notification will be sent when the creating new thread’s process name matchs the process name filter mask.

**THREAD\_TERMINATION\_NOTIFICATION**

If this flag is enabled and was registered to the process filter rule, the thread termination notification will be sent when the terminating thread’s process name matchs the process name filter mask.

**THREAD\_HANDLE\_OP\_NOTIFICATION**

If this flag is enabled and was registered to the process filter rule, the thread operation information notification will be sent when a handle for a thread is being created or duplicated.

**Comments**

This is the control flag for process filter rule, to prevent untrusted process from being launched or track the process operations.

### *typedef enum RegCallbackClass*

{

*Reg\_Pre\_Delete\_Key = 0x00000001,*

*Reg\_Pre\_Set\_Value\_Key = 0x00000002,*

*Reg\_Pre\_Delete\_Value\_Key = 0x00000004,*

*Reg\_Pre\_SetInformation\_Key = 0x00000008,*

*Reg\_Pre\_Rename\_Key = 0x00000010,*

*Reg\_Pre\_Enumerate\_Key = 0x00000020,*

*Reg\_Pre\_Enumerate\_Value\_Key = 0x00000040,*

*Reg\_Pre\_Query\_Key = 0x00000080,*

*Reg\_Pre\_Query\_Value\_Key = 0x00000100,*

*Reg\_Pre\_Query\_Multiple\_Value\_Key = 0x00000200,*

*Reg\_Pre\_Create\_Key = 0x00000400,*

*Reg\_Post\_Create\_Key = 0x00000800,*

*Reg\_Pre\_Open\_Key = 0x00001000,*

*Reg\_Post\_Open\_Key = 0x00002000,*

*Reg\_Pre\_Key\_Handle\_Close = 0x00004000,*

*//*

*// .Net only*

*//*

*Reg\_Post\_Delete\_Key = 0x00008000,*

*Reg\_Post\_Set\_Value\_Key = 0x00010000,*

*Reg\_Post\_Delete\_Value\_Key = 0x00020000,*

*Reg\_Post\_SetInformation\_Key = 0x00040000,*

*Reg\_Post\_Rename\_Key = 0x00080000,*

*Reg\_Post\_Enumerate\_Key = 0x00100000,*

*Reg\_Post\_Enumerate\_Value\_Key = 0x00200000,*

*Reg\_Post\_Query\_Key = 0x00400000,*

*Reg\_Post\_Query\_Value\_Key = 0x00800000,*

*Reg\_Post\_Query\_Multiple\_Value\_Key = 0x01000000,*

*Reg\_Post\_Key\_Handle\_Close = 0x02000000,*

*Reg\_Pre\_Create\_KeyEx = 0x04000000,*

*Reg\_Post\_Create\_KeyEx = 0x08000000,*

*Reg\_Pre\_Open\_KeyEx = 0x10000000,*

*Reg\_Post\_Open\_KeyEx = 0x20000000,*

*//*

*// new to Windows Vista*

*//*

*Reg\_Pre\_Flush\_Key = 0x40000000,*

*Reg\_Post\_Flush\_Key = 0x80000000,*

*Reg\_Pre\_Load\_Key = 0x100000000,*

*Reg\_Post\_Load\_Key = 0x200000000,*

*Reg\_Pre\_UnLoad\_Key = 0x400000000,*

*Reg\_Post\_UnLoad\_Key = 0x800000000,*

*Reg\_Pre\_Query\_Key\_Security = 0x1000000000,*

*Reg\_Post\_Query\_Key\_Security = 0x2000000000,*

*Reg\_Pre\_Set\_Key\_Security = 0x4000000000,*

*Reg\_Post\_Set\_Key\_Security = 0x8000000000,*

*//*

*// per-object context cleanup*

*//*

*Reg\_Callback\_Object\_Context\_Cleanup = 0x10000000000,*

*//*

*// new in Vista SP2*

*//*

*Reg\_Pre\_Restore\_Key = 0x20000000000,*

*Reg\_Post\_Restore\_Key = 0x40000000000,*

*Reg\_Pre\_Save\_Key = 0x80000000000,*

*Reg\_Post\_Save\_Key = 0x100000000000,*

*Reg\_Pre\_Replace\_Key = 0x200000000000,*

*Reg\_Post\_Replace\_Key = 0x400000000000,*

*//*

*// new in Windows 10*

*//*

*Reg\_Pre\_Query\_KeyName = 0x800000000000,*

*Reg\_Post\_Query\_KeyName = 0x1000000000000,*

*MAX\_REG\_CALLBACK\_CLASE = 0xffffffffffffffff,*

}

**REG\_PRE\_DELETE\_KEY**

Specifies that a thread is attempting to delete a key. This value indicates a pre-notification call to RegistryCallback.

**REG\_PRE\_SET\_VALUE\_KEY**

Specifies that a thread is attempting to set a value entry for a key. This value indicates a pre-notification call to RegistryCallback.

**REG\_PRE\_DELETE\_VALUE\_KEY**

Specifies that a thread is attempting to delete a value entry for a key. This value indicates a pre-notification call to RegistryCallback.

**REG\_PRE\_SETINFORMATION\_KEY**

Specifies that a thread is attempting to set the metadata for a key. This value indicates a pre-notification call to RegistryCallback.

**REG\_PRE\_RENAME\_KEY**

Specifies that a thread is attempting to rename a key. This value indicates a pre-notification call to RegistryCallback.

**REG\_PRE\_ENUMERATE\_KEY**

Specifies that a thread is attempting to enumerate a value entry for a key. This value indicates a pre-notification call to RegistryCallback.

**REG\_PRE\_ENUMERATE\_KEY**

Specifies that a thread is attempting to enumerate a value entry for a key. This value indicates a pre-notification call to RegistryCallback.

**REG\_PRE\_QUERY\_KEY**

Specifies that a thread is attempting to read the metadata for a key. This value indicates a pre-notification call to RegistryCallback.

**REG\_PRE\_QUERY\_VALUE\_KEY**

Specifies that a thread is attempting to read a value entry for a key. This value indicates a pre-notification call to RegistryCallback.

**REG\_PRE\_QUERY\_MULTIPLE\_VALUE\_KEY**

Specifies that a thread is attempting to query multiple value entries for a key. This value indicates a pre-notification call to RegistryCallback.

**REG\_PRE\_CREATE\_KEY**

Specifies that a thread is attempting to create a key. This value indicates a pre-notification call to RegistryCallback.

**REG\_POST\_CREATE\_KEY**

Specifies that a thread has successfully created a key. This value indicates a post-notification call to RegistryCallback.

**REG\_PRE\_OPEN\_KEY**

Specifies that a thread is attempting to open an existing key. This value indicates a pre-notification call to RegistryCallback.

**REG\_POST\_OPEN\_KEY**

Specifies that a thread has successfully opened an existing key. This value indicates a post-notification call to RegistryCallback.

**REG\_PRE\_KEY\_HANDLE\_CLOSE**

Specifies that a thread is attempting to close a key handle. This value indicates a pre-notification call to RegistryCallback.

**REG\_POST\_DELETE\_KEY**

Specifies that the system has attempted to delete the key. This value indicates a post-notification call to RegistryCallback.

**REG\_POST\_SET\_VALUE\_KEY**

Specifies that the system has attempted to set a value entry for a key. This value indicates a post-notification call to RegistryCallback.

**REG\_POST\_DELETE\_VALUE\_KEY**

Specifies that the system has attempted to delete a value entry for a key. This value indicates a post-notification call to RegistryCallback.

**REG\_POST\_SETINFORMATION\_KEY**

Specifies that the system has attempted to set the key's metadata. This value indicates a post-notification call to RegistryCallback.

**REG\_POST\_RENAME\_KEY**

Specifies that the system has attempted to rename the key. This value indicates a post-notification call to RegistryCallback.

**REG\_POST\_ENUMERATE\_KEY**

Specifies that the system has attempted to enumerate the subkey of a key. This value indicates a post-notification call to RegistryCallback.

**REG\_POST\_ENUMERATE\_VALUE\_KEY**

Specifies that the system has attempted to enumerate the value entry of a key. This value indicates a post-notification call to RegistryCallback.

**REG\_POST\_QUERY\_KEY**

Specifies that the system has attempted to query the metadata for a key. This value indicates a post-notification call to RegistryCallback.

**REG\_POST\_QUERY\_VALUE\_KEY**

Specifies that the system has attempted to query a value entry for the key. This value indicates a post-notification call to RegistryCallback.

**REG\_POST\_QUERY\_MULTIPLE\_VALUE\_KEY**

Specifies that the system has attempted to query multiple value entries for the key. This value indicates a post-notification call to RegistryCallback.

**REG\_POST\_KEY\_HANDLE\_CLOSE**

Specifies that the system has attempted to close a key handle. This value indicates a post-notification call to RegistryCallback.

**REG\_PRE\_CREATE\_KEYEX**

Specifies that a thread is attempting to create a key. This value indicates a pre-notification call to RegistryCallback.

**REG\_POST\_CREATE\_KEYEX**

Specifies that the system has attempted to create a key. This value indicates a post-notification call to RegistryCallback.

**REG\_PRE\_OPEN\_KEYEX**

Specifies that a thread is attempting to open an existing key. This value indicates a pre-notification call to RegistryCallback.

**REG\_POST\_OPEN\_KEYEX**

Specifies that the system has attempted to open an existing key. This value indicates a post-notification call to RegistryCallback.

**REG\_PRE\_FLUSH\_KEY**

Specifies that a thread is attempting to write a key to disk. This value indicates a pre-notification call to RegistryCallback.

**REG\_POST\_FLUSH\_KEY**

Specifies that the system has attempted to write a key to disk. This value indicates a post-notification call to RegistryCallback.

**REG\_PRE\_LOAD\_KEY**

Specifies that a thread is attempting to load a registry hive from a file. This value indicates a pre-notification call to RegistryCallback.

**REG\_POST\_LOAD\_KEY**

Specifies that the system has attempted to load a registry hive from a file. This value indicates a post-notification call to RegistryCallback.

**REG\_PRE\_UNLOAD\_KEY**

Specifies that a thread is attempting to unload a registry hive. This value indicates a pre-notification call to RegistryCallback.

**REG\_POST\_UNLOAD\_KEY**

Specifies that the system has attempted to unload a registry hive. This value indicates a post-notification call to RegistryCallback.

**REG\_PRE\_QUERY\_KEY\_SECURITY**

Specifies that a thread is attempting to obtain a registry key's security information. This value indicates a pre-notification call to RegistryCallback.

**REG\_POST\_QUERY\_KEY\_SECURITY**

Specifies that a thread has attempted to obtain a registry key's security information. This value indicates a post-notification call to RegistryCallback.

**REG\_PRE\_SET\_KEY\_SECURITY**

Specifies that a thread is attempting to set a registry key's security information. This value indicates a pre-notification call to RegistryCallback.

**REG\_POST\_SET\_KEY\_SECURITY**

Specifies that a thread has attempted to set a registry key's security information. This value indicates a post-notification call to RegistryCallback.

**REG\_CALLBACK\_OBJECT\_CONTEXT\_CLEANUP**

Specifies that the driver has called CmUnRegisterCallback or the driver's RegistryCallback routine has just finished processing a RegNtPreKeyHandleClose class value.

**REG\_PRE\_RESTORE\_KEY**

Specifies that a thread is attempting to restore a registry key's information. This value indicates a pre-notification call to RegistryCallback.

**REG\_POST\_RESTORE\_KEY**

Specifies that a thread has attempted to restore a registry key's information. This value indicates a post-notification call to RegistryCallback.

**REG\_PRE\_SAVE\_KEY**

Specifies that a thread is attempting to save a registry key's information. This value indicates a pre-notification call to RegistryCallback.

**REG\_POST\_SAVE\_KEY**

Specifies that a thread has attempted to save a registry key's information. This value indicates a post-notification call to RegistryCallback.

**REG\_PRE\_REPLACE\_KEY**

Specifies that a thread is attempting to replace a registry key's information. This value indicates a pre-notification call to RegistryCallback.

**REG\_POST\_REPLACE\_KEY**

Specifies that a thread has attempted to replace a registry key's information. This value indicates a post-notification call to RegistryCallback.

**REG\_PRE\_QUERY\_KEYNAME**

Specifies that a thread is attempting to obtain the full path of a registry key. Use this value on Windows 10 and later versions of the Windows operating system.

**REG\_POST\_QUERY\_KEYNAME**

Specifies that a thread has attempted to obtain the full path of a registry key. Use this value on Windows 10 and later versions of the Windows operating system.

**MAX\_REG\_CALLBACK\_CLASS**

Specifies the maximum value in this enumeration type.

**Comments**

This is registry callback class, to get the notification of the registry operations.

### *typedef enum RegControlFlag*

{

*REG\_ALLOW\_OPEN\_KEY = 0X00000001,*

*REG\_ALLOW\_CREATE\_KEY = 0X00000002,*

*REG\_ALLOW\_QUERY\_KEY = 0X00000004,*

*REG\_ALLOW\_RENAME\_KEY = 0X00000008*

*REG\_ALLOW\_DELETE\_KEY = 0X00000010,*

*REG\_ALLOW\_SET\_VALUE\_KEY\_INFORMATION = 0X00000020,*

*REG\_ALLOW\_SET\_INFORMATION\_KEY = 0X00000040,*

*REG\_ALLOW\_ENUMERATE\_KEY = 0X00000080,*

*REG\_ALLOW\_QUERY\_VALUE\_KEY = 0X00000100,*

*REG\_ALLOW\_ENUMERATE\_VALUE\_KEY = 0X00000200,*

*REG\_ALLOW\_QUERY\_MULTIPLE\_VALUE\_KEY = 0X00000400,*

*REG\_ALLOW\_DELETE\_VALUE\_KEY = 0X00000800,*

*REG\_ALLOW\_QUERY\_KEY\_SECURITY = 0X00001000,*

*REG\_ALLOW\_SET\_KEY\_SECURITY = 0X00002000,*

*REG\_ALLOW\_RESTORE\_KEY = 0X00004000,*

*REG\_ALLOW\_SAVE\_KEY = 0X00010000,*

*REG\_ALLOW\_FLUSH\_KEY = 0X00020000,*

*REG\_ALLOW\_LOAD\_KEY = 0X00040000,*

*REG\_ALLOW\_UNLOAD\_KEY = 0X00080000,*

*REG\_ALLOW\_KEY\_CLOSE = 0X00100000,*

*REG\_ALLOW\_KEY\_RENAME = 0X00200000,*

*REG\_MAX\_ACCESS\_FLAG = 0XFFFFFFFF,*

}

**REG\_ALLOW\_OPEN\_KEY**

If this flag is disabled, it will block the registry key open if the process name matchs the filter rule.

**REG\_ALLOW\_CREATE\_KEY**

If this flag is disabled, it will block the registry key being created if the process name matchs the filter rule.

**REG\_ALLOW\_QUERY\_KEY**

If this flag is disabled, it will block the registry key being queryed if the process name matchs the filter rule.

**REG\_ALLOW\_RENAME\_KEY**

If this flag is disabled, it will block the registry key being renamed if the process name matchs the filter rule.

**REG\_ALLOW\_DELETE\_KEY**

If this flag is disabled, it will block the registry key being deleted if the process name matchs the filter rule.

**REG\_ALLOW\_SET\_VALUE\_KEY\_INFORMATION**

If this flag is disabled, it will block the setting the value of the registry key if the process name matchs the filter rule.

**REG\_ALLOW\_SET\_INFORMATION\_KEY**

If this flag is disabled, it will block the setting the registry key if the process name matchs the filter rule.

**REG\_ALLOW\_ENUMERATE\_KEY**

If this flag is disabled, it will block the registry key being enumerated if the process name matchs the filter rule.

**REG\_ALLOW\_QUERY\_VALUE\_KEY**

If this flag is disabled, it will block the value of the registry key being queryed if the process name matchs the filter rule.

**REG\_ALLOW\_ENUMERATE\_VALUE\_KEY**

If this flag is disabled, it will block the value of the registry key being enumerated if the process name matchs the filter rule.

**REG\_ALLOW\_QUERY\_MULTIPLE\_VALUE\_KEY**

If this flag is disabled, it will block the querying of multiple values of the registry key if the process name matchs the filter rule.

**REG\_ALLOW\_DELETE\_VALUE\_KEY**

If this flag is disabled, it will block the value of the registry key being deleted if the process name matchs the filter rule.

**REG\_ALLOW\_QUERY\_KEY\_SECURITY**

If this flag is disabled, it will block the registry key security being queryed if the process name matchs the filter rule.

**REG\_ALLOW\_SET\_KEY\_SECURITY**

If this flag is disabled, it will block the registry key security being setted if the process name matchs the filter rule.

**REG\_ALLOW\_RESTORE\_KEY**

If this flag is disabled, it will block the registry key being restored if the process name matchs the filter rule.

**REG\_ALLOW\_SAVE\_KEY**

If this flag is disabled, it will block the registry key being saved if the process name matchs the filter rule.

**REG\_ALLOW\_FLUSH\_KEY**

If this flag is disabled, it will block the registry key security being flushed if the process name matchs the filter rule.

**REG\_ALLOW\_LOAD\_KEY**

If this flag is disabled, it will block the registry key being loaded if the process name matchs the filter rule.

**REG\_ALLOW\_UNLOAD\_KEY**

If this flag is disabled, it will block the registry key being unloaded if the process name matchs the filter rule.

**REG\_ALLOW\_KEY\_CLOSE**

If this flag is disabled, it will block the registry key being closed if the process name matchs the filter rule.

**REG\_ALLOW\_KEY\_RENAME**

If this flag is disabled, it will block the registry key being renamed if the process name matchs the filter rule.

**REG\_MAX\_ACCESS\_FLAG**

Enable the maximum access right for the registry access.

**Comments**

This is the control flag for registry filter rule, to track or control the registry operations.

### *typedef enum FileEventType*

{

FILE\_WAS\_CREATED = 0x00000020,

FILE\_WAS\_WRITTEN = 0x00000040,

FILE\_WAS\_RENAMED = 0x00000080,

FILE\_WAS\_DELETED = 0x00000100,

FILE\_SECURITY\_CHANGED = 0x00000200,

FILE\_INFO\_CHANGED = 0x00000400,

FILE\_WAS\_READ = 0x00000800,

};

**Members**

**FILE\_WAS\_CREATED**

The new file was created event.

**FILE\_WAS\_WRITTEN**

The file was written with data event.

**FILE\_WAS\_RENAMED**

The file was renamed event.

**FILE\_SECURITY\_CHANGED**

The file security was changed event.

**FILE\_INFO\_CHANGED**

The file information was changed event.

**FILE\_WAS\_READ**

The file data was read event.

**Comments**

This is the file change event, you can get the notification when the file was changed by registering the specific events.

### *typedef enum AccessFlag*

{

*EXCLUDE\_FILTER\_RULE = 0X00000000,*

*EXCLUDE\_FILE\_ACCESS = 0x00000001,*

*REPARSE\_FILE\_OPEN = 0x00000002,*

*HIDE\_FILES\_IN\_DIRECTORY\_BROWSING = 0x00000004,*

*FILE\_ENCRYPTION\_RULE = 0x00000008,*

*ALLOW\_OPEN\_WTIH\_ACCESS\_SYSTEM\_SECURITY = 0x00000010,*

*ALLOW\_OPEN\_WITH\_READ\_ACCESS = 0x00000020,*

*ALLOW\_OPEN\_WITH\_WRITE\_ACCESS = 0x00000040,*

*ALLOW\_OPEN\_WITH\_CREATE\_OR\_OVERWRITE\_ACCESS = 0x00000080,*

*ALLOW\_OPEN\_WITH\_DELETE\_ACCESS = 0x00000100,*

*ALLOW\_READ\_ACCESS = 0x00000200,*

*ALLOW\_WRITE\_ACCESS = 0x00000400,*

*ALLOW\_QUERY\_INFORMATION\_ACCESS = 0x00000800,*

*ALLOW\_SET\_INFORMATION = 0x00001000,*

*ALLOW\_FILE\_RENAME = 0x00002000,*

*ALLOW\_FILE\_DELETE = 0x00004000,*

*ALLOW\_FILE\_SIZE\_CHANGE = 0x00008000,*

*ALLOW\_QUERY\_SECURITY\_ACCESS = 0x00010000,*

*ALLOW\_SET\_SECURITY\_ACCESS = 0x00020000,*

*ALLOW\_DIRECTORY\_LIST\_ACCESS = 0x00040000,*

*ALLOW\_FILE\_ACCESS\_FROM\_NETWORK = 0x00080000,*

*ALLOW\_NEW\_FILE\_ENCRYPTION = 0x00100000,*

*ALLOW\_READ\_ENCRYPTED\_FILES = 0x00200000,*

*ALLOW\_ALL\_SAVE\_AS = 0x00400000,*

*ALLOW\_COPY\_PROTECTED\_FILES\_OUT = 0x00800000,*

*ALLOW\_FILE\_MEMORY\_MAPPED = 0x01000000,*

*DISABLE\_ENCRYPT\_ON\_READ = 0x02000000,*

*ALLOW\_COPY\_PROTECTED\_FILES\_TO\_USB = 0x04000000,*

*LEAST\_ACCESS\_FLAG = 0xf0000000,*

*ALLOW\_MAX\_RIGHT\_ACCESS = 0xfffffff0,*

};

**Members**

**EXCLUDE\_FILTER\_RULE**

EXCLUDE\_FILTER\_RULE is the rule which bypass the files matched the FilterMask. It can`t combine to use with the other access flags. If a file matchs the exclude filter rule,the filter will bypass this file,you won`t get any Io request notification or control. If a file matches both the exclude filter rule and monitor rule, the exclude filter rule will be applied.

**EXCLUDE\_FILE\_ACCESS**

EXCLUDE\_FILE\_ACCESS is the flag indicates the filter will deny the access to the files which match the FilterMask.

**REPARSE\_FILE\_OPEN**

REPARSE\_FILE\_OPEN is the rule which reparses the file matched the FilterMask open to the other files which match the ReparseMask.

Example:

Reparse the file open in folder c:\test to another folder c:\reparseFolder"

AddFileFilterRule(ALLOW\_MAX\_RIGHT\_ACCESS|REPARSE\_FILE\_OPEN, L"c:\\test\\\*", 1);

AddReparseFileMaskToFilterRule(L"c:\\test\\\*",L"c:\\reparseFolder\\\*");

**HIDE\_FILES\_IN\_DIRECTORY\_BROWSING**

HIDE\_FILES\_IN\_DIRECTORY\_BROWSING is the flag let you hide the files in the managed folder when it matches the mask.

Example:

Hide the files in folder c:\test for process "explorer.exe"

AddFileFilterRule(ALLOW\_MAX\_RIGHT\_ACCESS|HIDE\_FILES\_IN\_DIRECTORY\_BROWSING, L"c:\\test\\\*", FALSE, 1);

AddIncludeProcessNameToFilterRule(L"c:\\test\\\*",L"explorer.exe");

AddHiddenFileMaskToFilterRule(L"c:\\test\\\*",L"\*.\*");

**ENCRYPTION\_FILTER\_RULE**

ENCRYPTION\_FILTER\_RULE is the flag indicates the filter will encrypt the new created files which match the FilterMask.If the other flag were set, this flag is automatically enabled.

**ALLOW\_OPEN\_WTIH\_ACCESS\_SYSTEM\_SECURITY**

ALLOW\_OPEN\_WTIH\_ACCESS\_SYSTEM\_SECURITY is the flag indicates if you can open the file with the desired access with the ACCESS\_SYSTEM\_SECURITY set.

**ALLOW\_OPEN\_WITH\_READ\_ACCESS**

ALLOW\_OPEN\_WITH\_READ\_ACCESS is the flag indicates if you can open the file with read access.

**ALLOW\_OPEN\_WITH\_WRITE\_ACCESS**

ALLOW\_OPEN\_WITH\_WRITE\_ACCESS is the flag indicates if you can open the file with write access.

**ALLOW\_OPEN\_WITH\_CREATE\_OR\_OVERWRITE\_ACCESS**

ALLOW\_OPEN\_WITH\_CREATE\_OR\_OVERWRITE\_ACCESS is the flag indicates if you can open with create a new file or overwrite the exist file.

**ALLOW\_OPEN\_WITH\_DELETE\_ACCESS**

ALLOW\_OPEN\_WITH\_DELETE\_ACCESS is the flag indicates if you can open the file for deletion or rename access.

**ALLOW\_READ\_ACCESS**

ALLOW\_READ\_ACCESS is the flag indicates if you have the permission to read the file.

**ALLOW\_WRITE\_ACCESS**

ALLOW\_WRITE\_ACCESS is the flag indicates if you have the permission to write the file.

**ALLOW\_QUERY\_INFORMATION\_ACCESS**

ALLOW\_QUERY\_INFORMATION\_ACCESS is the flag indicates if you have the permission to query the file information.

**ALLOW\_SET\_INFORMATION**

ALLOW\_SET\_INFORMATION is the flag indicates if you have the permission to set the file information.

**ALLOW\_FILE\_RENAME**

ALLOW\_FILE\_RENAME is the flag indicates if you have the permission to rename the file. If the flag ALLOW\_SET\_INFORMATION is unset, the rename is blocked automatically.

**ALLOW\_FILE\_DELETE**

ALLOW\_FILE\_DELETE is the flag indicates if you have the permission to delete the file. If the flag ALLOW\_SET\_INFORMATION is unset, the deletion is blocked automatically.

**ALLOW\_FILE\_SIZE\_CHANGE**

ALLOW\_FILE\_SIZE\_CHANGE is the flag indicates if you have the permission to change the file size. If the flag ALLOW\_SET\_INFORMATION is unset, the file size chage is blocked automatically.

**ALLOW\_QUERY\_SECURITY\_ACCESS**

ALLOW\_QUERY\_SECURITY\_ACCESS is the flag indicates if you have the permission to query the file security.

**ALLOW\_SET\_SECURITY\_ACCESS**

ALLOW\_SET\_SECURITY\_ACCESS is the flag indicates if you have the permission to set the file security.

**ALLOW\_DIRECTORY\_LIST\_ACCESS**

ALLOW\_DIRECTORY\_LIST\_ACCESS is the flag indicates if you have the permission to browse the directory.

**ALLOW\_FILE\_ACCESS\_FROM\_NETWORK**

ALLOW\_FILE\_ACCESS\_FROM\_NETWORK is the flag indicates if you have the permission to access the files from the network server.

**ALLOW\_NEW\_FILE\_ENCRYPTION**

ALLOW\_NEW\_FILE\_ENCRYPTION is the flag indicates if you allow to encrypt the new created files by the filter driver.

**ALLOW\_READ\_ENCRYPTED\_FILES**

ALLOW\_READ\_ENCRYPTED\_FILES is the flag indicates if you allow to read encrypted files, if it is disabled, the encrypted data will return to the users.

**ALLOW\_ALL\_SAVE\_AS**

ALLOW\_ALL\_SAVE\_AS is the flag indicates if the process has the permission to create a new file.

**ALLOW\_COPY\_PROTECTED\_FILES\_OUT**

ALLOW\_COPY\_PROTECTED\_FILES\_OUT is the flag indicates if allow the protected files being copy out of the protected folder, if allow\_all\_save\_as is true.

**ALLOW\_FILE\_MEMORY\_MAPPED**

ALLOW\_FILE\_MEMORY\_MAPPED indicates if the file can be opened with memory mapped, a file excution must be opened with memory mapped.

**DISABLE\_ENCRYPT\_DATA\_ON\_READ**

DISABLE\_ENCRYPT\_DATA\_ON\_READ is fale, and the encryption is enabled, it can encrypt the data when the application reads the file, it enables you to encrypt your files when you upload or copy out your files to other places.

**LEAST\_ACCESS\_FLAG**

LEAST\_ACCESS\_FLAG indicates the file has the least access right, it can’t be set to 0, or it will be excluded by the filter driver.

**ALLOW\_COPY\_PROTECTED\_FILES\_TO\_USB**

If this flag was disabled, the protected files can’t be copied out to the USB drive.

**ALLOW\_MAX\_RIGHT\_ACCESS**

ALLOW\_MAX\_RIGHT\_ACCESS indicates if you have the maximum access right to the file.

**Comments**

A accessFlag is the control flag of a filter rule, you can control the file access by disabling the specific flags.

### *typedef enum AESFlags*

{

*Flags\_Enabled\_Expire\_Time = 0x00000010,*

*Flags\_Enabled\_Check\_ProcessName = 0x00000020,*

*Flags\_Enabled\_Check\_UserName = 0x00000040,*

*Flags\_Enabled\_Check\_AccessFlags = 0x00000080,*

*Flags\_Enabled\_Check\_User\_Permit = 0x00000100,*

*Flags\_AES\_Key\_Was\_Embedded = 0x00000200,*

*Flags\_Enabled\_Request\_IV\_And\_Key = 0x00000400,*

*Flags\_Enabled\_Revoke\_Access\_Control = 0x00000800,*

*Flags\_Enabled\_Check\_Computer\_Id = 0x00001000,*

*Flags\_Enabled\_Check\_User\_Password = 0x00002000,*

};

**Members**

**Flags\_Enabled\_Expire\_Time**

Flags\_Enabled\_Expire\_Time is the flag which indicates that the filter driver will check if the encrypted file was expired before it can be opened.

**Flags\_Enabled\_Check\_ProcessName**

Flags\_Enabled\_Check\_ProcessName is the flag which indicates that the filter driver will check if the process has the permission to open the encrypted file.

**Flags\_Enabled\_Check\_UserName**

Flags\_Enabled\_Check\_UserName is the flag which indicates that the filter driver will check if the user has the permission to open the encrypted file.

**Flags\_Enabled\_Check\_AccessFlags**

Flags\_Enabled\_Check\_AccessFlags is the flag which indicates that the filter driver will check the access flags if the encrypted file can be opened.

**Flags\_Enabled\_Check\_User\_Permit**

Flags\_Enabled\_Check\_User\_Permit is the flag which indicates that the filter driver will require the user permission before the encrypted file can be opened.

**Flags\_AES\_KEY\_WAS\_EMBEDDED**

Flags\_AES\_KEY\_WAS\_EMBEDDED is the flag which indicates that the filter driver will user the embedded AES key to decrypt the file.

**Flags\_Enabled\_Request\_IV\_And\_Key**

Flags\_Enabled\_Request\_IV\_And\_Key is the flag which indicates that the filter driver will require encryption key and IV from user mode service to decrypt the file.

**Flags\_Enabled\_Revoke\_Access\_Control**

Flags\_Enabled\_Revoke\_Access\_Control is the flag which indicates that the filter driver will require the permission from the control server and get the encryption key and IV to decrypt the file.

**Flags\_Enabled\_Check\_Computer\_Id**

Flags\_Enabled\_Check\_Computer\_Id is the flag which indicates that the filter driver will check if the computer can access the encrypted files.

**Flags\_Enabled\_Check\_User\_Password**

Flags\_Enabled\_Check\_User\_Password is the flag which indicates that the filter driver will require the user password before the encrypted file can be opened.

**Comments**

A AESFlags is a flag inside the metadata of the encrypted file.

### *typedef struct \_AES\_DATA*

{

*ULONG VerificationKey;*

*ULONG AESFlags;*

*ULONG Version;*

*UCHAR IV[16];*

*ULONG EncryptionKeyLength;*

*UCHAR EncryptionKey[32];*

*LONGLONG FileSize;*

*ULONG CryptoType;*

*ULONG PaddingSize;*

*ULONG AESDataSize;*

*LONGLONG FileSizeOnDisk;*

*ULONG AccessFlags;*

*ULONG Reserve1;*

*ULONG Reserve2;*

*ULONG TagDataLength;*

*WCHAR TagData[1];*

}

**Members**

**EaseTagKey**

The verification key of the AES data structure, the value is 0xccb76e80.

**AESFlags**

The Flags of the encrypted file’s meta data,indicates the filter driver what action needs to do for the encrypted file opens.

**Version**

The version indicates the encryption engine OpenSSL library if it is 16, or use Micrsoft CNG library if it is 32.

**IV**

The 16 bytes initialization vector, is an arbitrary number that can be used along with an encryption key for the file encryption.

**EncryptionKeyLength**

The length of the encryption key, it should be 16,24 or 32.

**EncryptionKey**

The encryption key for the file encryption.

**FileSize**

The file size was present to the user for the encrypted file, this is file size doesn’t include the padding size and the header size.

**CryptoType**

The crypto type for the encryption, default is 0 which is using AES CTR mode.

**PaddingSize**

The padding size of the last block if it needs the padding.

**AESDataSize**

The total size of this structure, this is the size which will be appended to the encrypted file, by default is 1024.

**FileSizeOnDisk**

This is the actual physical file size of the encrytped file, includes the pading size and the header size.

**AccessFlags**

The file access control flag if the bit *Flags\_Enabled\_Check\_AccessFlags* in the AESFlags flag was enabled.

**Reserve1**

The reserve data for future use.

**Reserve2**

The reserve data for future use.

**TagDataLength**

The length of the custom data.

**TagData**

The custom tag data which was added by the user from the encryption API.

**Comments**

*AES\_DATA* structure is the metadata of the encrypted file which was appended to the end of the file.

### *Typedef enum FilterStatus*

*{*

*FILTER\_MESSAGE\_IS\_DIRTY = 0x00000001, FILTER\_COMPLETE\_PRE\_OPERATION = 0x00000002, FILTER\_DATA\_BUFFER\_IS\_UPDATED = 0x00000004,*

*};*

**Members**

**FILTER\_MESSAGE\_IS\_DIRTY**

FILTER\_MESSAGE\_IS\_DIRTY is the flag indicates the reply message was modified and needs to be processed in filter driver. Set this flag if you change the reply message.

**FILTER\_COMPLETE\_PRE\_OPERATION**

FILTER\_COMPLETE\_PRE\_OPERATION is the flag indicates the filter needs to complete this pre I/O request.Only set this flag with pre operation request when you don`t want the request goes down to the file system.

**FILTER\_DATA\_BUFFER\_IS\_UPDATED**

FILTER\_DATA\_BUFFER\_IS\_UPDATED is the flag indicates the data buffer of the reply message was updated.The filter will process this data buffer.

**Comments**

FitlerStatus is the status code which returns to the filter driver,it is for control filter driver.It instructs the filter drive what action needs to be done.

### *typedef struct \_MESSAGE\_SEND\_DATA*

*{*

*ULONG VerificationNumber;*

*ULONG MessageId;*

*ULONG FilterRuleId;*

*WCHAR RemoteIP[INET\_ADDR\_STR\_LEN];*

*PVOID FileObject;*

*PVOID FsContext;*

*ULONG MessageType;*

*ULONG ProcessId;*

*ULONG ThreadId;*

*LONGLONG Offset;*

*ULONG Length;*

*LONGLONG FileSize;*

*LONGLONG TransactionTime;*

*LONGLONG CreationTime;*

*LONGLONG LastAccessTime;*

*LONGLONG LastWriteTime;*

*ULONG FileAttributes;*

*ULONG DesiredAccess;*

*ULONG Disposition;*

*ULONG ShareAccess;*

*ULONG CreateOptions;*

*ULONG CreateStatus;*

*ULONG InfoClass;*

*ULONG Status;*

*ULONG ReturnLength;*

*ULONG FileNameLength;*

*WCHAR FileName[MAX\_FILE\_NAME\_LENGTH];*

*ULONG SidLength;*

*UCHAR Sid[MAX\_SID\_LENGTH];*

*ULONG DataBufferLength;*

*UCHAR DataBuffer[MAX\_MESSAGE\_SIZE];*

*} MESSAGE\_SEND\_DATA, \*PMESSAGE\_SEND\_DATA;*

**Members**

**VerificationNumber**

The verification number to verify the data structure integerity.

**MessageId**

This is the sequential number of the transaction.

**FilterRuleId**

This is the filter rule Id of the file I/O was managed.

**RemoteIP**

The remote IP address if the file I/O was accessed from the smb network computer.

**FileObject**

The FileObject is the pointer to the file object,it is a unique number to every file open.

**FsContext**

The FsContext is the pointer to the file context,it is unique number to the same file.

**MessageType**

MessageType is the I/O request type for this transaction.

**ProcessId**

The ProcessId is the id of the process associated with the thread that originally requested the I/O operation.

**ThreadId**

The ThreadId is the id of thread which requested the I/O operation.

**Offset**

The Offset is the read or write offset.

**Length**

The Length is the length for read or write.

**FileSize**

The FileSize is the size of the file for this I/O request.

**TransactionTime**

The transaction time in UTC format of the request.

**CreationTime**

The creation time in UTC format of the file we are requesting.

**LastAccessTime**

The last access time in UTC format of the file we are requesting.

**LastWriteTime**

The last write time in UTC format of the file we are requesting.

**FileAttributes**

The file attributes of the file we are requesting.

**DesiredAccess**

The DesiredAccess is the request access to the file for the Create I/O request,which can be summarized as read,write,both or neither zero.For more information reference the Windows API CreateFile.

**Disposition**

The disposition is the action to take on a file that exist or does not exist. For more information reference the Windows API CreateFile.

**SharedAccess**

The SharedAccess is the requested sharing mode of the file which can be read,write,both,delete,all of these,or none. For more information reference the Windows API CreateFile.

**CreateOptions**

The CreateOptions specifies the options to be applied when creating or opening the file. For more information reference the Windows API CreateFile.

**CreateStatus**

The CreateStatus is the status after the Create I/O request completed.It could be the one of the following values:

FILE\_SUPERSEDED = 0x00000000,

FILE\_OPENED = 0x00000001,

FILE\_CREATED = 0x00000002,

FILE\_OVERWRITTEN = 0x00000003,

FILE\_EXISTS = 0x00000004,

FILE\_DOES\_NOT\_EXIST = 0x00000005,

**InfoClass**

The infoClss is the information class for query/set information I/O request, or directory browsing request. For query/set security request, it is the security information.For more information reference the windows Filter API FltQueryInformationFile, FltQueryDirectoryFile,FltQuerySecurityObject.

**Status**

The Status is the I/O status which returns from the file system,indicates if the I/O request succeeded.It is only meaningful to the post I/O requests.

**ReturnLength**

The return length of the I/O for read or write.

**FileNameLength**

The file name length in byte of the file we are requesting.

**FileName**

The file name we are requesting.

**SidLength**

The length of the security identifier buffer in byte.

**Sid**

The buffer of the security identifier data.

**DataBufferLength**

The data buffer length for read, write, security, information,directory I/O requests.

**DataBuffer**

The The data buffer length for read, write, security, information, directory I/O requests.

**Comments**

The MESSAGE\_SEND\_DATA structure is used to transfer the data from kernel to the user mode application. It includes all the information needed for the user.

### *typedef struct \_PROCESS\_INFO*

*{*

*ULONG MessageId;*

*PVOID Reserve1;*

*PVOID Reserve2;*

*ULONG MessageType;*

*LONGLONG TransactionTime;*

*ULONG ProcessId;*

*ULONG ThreadId;*

*ULONG ParentProcessId;*

*ULONG CreatingProcessId;*

*ULONG CreatingThreadId;*

*ULONG DesiredAccess;*

*ULONG Operation;*

*BOOL FileOpenNameAvailable;*

*ULONG SidLength;*

*UCHAR Sid[MAX\_SID\_LENGTH];*

*ULONG FileNameLength;*

*WCHAR FileName[MAX\_FILE\_NAME\_LENGTH];*

*ULONG CommandLineLength;*

*WCHAR CommandLine[MAX\_FILE\_NAME\_LENGTH];*

*ULONG Status;*

*ULONG VerificationNumber;*

*}* ***PROCESS\_INFO****, \*****PPROCESS\_INFO****;*

**Members**

**MessageId**

This is the sequential number of the transaction.

**Reserve1**

The reserve data field1.

**Reserve2**

The reserve data field2.

**MessageType**

MessageType is the process message type which is the filter command, reference filter command enumeration.

**TransactionTime**

The transaction time in UTC format of the request.

**ProcessId**

The ProcessId is the id of the current process associated with the thread that originally requested the process operation.

**ThreadId**

The thread Id of the current operation thread.

**ParentProcessId**

The process Id of the parent process for the new process. Note that the parent process is not necessarily the same process as the process that created the new process.

**CreatingProcessId**

The process Id of the process that created new process.

**CreatingThreadId**

The thread Id of the thread that created the new process.

**DesiredAccess**

An ACCESS\_MASK value that specifies the access rights to grant for the handle.

**Operation**

The type of handle operation, this member might be one of the following values:OB\_OPERATION\_HANDLE\_CREATE, OB\_OPERATION\_HANDLE\_DUPLICATE.

**FileOperNameAvailable**

A boolean value that specifies whether the ImageFilename member contains the exact file name that is used to open the process executable file.

**SidLength**

The length of the security identifier buffer in byte.

**Sid**

The buffer of the security identifier data.

**ImageFileNameLength**

The image file name length in byte of the file name that is used to open the process executable file.

**ImageFileName**

The file name that is used to open the process executable file.

**CommandLineLength**

The length in byte of the command line.

**CommandLine**

The process executable file and command line.

**Status**

The Status is the I/O status which returns from the file system,indicates if the I/O request succeeded.

**VerificationNumber**

The verification number to verify the data structure integerity.

**Comments**

Replace MESSAGE\_SEND\_DATA structure with this one when process filter driver sends the process operation callback notification, is used to transfer the data from kernel to the user mode application.

### *typedef struct \_MESSAGE\_REPLY\_DATA*

*{*

*ULONG MessageId;*

*ULONG MessageType;*

*ULONG ReturnStatus;*

*ULONG FilterStatus;*

*union*

*{*

*Struct{*

*ULONG DataBufferLength;*

*UCHAR DataBuffer[MAX\_MESSAGE\_SIZE];*

*}Data;*

*Struct{*

*ULONG AESDataLength;*

*UCHAR IV[16];*

*ULONG EncryptionKeyLength;*

*UCHAR EncryptionKey[1];*

*}AESData;*

*Struct{*

*ULONG UserNameLength;*

*UCHAR UserName[1];*

*}UserInfo;*

*Struct{*

*ULONG FileNameLength;*

*UCHAR FileName[1];*

*}FileInfo;*

*}ReplyData*

*} MESSAGE\_REPLY\_DATA, \*PMESSAGE\_REPLY\_DATA;*

**Members**

**MessageId**

This is the sequential number of the transaction.

**MessageType**

MessageType is the I/O request type for this transaction. Reference MessageType enum type.

**ReturnStatus**

The ReturnStatus is the I/O status which returns to filter driver, and filter will return this status to the user application for the request.

**FilterStatus**

The FitlerStatus is the status code which returns to the filter driver,it instructs the filter what process needs to be done. For more information reference the FilterStatus enum.

**DataBufferLength**

The data buffer length which returns to the filter driver.

**DataBuffer**

The data buffer which returns to the filter driver.

**AESDataLength**

The total length of the AESData.

**IV**

The 16 bytes initialization vector returns to filter driver.

**EncryptionKeyLength**

The length of the encryption key.

**EncrytionKey**

The encrytion key returns to the filter driver.

**UserNameLength**

The length of the user name.

**UserName**

The user name buffer.

**FileNameLength**

The length of the file name.

**FileName**

The file name buffer.

**Comments**

MESSAGE\_REPLY\_DATA is only for control filter, when it needs to change the data or status of the I/O request. To update the reply data buffer, you must understand the format of the buffer, incorrect data could cause your system unfunctional, even crash.

## Types

### typedef BOOL (\_\_stdcall \*Proto\_Message\_Callback)(

IN PMESSAGE\_SEND\_DATA pSendMessage,

IN OUT PMESSAGE\_REPLY\_DATA pReplyMessage)

**Comments**

This is the proto type of the message callback function. The function will be called when the registed I/O requests match the filter rule. The second parameter “pReplyMessage” is always NULL for the file system monitor filter.

### typedef VOID (\_\_stdcall \*Proto\_Disconnect\_Callback)()

**Comments**

This is the proto type of disconnect function.The function will be called when the connection to the filter is disconnected.

## Exported API

***BOOL***

### *InstallDriver()*

**Return Value**

Return true if it succeeds, else return false.

**Comments**

Install the EaseFilter driver to the system.To install the driver you need the administrator permission.

***BOOL***

### *UnInstallDriver()*

**Return Value**

Return true if it succeeds, else return false.

**Comments**

UnInstall the EaseFilter driver from the system. To UnInstall the driver you need the administrator permission.

***BOOL***

### *SetRegistrationKey(*

*IN WCHAR\* RegisterKey)*

**Parameters**

**RegisterKey**

Your register key.

**Return Value**

Return true if it succeeds, else return false.

**Comments**

You have to set the registration key before you can start the filter.

***BOOL***

### *RegisterMessageCallback(*

*ULONG ThreadCount,*

*Proto\_Message\_Callback MessageCallback,*

*Proto\_Disconnect\_Callback DisconnectCallback )*

**Parameters**

**ThreadCount**

The number of threads used for connection to the filter.

**MessageCallback**

The message callback function for the registered I/O requests.

**DisconnectCallback**

The disconnect callback function when the connection is disconnected.

**Return Value**

Return true if it succeeds, else return false.

**Comments**

RegisterMessageCallback is the first API you need to call, it is the API start the filter and create the connection to the filter.

***VOID***

### *Disconnect()*

**Comments**

Disconnect is the API when you want to stop filter and filter connection.

***BOOL***

### *GetLastErrorMessage(WCHAR\* Buffer, PULONG BufferLength)*

**Parameters**

**Buffer**

This the pointer of the buffer to receive the last error message.

**BufferLength**

The length of the buffer.

**Return Value**

Return true if it succeeds,else return false if the buffer length is not big enough to contain the message,and the BufferLength is set with the right size needed.

**Comments**

This API is called right after if the other API is failed. It will return the error message.

***BOOL***

### *ResetConfigData();*

**Return Value**

Return true if it succeeds, else return false.

**Comments**

ResetConfigData is the API reset all the configuration of the filter, it will clear up all the setting includes the filter rules.

***BOOL***

### *SetFilterType(ULONG FilterType)*

**Parameters**

**FilterType**

The type of the filter you want to set. There are FILE\_SYSTEM\_MONITOR filter and FILE\_SYSTEM\_CONTROL filter.

**Return Value**

Return true if it succeeds, else return false.

**Comments**

The default filter type is file system monitor filter.

***BOOL***

### *SetConnectionTimeout(ULONG TimeOutInSeconds)*

**Parameters**

**TimeOutInSeconds**

The value of the filter wait time out.

**Return Value**

Return true if it succeeds, else return false.

**Comments**

This is the maixmum time for the filter driver wait for the response from user mode, the user mode application should return as fast as possible, or it will block the system requests.Set it bigger if your application needs to process with more time.

***BOOL***

### *SetVolumeControlFlag(ULONG VolumeControlFlag)*

**Parameters**

**VolumeControlFlag**

The value of the volume control flag.

**Return Value**

Return true if it succeeds, else return false.

**Comments**

This is the API to register the notification when the filter driver attached the volume or detached the volume. You also can prevent the attached volumes from being formatted or dismounted.

***BOOL***

### *AddFileFilterRule(*

*IN ULONG AccessFlag,*

*IN WCHAR\* FilterMask,*

*IN BOOLEAN IsResident,*

*IN ULONG FilterRuleId*

*)*

**Parameters**

**AccessFlag**

The AccessFlag of this filter rule.

**FilterMask**

The FilterMask set the target folder or files.The mask is dos format,it can include wild character ‘\*’or ‘?’. For example:

C:\test\\*txt

The filter only monitor the files end with ‘txt’ in the folder c:\test.

**IsResident**

The flag indicates if the filter rule will be saved to the registry.

**FilterRuleId**

The Id of this filter rule, when you get the I/O callback or the notification, the filter rule Id in the data strucure “MessageSend” can tell you which filter rule was applied to this I/O.

**Return Value**

Return true if it succeeds, else return false.

**Comments**

AddFileFilterRule is the API to setup the filter rule,You can set up multiple filte rules, the FilterMask must be different, if the FilterMask is the same, it will overwrite the previous one.

***BOOL***

### *RegisterFileChangeEventToFilterRule(*

*IN WCHAR\* FilterMask,*

*IN ULONG EventType*

*)*

**Parameters**

**FilterMask**

The FilterMask which was set in API AddFileFilterRule.

**EventType**

The event types were registered to the filter rule, to monitor the file events.

**Comments**

If you want to monitor the file events for the filter rule, this is the API to register the event types.

***BOOL***

### *RegisterMonitorIOToFilterRule(*

*IN WCHAR\* FilterMask,*

*IN ULONG RegisterIO*

*)*

**Parameters**

**FilterMask**

The FilterMask which was set in API AddFileFilterRule.

**RegisterIO**

The IOs were registered to the filter rule,only post-IOs can be registered, it was used to monitor the file IOs, when it was triggered, filter driver will send the notification to the user.

**Comments**

If you want to get the notification of the file IOs for the filter rule, this is the API to register the IOs which you are interested.

***BOOL***

### *RegisterControlIOToFilterRule(*

*IN WCHAR\* FilterMask,*

*IN ULONG RegisterIO*

*)*

**Parameters**

**FilterMask**

The FilterMask which was set in API AddFileFilterRule.

**RegisterIO**

The IOs were registered to the filter rule, were used to control the file IOs, when it was triggered, filter driver will send the notification to the user, block and wait for the response, it can control the IOs data and status based on the return result.

**Comments**

If you want to control the file requests, this is the API to register the IOs which you are interested.

***BOOL***

### *AddRegisterIOFilterToFilterRule(*

*IN WCHAR\* FilterMask,*

*IN ULONG FilterByDesiredAccess,*

*IN ULONG FilterByDisposition,*

*IN ULONG FilterByCreateOptions*

*)*

**Parameters**

**FilterMask**

The FilterMask which was set in API AddFileFilterRule.

**FilterByDesiredAccess**

Filter the register IO option with file opens DesiredAccess.

**FilterByDisposition**

Filter the register IO option with file opens Disposition.

**FilterByCreateOptions**

Filter the register IO option with file opens CreateOptions.

**Comments**

Filter the callback IOs by the file open options if the callback IOs were registered.

***BOOL***

### *AddEncryptionKeyToFilterRule(*

*IN WCHAR\* FilterMask,*

*IN ULONG EncryptionKeyLength,*

*IN UCHAR\* EncryptionKey*

*)*

**Parameters**

**FilterMask**

The FilterMask which was set in API AddFileFilterRule.

**EncryptionKeyLength**

The length of the encryption key.

**EncryptionKey**

The encryption key for the filter rule.

**Return Value**

Return true if it succeeds, else return false.

**Comments**

If the encryption was enabled in the access flag in the API AddFileFilterRule, this is the API to add the encryption key for the filter rule, every file will use an unique iv.

***BOOL***

### *AddEncryptionKeyAndIVToFilterRule(*

*IN WCHAR\* FilterMask,*

*IN ULONG EncryptionKeyLength,*

*IN UCHAR\* EncryptionKey,*

*IN ULONG IVLength,*

*IN UCHAR\* IV*

*)*

**Parameters**

**FilterMask**

The FilterMask which was set in API AddFileFilterRule.

**EncryptionKeyLength**

The length of the encryption key.

**EncryptionKey**

The encryption key for the filter rule.

**IVLength**

The length of the encryption iv.

**IV**

The encryption iv for the filter rule.

**Return Value**

Return true if it succeeds, else return false.

**Comments**

If the encryption was enabled in the access flag in the API AddFileFilterRule, this is the API to add the encryption key and iv for the filter rule, all files in this filter rule will use the same key and iv.

***BOOL***

### *AddReparseFileMaskToFilterRule(*

*IN WCHAR\* FilterMask,*

*IN WCHAR\* ReparseFilterMask*

*)*

**Parameters**

**FilterMask**

The FilterMask which was set in API AddFileFilterRule .

**ReparseFilterMask**

The reparse folder mask, it can include the wild character, but it must match the wild character in FilterMask.

For example:

FilterMask = c:\test\\*txt

ReparseFilterMask = d:\reparse\\*doc

If you open file c:\test\MyTest.txt, it will reparse to the file d:\reparse\MyTest.doc.

**Return Value**

Return true if it succeeds, else return false.

**Comments**

If the REPARSE\_FILE\_OPEN was enabled in the access flag in the API AddFileFilterRule , this is the API to add the reparse filter mask for the filter rule.

***BOOL***

### *AddHiddenFileMaskToFilterRule(*

*IN WCHAR\* FilterMask,*

*IN WCHAR\* HiddenFileFilterMask*

*)*

**Parameters**

**FilterMask**

The FilterMask which was set in API AddFileFilterRule .

**HiddenFileFilterMask**

The hidden file filter mask for the files to be hidden.

For example:

FilterMask = c:\hideFilesTest\\\*

HiddenFileFilterMask = \*.doc

If you open folder c:\hideFilesTest, all the files with extension .doc won’t show up in the folder.

**Return Value**

Return true if it succeeds, else return false.

**Comments**

If the HIDE\_FILES\_IN\_DIRECTORY\_BROWSING was enabled in the access flag in the API AddFileFilterRule , this is the API to add the hidden filter mask for the filter rule.

***BOOL***

### *AddExcludeFileMaskToFilterRule(*

*IN WCHAR\* FilterMask,*

*IN WCHAR\* ExcludeFileFilterMask*

*)*

**Parameters**

**FilterMask**

The FilterMask which was set in API AddFileFilterRule .

**ExcludeFileFilterMask**

The file filter mask to be excluded.

For example:

FilterMask = \*.txt

ExcludeFileFilterMask = c:\windows\\*

The filter driver target file is all the files with extension .txt except the files in folder c:\windows and its subfolders.

**Return Value**

Return true if it succeeds, else return false.

**Comments**

This is the API to add the exclude file filter mask for the filter rule which was set in AddFileFilterRule .

***BOOL***

### *AddIncludeProcessIdToFilterRule(*

*IN WCHAR\* FilterMask,*

*IN ULONG IncludeProcessId*

*)*

**Parameters**

**FilterMask**

The FilterMask which was set in API AddFileFilterRule .

**IncludeProcessId**

The process Id to be included by filter driver.

**Return Value**

Return true if it succeeds, else return false.

**Comments**

This is the API to add the include process Id for the filter rule which was set in AddFileFilterRule ,only the files opened by the processes in the included process Ids and process names will be monitored by the filter driver.

***BOOL***

### *AddExcludeProcessIdToFilterRule(*

*IN WCHAR\* FilterMask,*

*IN ULONG ExcludeProcessId*

*)*

**Parameters**

**FilterMask**

The FilterMask which was set in API AddFileFilterRule .

**ExcludeProcessId**

The process Id to be excluded by filter driver.

**Return Value**

Return true if it succeeds, else return false.

**Comments**

This is the API to add the exclude process Id for the filter rule which was set in AddFileFilterRule , all the files were opened by the processes in the excluded process Ids and process names won’t be monitored by the filter driver.

***BOOL***

### *AddIncludeProcessNameToFilterRule(*

*IN WCHAR\* FilterMask,*

*IN WCHAR\* IncludeProcessName*

*)*

**Parameters**

**FilterMask**

The FilterMask which was set in API AddFileFilterRule .

**IncludeProcessName**

The process name to be included by filter driver.

**Return Value**

Return true if it succeeds, else return false.

**Comments**

This is the API to add the include process name for the filter rule which was set in AddFileFilterRule , only the files opened by the processes in the included process Ids and process names will be monitored by the filter driver.

***BOOL***

### *AddExcludeProcessNameToFilterRule(*

*IN WCHAR\* FilterMask,*

*IN WCHAR\* ExcludeProcessName*

*)*

**Parameters**

**FilterMask**

The FilterMask which was set in API AddFileFilterRule .

**ExcludeProcessName**

The process name to be excluded by filter driver.

**Return Value**

Return true if it succeeds, else return false.

**Comments**

This is the API to add the exclude process name for the filter rule which was set in AddFileFilterRule , all the files were opened by the processes in the excluded process Ids and process names won’t be monitored by the filter driver.

***BOOL***

### *AddIncludeUserNameToFilterRule(*

*IN WCHAR\* FilterMask,*

*IN WCHAR\* IncludeUserName*

*)*

**Parameters**

**FilterMask**

The FilterMask which was set in API AddFileFilterRule .

**IncludeUserName**

The user name to be included by filter driver.

**Return Value**

Return true if it succeeds, else return false.

**Comments**

This is the API to add the include user name for the filter rule which was set in AddFileFilterRule ,only the files were opened by the useres in the included user names will be monitored by the filter driver.

***BOOL***

### *AddExcludeUserNameToFilterRule(*

*IN WCHAR\* FilterMask,*

*IN WCHAR\* ExcludeUserName*

*)*

**Parameters**

**FilterMask**

The FilterMask which was set in API AddFileFilterRule .

**ExcludeUserName**

The process name to be excluded by filter driver.

**Return Value**

Return true if it succeeds, else return false.

**Comments**

This is the API to add the exclude user name for the filter rule which was set in AddFileFilterRule , all the files were opened by the users in the excluded user names won’t be monitored by the filter driver.

***BOOL***

### *AddSha256ProcessAcessRightsToFilterRule(*

*IN WCHAR\* FilterMask,*

*IN UCHAR\* ImageSHA256,*

*IN ULONG HashLength,*

*IN ULONG AccessFlag*

*)*

**Parameters**

**FilterMask**

The FilterMask which was set in API AddFileFilterRule .

**ImageSHA256**

The sha256 hash of the executable binary.

**HashLength**

The length of the sha256.

**AccessFlag**

The access rights for the process.

**Return Value**

Return true if it succeeds, else return false.

**Comments**

This is the API to add the access flags to the specific process with the same sha256,when the process accesses the files, it needs to check permission from access flags firt to allow or deny the file access.

***BOOL***

### *AddSignedProcessAcessRightsToFilterRule(*

*IN WCHAR\* FilterMask,*

*IN WCHAR\* CertificateName,*

*IN ULONG LengthOfCertificate,*

*IN ULONG AccessFlag*

*)*

**Parameters**

**FilterMask**

The FilterMask which was set in API AddFileFilterRule .

**CertificateName**

The subject name of the certificate of the signed process.

**LengthOfCertificate**

The length of the certificate.

**AccessFlag**

The access rights for the process.

**Return Value**

Return true if it succeeds, else return false.

**Comments**

This is the API to add the access flags to the signed process with the specific subject name of the certificate, when the signed process accesses the files, it needs to check permission from access flags firt to allow or deny the file access.

***BOOL***

### *AddProcessRightsToFilterRule(*

*IN WCHAR\* FilterMask,*

*IN WCHAR\* ProcessName,*

*IN ULONG AccessFlag*

*)*

**Parameters**

**FilterMask**

The FilterMask which was set in API AddFileFilterRule .

**ProcessName**

The process name to be set access rights.

**AccessFlag**

The access rights for the process.

**Return Value**

Return true if it succeeds, else return false.

**Comments**

This is the API to add the access flags to the specific process,when the process accesses the files, it needs to check permission from access flags firt to allow or deny the file access.

***BOOL***

### *RemoveProcessRightsFromFilterRule(*

*IN WCHAR\* FilterMask,*

*IN WCHAR\* ProcessName*

*)*

**Parameters**

**FilterMask**

The FilterMask which was set in API AddFileFilterRule .

**ProcessName**

The process name to be set access rights.

**Return Value**

Return true if it succeeds, else return false.

**Comments**

This is the API to remove the process access rights setting from the filter rule.

***BOOL***

### *AddProcessIdRightsToFilterRule(*

*IN WCHAR\* FilterMask,*

*IN ULONG ProcessId,*

*IN ULONG AccessFlag*

*)*

**Parameters**

**FilterMask**

The FilterMask which was set in API AddFileFilterRule .

**ProcessId**

The process Id to be set access rights.

**AccessFlag**

The access rights for the process.

**Return Value**

Return true if it succeeds, else return false.

**Comments**

This is the API to add the access flags to the specific process with the process Id, when the process accesses the files, it needs to check permission from access flags firt to allow or deny the file access.

***BOOL***

### *AddUserRightsToFilterRule(*

*IN WCHAR\* FilterMask,*

*IN WCHAR\* UserName,*

*IN ULONG AccessFlag*

*)*

**Parameters**

**FilterMask**

The FilterMask which was set in API AddFileFilterRule .

**UserName**

The user name to be set access rights.

**AccessFlag**

The access rights for the user.

**Return Value**

Return true if it succeeds, else return false.

**Comments**

This is the API to add the access flags to the specific user,when the user accesses the files, it needs to check permission from access flags firt to allow or deny the file access.

***BOOL***

### *AddBooleanConfigToFilterRule(*

*IN WCHAR\* FilterMask,*

*IN ULONG BooleanConfig*

*)*

**Parameters**

**FilterMask**

The FilterMask which was set in API AddFileFilterRule .

**Booleanconfig**

The boolean config setting.

**Return Value**

Return true if it succeeds, else return false.

**Comments**

This is the API to add the boolean config setting for the filter rule which was set in AddFileFilterRule , please reference the BooleanConfig enumeration.

***BOOL***

### *RemoveFilterRule(WCHAR\* FilterMask);*

**Parameters**

**FilterMask**

The FilterMask associated to the filter rule.

**Return Value**

Return true if it succeeds, else return false.

**Comments**

You can remove the filter rule which was set by AddFileFilterRule API.

***BOOL***

### *AddIncludedProcessId(ULONG ProcessId)*

**Parameters**

**ProcessId**

The process Id you want to be included by filter.

**Return Value**

Return true if it succeeds, else return false.

**Comments**

This API let the filter dirver only intercept the I/O for the included processes,discard all other I/O from other processes, you can add multiple process Id.

***BOOL***

### *RemoveExcludeProcessId(ULONG ProcessId)*

**Parameters**

**ProcessId**

The process Id you want to remove which set by AddIncludedProcessId API.

**Return Value**

Return true if it succeeds, else return false.

**Comments**

This API removes the included process Id from filter.

***BOOL***

### *AddExcludedProcessId(ULONG ProcessId)*

**Parameters**

**ProcessId**

The process Id you want to be excluded by filter.

**Return Value**

Return true if it succeeds, else return false.

**Comments**

This API let you can bypass the filter for specific processes, you can add multiple process Id.

***BOOL***

### *RemoveExcludeProcessId(ULONG ProcessId)*

**Parameters**

**ProcessId**

The process Id you want to remove which set by AddExcludedProcessId API.

**Return Value**

Return true if it succeeds, else return false.

**Comments**

This API removes the excluded process Id from filter.

***BOOL***

### *AddRegistryFilterRule(*

*IN ULONG ProcessNameLength*

*IN WCHAR\* ProcessName,*

*IN ULONG ProcessId,*

*IN ULONG UserNameLength,*

*IN WCHAR\* UserNameFilterMask,*

*IN ULONG KeyNameLength,*

*IN WCHAR\* KeyNameFilterMask,*

*IN ULONG AccessFlag,*

*IN ULONGLONG RegCallbackClass,*

*IN BOOL IsExcludeFilter,*

*IN ULONG FilterRuleId*

*)*

**Parameters**

**ProcessNameLength**

The length of the process name string in bytes.

**ProcessName**

The process name to be filtered, use ‘\*’ to include all processes.

**ProcessId**

If the process Id is not 0, then filter with the process Id instead of the process name.

**UserNameLength**

The length of the user name string in bytes.

**UserNameFilterMask**

The user name to be filtered, use ‘\*’ to include all users.

**KeyNameLength**

The length of the process name string in bytes.

**KeyNameFilterMask**

The register key name to be filtered, use ‘\*’ to include all the keys.

**AccessFlag**

The access control flag for the registry filter rule.

**RegCallbackClass**

Register the callback class for the registry filter rule.

**IsExcludeFilter**

The flag indicates if the filter rule exclude filter rule, if it is true, the filter driver will skip all the registry operation for this filter rule.

**FilterRuleId**

The Id of this filter rule.

**Comments**

This is the API to add the registry filter rule, filter by process name.

***BOOL***

### *AddRegistryFlterRuleByName(*

*IN ULONG ProcessNameLength*

*IN WCHAR\* ProcessName,*

*IN ULONG AccessFlag,*

*IN ULONGLONG RegCallbackClass,*

*IN BOOL IsExcludeFilter*

*)*

**Parameters**

**ProcessNameLength**

The length of the process name string in bytes.

**ProcessName**

The process name to be filtered, use ‘\*’ to include all processes.

**AccessFlag**

The access control flag for the registry filter rule.

**RegCallbackClass**

Register the callback class for the registry filter rule.

**IsExcludeFilter**

The flag indicates if the filter rule exclude filter rule, if it is true, the filter driver will skip all the registry operation for this filter rule.

**Comments**

This is the API to add the registry filter rule, filter by process name.

***BOOL***

### *AddRegistryFilterRuleByProcessId(*

*IN ULONG ProcessId*

*IN ULONG AccessFlag,*

*IN ULONGLONG RegCallbackClass,*

*IN BOOL IsExcludeFilter*

*)*

**Parameters**

**ProcessId**

The process Id of the process which will be managed by registry filter driver.

**AccessFlag**

The access control flag for the registry filter rule.

**RegCallbackClass**

Register the callback class for the registry filter rule.

**IsExcludeFilter**

The flag indicates if the filter rule exclude filter rule, if it is true, the filter driver will skip all the registry operation for this filter rule.

**Comments**

This is the API to add the registry filter rule, filter by process Id.

***BOOL***

### *RemoveRegistryFilterRuleByProcessId(*

*IN ULONG ProcessId*

*)*

**Parameters**

**ProcessId**

The process Id of the process which will be managed by registry filter driver.

**Comments**

This is the API to remove the registry filter rule, filter by process Id.

***BOOL***

### *RemoveRegistryFilterRuleByName(*

*IN ULONG ProcessNameLength*

*IN WCHAR\* ProcessName*

*)*

**Parameters**

**ProcessNameLength**

The length of the process name string in bytes.

**ProcessName**

The process name to be filtered.

**Comments**

This is the API to remove the registry filter rule, filter by process name.

***BOOL***

### *RemoveRegistryFilterRuleByRegKeyName(*

*IN ULONG RegKeyNameLength*

*IN WCHAR\* RegKeyName*

*)*

**Parameters**

**RegNameLength**

The length of the registry key name string in bytes.

**RegKeyName**

The registry key name to be filtered.

**Comments**

This is the API to remove the registry filter rule, filter by registry key name.

***BOOL***

### *AddProceeFilterRule(*

*IN ULONG ProcessNameMaskLength,*

*IN WCHAR\* ProcessNameMask,*

*IN ULONG ControlFlag,*

*IN ULONG FilterRuleId*

*)*

**Parameters**

**ProcessNameLength**

The length of the process name mask string in bytes.

**ProcessName**

The process name mask to be filtered.

**ControlFlag**

The control flag for the process, reference the ProcessControlFlag enumeration.

**FilterRuleId**

The Id of this process filter rule.

**Comments**

This is the API to add the process filter rule, to prevent the process from being launched, or register the callback notification of the process operation.

***BOOL***

### *RemoveProceeFilterEntry(*

*IN ULONG ProcessNameMaskLength,*

*IN WCHAR\* ProcessNameMask*

*)*

**Parameters**

**ProcessNameLength**

The length of the process name mask string in bytes.

**ProcessName**

The process name mask to be filtered.

**Comments**

This is the API to remove the process filter rule.

***BOOL***

### *AddFileControlToProcessByName(*

*IN ULONG ProcessNameMaskLength,*

*IN WCHAR\* ProcessNameMask,*

*IN ULONG FileNameNameMaskLength,*

*IN WCHAR\* FileNameMask,*

*IN ULONG AccessFlag*

*)*

**Parameters**

**ProcessNameLength**

The length of the process name mask string in bytes.

**ProcessName**

The process name mask to be filtered.

**FileNameMaskLength**

The length of the file name mask string in bytes.

**FileNameMask**

The file name mask to be filtered.

**AccessFlag**

The file access flag which control the process file access rights.

**MonitorIO**

Register the callback notification of the monitor IO.

**ControlIO**

Register the callback notification of the control IO.

**Comments**

This is the API to add the file access rights of the specific files to the specific processes.

***BOOL***

### *AddFilterCallbackIOToProcessByName(*

*IN ULONG ProcessNameMaskLength,*

*IN WCHAR\* ProcessNameMask,*

*IN ULONG FileNameNameMaskLength,*

*IN WCHAR\* FileNameMask,*

*IN ULONG MonitorIO,*

*IN ULONG ControlIO,*

*IN ULONG FilterByDesiredAccess,*

*IN ULONG FilterByDisposition,*

*IN ULONG FilterByCreateOptions*

*)*

**Parameters**

**ProcessNameLength**

The length of the process name mask string in bytes.

**ProcessName**

The process name mask to be filtered.

**FileNameMaskLength**

The length of the file name mask string in bytes.

**FileNameMask**

The file name mask to be filtered.

**MonitorIO**

Register the callback notification of the monitor IO.

**ControlIO**

Register the callback notification of the control IO.

**FilterByDesiredAccess**

Filter the register IO option with file opens DesiredAccess.

**FilterByDisposition**

Filter the register IO option with file opens Disposition.

**FilterByCreateOptions**

Filter the register IO option with file opens CreateOptions.

**Comments**

Register the callback IOs for the process, filter the callback IOs by the file open options if they are not zero.

***BOOL***

### *RemoveFileControlFromProcessByName(*

*IN ULONG ProcessNameMaskLength,*

*IN WCHAR\* ProcessNameMask,*

*IN ULONG FileNameNameMaskLength,*

*IN WCHAR\* FileNameMask*

*)*

**Parameters**

**ProcessNameLength**

The length of the process name mask string in bytes.

**ProcessName**

The process name mask to be filtered.

**FileNameMaskLength**

The length of the file name mask string in bytes.

**FileNameMask**

The file name mask to be filtered.

**Comments**

This is the API to remove the file access rights of the specific files to the specific processes if it was set.

***BOOL***

### *AddFileControlToProcessById(*

*IN ULONG ProcessId,*

*IN ULONG FileNameNameMaskLength,*

*IN WCHAR\* FileNameMask,*

*IN ULONG AccessFlag*

*)*

**Parameters**

**ProcessId**

The process Id of the process which will be added file access rights.

**ProcessName**

The process name mask to be filtered.

**FileNameMaskLength**

The length of the file name mask string in bytes.

**FileNameMask**

The file name mask to be filtered.

**AccessFlag**

The file access flag which control the process file access rights.

**Comments**

This is the API to add the file access rights of the specific files to the specific process.

***BOOL***

### *RemoveFileControlFromProcessById(*

*IN ULONG ProcessId,*

*IN ULONG FileNameNameMaskLength,*

*IN WCHAR\* FileNameMask*

*)*

**Parameters**

**ProcessId**

The process Id of the process which will bed removed file access rights.

**FileNameMaskLength**

The length of the file name mask string in bytes.

**FileNameMask**

The file name mask to be filtered.

**Comments**

This is the API to remove the file access rights of the specific files to the specific process if it was set.

***BOOL***

### *AddProtectedProcessId(ULONG ProcessId)*

**Parameters**

**ProcessId**

The process Id you want to be protected by filter.

**Return Value**

Return true if it succeeds, else return false.

**Comments**

This API can prevent the process being terminated, you can add multiple process Id, this API is supported in OS vista or later versions.

***BOOL***

### *RemoveProtectedProcessId(ULONG ProcessId)*

**Parameters**

**ProcessId**

The process Id you want to remove which set by *Add****Protect****edProcessId* API.

**Return Value**

Return true if it succeeds, else return false.

**Comments**

This API removes the protected procss Id.

***BOOL***

### *RegisterIoRequest(ULONG RequestRegistration)*

**Parameters**

**RequestRegistration**

The RequestRegistration is the bit combination of the request type.

**Return Value**

Return true if it succeeds, else return false.

**Comments**

Register the I/O requests which you want to monitor. For File\_SYSTEM\_MONITOR filter, only post I/O requests registration are affected, since it only can get notification after the request was completed by file system.

For FILE\_SYSTEM\_CONTROL filter you can register both pre and post reqeusts. If you want to deny, cancel or return with your own data instead of going down to the file system, you need to register the pre request.

For some post I/O requests, you can’t cancel or deny it, for example Create, Set information,Set security, Write requests.

***BOOL***

### *GetFileHandleInFilter(WCHAR\* FileName,* *ULONG DesiredAccess, Handle\* FileHandle);*

**Parameters**

**FileName**

The full path of the file which you want to open.

**DesiredAccess**

The requested access to the file or device, which can be summarized as read, write, both or neither zero).

**FileHandle**

The pointer to the file handle which will receive the file handle after the file was opened.

**Return Value**

Return true if it succeeds, else return false.

**Comments**

Use this API to open the file,it will bypass the filter, avoid reentrant issue.It also will bypass the security check.Close the handle with CloseHandle win32 API.

***BOOL***

### *AESEncryptFile(*

*IN WCHAR\* FileName,*

*IN ULONG KeyLength,*

*IN UCHAR\* Key,*

*IN ULONG IVLength,*

*IN UCHAR\* IV,*

*IN BOOL AddAESData )*

**Parameters**

**FileName**

The file name to be encrypted.

**KeyLength**

The encryption key length,it has to be 16(128bits),24(192bits) or 32(256bits).

**Key**

The encryption key,it is an unsigned char array with KeyLength size.

**IVLength**

The initial vector length,if it is 0, the sysem will allocate an unique IV for the file.

**IV**

The initial vector,when IVLenght is 0, it sets to NULL.

**AddAESData**

If it is true,it will add the AESData structure to the encrypted file,then the encryption filter driver can recognize this encrypted file.

**Return Value**

Return true if it succeeds, else return false.

**Comments**

AESEncryptFile is the API to encrypt file file with AES encryption cryptographic algorithm.

***BOOL***

### *AESEncryptFileWithTag(*

*IN WCHAR\* FileName,*

*IN ULONG KeyLength,*

*IN UCHAR\* Key,*

*IN ULONG IVLength,*

*IN UCHAR\* IV,*

*IN ULONG TagDataLength,*

*IN UCHAR\* TagData )*

**Parameters**

**FileName**

The file name to be encrypted.

**KeyLength**

The encryption key length,it has to be 16(128bits),24(192bits) or 32(256bits).

**Key**

The encryption key,it is an unsigned char array with KeyLength size.

**IVLength**

The initial vector length,if it is 0, the sysem will allocate an unique IV for the file.

**IV**

The initial vector,when IVLenght is 0, it sets to NULL.

**TagDataLength**

The the length of the tag data.

**TagData**

The custom tag data which was added to the AESData structure in the encrypted file header, the user can get this custom data when the filter request the encyrption iv and key.

**Return Value**

Return true if it succeeds, else return false.

**Comments**

AESEncryptFile is the API to encrypt file file with AES encryption cryptographic algorithm.

***BOOL***

### *AESEncryptFileToFile(*

*IN WCHAR\* SourceFileName,*

*IN WCHAR\* DestFileName,*

*IN ULONG KeyLength,*

*IN UCHAR\* Key,*

*IN ULONG IVLength,*

*IN UCHAR\* IV,*

*IN BOOL AddAESData )*

**Parameters**

**SourceFileName**

The source file name to be encrypted.

**DestFileName**

The target file name was encrypted.

**KeyLength**

The encryption key length,it has to be 16(128bits),24(192bits) or 32(256bits).

**Key**

The encryption key,it is an unsigned char array with KeyLength size.

**IVLength**

The initial vector length,if it is 0, the sysem will allocate an unique IV for the file.

**IV**

The initial vector,when IVLenght is 0, it sets to NULL.

**AddAESData**

If it is true,it will add the AESData structure to the encrypted file,then the encryption filter driver can recognize this encrypted file.

**Return Value**

Return true if it succeeds, else return false.

**Comments**

AESEncryptFileToFile is the API to encrypt file file with AES encryption cryptographic algorithm.

***BOOL***

### *AESEncryptFileToFileWithTag(*

*IN WCHAR\* SourceFileName,*

*IN WCHAR\* DestFileName,*

*IN ULONG KeyLength,*

*IN UCHAR\* Key,*

*IN ULONG IVLength,*

*IN UCHAR\* IV,*

*IN ULONG TagDataLength,*

*IN UCHAR\* TagData )*

**Parameters**

**SourceFileName**

The source file name to be encrypted.

**DestFileName**

The target file name was encrypted.

**KeyLength**

The encryption key length,it has to be 16(128bits),24(192bits) or 32(256bits).

**Key**

The encryption key,it is an unsigned char array with KeyLength size.

**IVLength**

The initial vector length,if it is 0, the sysem will allocate an unique IV for the file.

**IV**

The initial vector,when IVLenght is 0, it sets to NULL.

**TagDataLength**

The the length of the tag data.

**TagData**

The custom tag data which was added to the AESData structure in the encrypted file header, the user can get this custom data when the filter request the encyrption iv and key.

**Return Value**

Return true if it succeeds, else return false.

**Comments**

AESEncryptFileToFile is the API to encrypt file file with AES encryption cryptographic algorithm.

***BOOL***

### *AESDecryptFile(*

*IN WCHAR\* FileName,*

*IN ULONG KeyLength,*

*IN UCHAR\* Key,*

*IN ULONG IVLength,*

*IN UCHAR\* IV )*

**Parameters**

**FileName**

The file name to be decrypted.

**KeyLength**

The encryption key length,it has to be 16(128bits),24(192bits) or 32(256bits).

**Key**

The encryption key,it is an unsigned char array with KeyLength size.

**IVLength**

The initial vector length,if the encrypted file already has IVTag,it will use the IV tag instead of the pass in IV, if the encrypted file doesn’t set the IV tag,then the IVLength can’t be 0, and IV can’t be NULL.

**IV**

The initial vector,when the encrypted file doesn’t set IV tag, the IV can’t be NULL, or it can be NULL.

**Return Value**

Return true if it succeeds, else return false.

**Comments**

AESDecryptFile is the API to decrypt file file with AES encryption cryptographic algorithm.

***BOOL***

### *AESDecryptFileToFile(*

*IN WCHAR\* SourceFileName,*

*IN WCHAR\* DestFileName,*

*IN ULONG KeyLength,*

*IN UCHAR\* Key,*

*IN ULONG IVLength,*

*IN UCHAR\* IV )*

**Parameters**

**SourceFileName**

The encrypted file name.

**DestFileName**

The target file name was decrypted.

**KeyLength**

The encryption key length,it has to be 16(128bits),24(192bits) or 32(256bits).

**Key**

The encryption key,it is an unsigned char array with KeyLength size.

**IVLength**

The initial vector length,if the encrypted file already has IVTag,it will use the IV tag instead of the pass in IV, if the encrypted file doesn’t set the IV tag,then the IVLength can’t be 0, and IV can’t be NULL.

**IV**

The initial vector,when the encrypted file doesn’t set IV tag, the IV can’t be NULL, or it can be NULL.

**Return Value**

Return true if it succeeds, else return false.

**Comments**

AESDecryptFileToFile is the API to decrypt file file with AES encryption cryptographic algorithm.

***BOOL***

### *GetAESHeader(*

*IN WCHAR\* FileName,*

*IN PULONG HeaderSize,*

*IN UCHAR\* Header )*

**Parameters**

**FileName**

The file name was encrypted.

**HeaderSize**

The pointer of the header buffer size.

**Header**

The header buffer to store the header data.

**Return Value**

Return true if it succeeds, else return false.

**Comments**

GetAESHeader is the API to get encrypted file header, return true if it exists, or it will return false.

***BOOL***

### *GetAESTagData(*

*IN WCHAR\* FileName,*

*IN PULONG TagDataSize,*

*IN UCHAR\* TagData )*

**Parameters**

**FileName**

The file name was encrypted.

**TagDataSize**

The pointer of the Tag Data size.

**TagData**

The custom tag data which was added to the header of the encrypted file.

**Return Value**

Return true if it succeeds, else return false.

**Comments**

GetTagData is the API to get the custom tag data from the encrypted file header, return true if it exists, or it will return false.

# How to use EaseFilter SDK

## The components

The EaseFilter file system filter SDK includes two components (EaseFlt.sys and FilterAPI.dll), The EaseFlt.sys and FilterAPI.dll are different for 32bit and 64bit windows system. EaseFlt.sys is the file system filter driver which implements all the functionalities in the file system level. FilterAPI.dll is a wrapper DLL which exports the API to the user mode applications.

To check the binary is 32 bit or 64 bit you can right click file and go to the property, then go to the “Details” tag and check the “file description” section.

## Set up the filter

Install the filter driver with *InstallDriver*() method if the driver has not been installed yet. After filter driver was installed, the filter was loaded, if not you can load the filter with command “Fltmc load EaseFlt” in dos prompt. To remove the filter driver from the system, call *UninstallDriver*() method.

## Start the filter

1. Enable the filter with API *SetRegistrationKey*(). You can request the trial license key with the link: <http://www.easefilter.com/Order.htm> or email us [info@easefilter.com](mailto:info@easevault.com)
2. Start the filter driver by registering the callback function with API RegisterMessageCallback:

*RegisterMessageCallback(FilterConnectionThreadsCount, MessageCallback, DisconnectCallback);*

1. Setup global configuration setting for the filter driver:

//*To enable the monitor/control/encryption/process/registry filter driver by setting the filter type with proper license key.*

*SetFilterType(FILE\_SYSTEM\_MONITOR|FILE\_SYSTEM\_CONTROL);*

*//Set up the connection timeout in seconds, this is the maximum time for the filter driver waiting for the response of the callback function.*

*SetConnectionTimeout(30);*

*//Set up the global Boolean config setting to enable some features: ENABLE\_SEND\_DENIED\_EVENT|ENABLE\_SEND\_DATA\_BUFFER*

*SetBooleanConfig(booleanConfig);*

*//Add or remove protected process Id to the filter driver to prevent it from being terminated ungratefully:*

*AddProtectedProcessId (processId);*

*RemoveProtectedProcessId (processId);*

*//Set up the volume control flag to block the USB read write: BLOCK\_USB\_READ|BLOCK\_USB\_WRITE*

*SetvolumeControlFlag(volumeControlFlag);*

1. Setup the file filter rule for the filter driver:

*//Setup the filter rule to* ***monitor*** *the file I/O in c:\\test folder*

*AddFileFilterRule(AccessFlags,L”C:\\test\\\*”, FALSE, FilterRuleId);*

*//Register the file change events to get the notification for below events:* FILE\_WAS\_CREATED|FILE\_WAS\_WRITTEN|FILE\_WAS\_RENAMED|FILE\_WAS\_DELETED|FILE\_SECURITY\_CHANGED|FILE\_INFO\_CHANGED

*RegisterFileChangedEventsToFilterRule(L”C:\\test\\\*”, FileChangedEvents);*

*//Register the POST IO to get the notification when the IO was processed by the file system.*

*RegisterMonitorlOToFilterRule(L”c:\\test\\\*”, POST\_NEW\_FILE\_CREATED|POST\_FILE\_RENAMED|POST\_FILE\_DELETED);*

*//Setup the filter rule to* ***control*** *the file I/O in c:\\protected folder*

*configure the access control flag to protect the folder, not allow the file being renamed or deleted.*

*AccessFlags= ALLOW\_MAX\_RIGHT\_ACCESS & (~ALLOW\_FILE\_RENAME) & (~ALLOW\_FILE\_DELETE)*

*AddFileFilterRule(AccessFlags,L”C:\\protected\\\*”, FALSE, FilterRuleId);*

*//Register the PRE IO to get the callback before the IO was processed by the file system, you can allow or deny the file I/O in the callback function.*

*RegisterControllOToFilterRule(L”c:\\protected\\\*”, PRE\_CREATE);*

*//Add or remove the access rights for a specific process to the files in the protected folder. i.e., Set full access rights for process “wordpad.exe”*

*AddProecessRightsToFilterRule(L”c:\\protected\\\*”, L”wordpad.exe”, ALLOW\_MAX\_RIGHT\_ACCESS);*

*//Hide your sensitive files from the protected folder by enabling the hide file access flag. i.e., hide the files with extension .prt in folder c:\\protected.*

*AccessFlags= ALLOW\_MAX\_RIGHT\_ACCESS | ENABLE\_HIDE\_FILES\_DIRECTORY\_BROWSING*

*AddFileFilterRule(AccessFlags,L”C:\\protected\\\*”, FALSE, FilterRuleId);*

*AddHiddenFileMaskToFilterRule(L”c:\\protected\\\*”, L”\*.prt”);*

*// Setup the filter rule for* ***AUTO FILE ENCRYPTION*** *in c:\\protected folder*

*AccessFlags= ALLOW\_MAX\_RIGHT\_ACCESS | ENABLE\_FILE\_ENCRYPTION\_RULE*

*AddFileFilterRule(AccessFlags,L”C:\\encrypt\\\*”, FALSE, FilterRuleId);*

*//Configure the processes to read the raw encrypted data for backup software or other software needs to send raw encrypted files.*

*AccessFlags= ALLOW\_MAX\_RIGHT\_ACCESS & (~ALLOW\_READ\_ENCRYPTED\_FILES)*

*AddProecessRightsToFilterRule(L”c:\\encrypt\\\*”, L”explorer.exe”, AccessFlags);*

*//Setup the* ***registry*** *filter rule to* ***monitor or protect the registry access***

*You can block the registry access, for example you block the registry key change, you also can get the notification for the registry access.*

*AddRegistryFilterRule(2,L"\*",0,2,L"\*",0,NULL,REG\_MAX\_ACCESS\_FLAG,regCallbackClass,FALSE, FilterRuleId);*

*//Setup the* ***process*** *filter rule to* ***monitor or protect the process operation***

*You can block the new process launching, you can get the notification when the process was created or terminated.*

*AddProcessFilterRule((ULONG)wcslen(ProcessFilterMask)\*sizeof(WCHAR),ProcessFilterMask,ControlFlag);*

We provide C++ example and C# example to demonstrate how to use the EaseFilter File System Monitor and Control Filter.

## C++ Example

Copy the correct version (32bit or 64bit) EaseFlt.sys, FilterAPI.DLL,FilterAPI.h and FilterAPI.lib to your folder. FilterAPI.h file includes all the functions and structures used for connecting to the filter driver. WinDataStructures.h file is part of the structures of windows API which is used in the example, for more structures please reference Microsoft MSDN website.

For monitor filter, you can get the notification with file I/O messages which include process Id, Thread Id, file name, user name, file system I/O type, etc.

For Control filter, the filter will block and wait for the response if that I/O was registered, so it is better handling this request as soon as possible, or it will block the system call.

## C# Example

Copy the correct version (32bit or 64bit) EaseFlt.sys, FilterAPI.DLL to your binary folder, then add the reference “FilterControl” project to your project.

FilterControl filterControl = new FilterControl();

if (!filterControl.StartFilter(filterType, serviceThreads, connectionTimeOut, licenseKey, ref lastError))

{

Console.WriteLine("Start Filter Service failed with error:" + lastError);

return;

}

//create a file monitor filter rule, every filter

//rule must have the unique watch path.

FileFilter fileMonitorFilter = new FileFilter(watchPath);

//Filter the file change event to monitor all file change events.

fileMonitorFilter.FileChangeEventFilter = (FilterAPI.FileChangedEvents)FilterAPI.NotifyAllFileEvents;

//register the file change callback events.

fileMonitorFilter.NotifyFileWasChanged += NotifyFileChanged;

//Filter the monitor file IO events

fileMonitorFilter.MonitorFileIOEventFilter = (ulong)(MonitorFileIOEvents.OnFileOpen | MonitorFileIOEvents.OnFileRead);

//fileMonitorFilter.OnFileOpen += OnFileOpen;

//fileMonitorFilter.OnFileRead += OnFileRead;

filterControl.AddFilter(fileMonitorFilter);

if (!filterControl.SendConfigSettingsToFilter(ref lastError))

{

Console.WriteLine("SendConfigSettingsToFilter failed." + lastError);

return;

}

For more programming reference, go to : <https://www.easefilter.com/programming.htm>