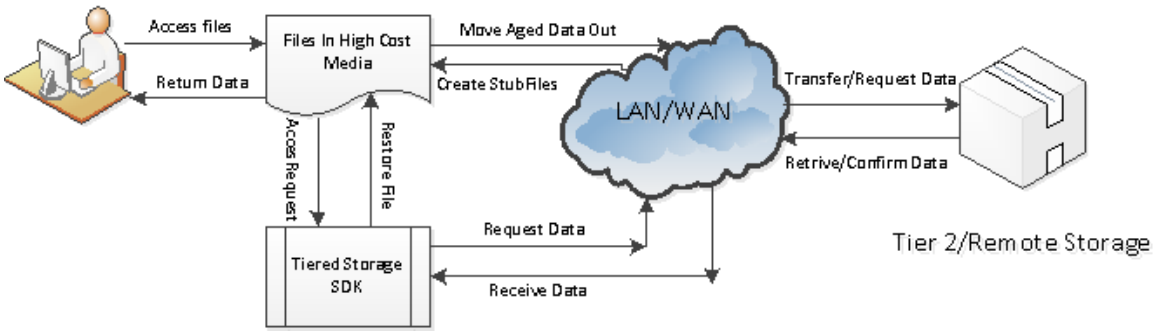
Introduction

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.

Tiered Storage File System Filter Driver SDK

Tiered Storage File System Filter Driver SDK, is a data storage technique which automatically moves data between high-cost and low-cost storage media. Tiered Storage Filter systems exist because high-speed storage devices, such as hard disk drive arrays, are more expensive (per byte stored) than slower devices, such as optical discs and magnetic tape drives. Tiered Storage Filter systems store the bulk of the enterprise's data on slower devices. A stub file is created to replace each migrated file in the fast disk drives. On the local system, a stub file looks and acts like a regular file. When you or a Windows application accesses a migrated stub file, the Windows operating system transparently directs a file access request to the Tiered Storage Filter driver. This driver retrieves the full file or block data back from the repository to which it was migrated.



Supported Platforms

- Windows 8/10 (32bit,64bit)
- Windows 2012 Server R2 (32bit,64bit)
- Windows 2008 Server R2 (32bit, 64bit)
- Windows 7 (32bit,64bit)
- Windows 2008 Server (32bit, 64bit)
- Windows Vista (32bit,64bit)
- Windows 2003 Server(32bit,64bit)
- Windows XP(32bit,64bit)

Symbol Reference

Structures, Enums

Typedef enum MessageType

```
{
      MESSAGE_TYPE_RESTORE_BLOCK_OR_FILE              = 0x00000001,
      MESSAGE_TYPE_RESTORE_FILE                       = 0x00000002,
      MESSAGE_TYPE_RESTORE_FILE_TO_CACHE              = 0x00000008,
      MESSAGE_TYPE_SEND_EVENT_NOTIFICATION            = 0x00000010,
};
```

Members

MESSAGE_TYPE_RESTORE_BLOCK_OR_FILE

It indicates that you can return block data to the filter or return the cache file name with original data.

MESSAGE_TYPE_RESTORE_FILE

It indicates that you need to restore the whole stub file with the original data.

MESSAGE_TYPE_RESTORE_FILE_TO_CACHE

It indicates that you need to return the caceh file name with the original data.

MESSAGE_TYPE_SEND_EVENT_NOTIFICATION

This is the request of from filter driver,if you register the events (CREATED,CHANGED,RENAMED,DELETED) for folders, if there are events happened,it will send the event message to the service,this request doesn't need to reply.

Comments

MessageType is the message type of the filter sending to the user mode application. The user mode applicatiob needs to handle this request properly.

Typedef enum FilterStatus

```
{
      BLOCK_DATA_WAS_RETURNED              = 0x00000008,
      CACHE_FILE_WAS_RESTORED              = 0x00000010,
};
```

Members

BLOCK_DATA_WAS_RETURNED

It indicates to the filter that the block data was returned in data buffer.

CACHE_FILE_WAS_RESTORED

It indicates to the filter that the cache file with original data was returned.

Comments

FitlerStatus is the status code which returns to the filter driver.It instructs the filter what process needs to be done.

Typedef enum EventType

```
{
      FILE_CREATED = 0x00000020,
      FILE_CHANGED = 0x00000040,
      FILE_RENAMED = 0x00000080,
      FILE_DELETED = 0x00000100,
};
```

Members

FILE_CREATED

It indicates that there are new file created in the monitor folder,the file name was stored in the field “FileName” in the messageSend struture.

FILE_CHANGED

It indicates that the file was modified in the monitor folder,the file name was stored in the field “FileName” in the messageSend struture.

FILE_RENAMEED

It indicates that the file was renamed in the monitor folder,the file name was stored in the field "FileName" in the messageSend struture,the new file name was stored in the field "DataBuffer" in the messageSend struture.

FILE_DELETED

It indicates that the file was deleted in the monitor folder,the file name was stored in the field "FileName" in the messageSend struture.

Comments

EventType is used forthe message send notification request. The field "InfoClass" is the event type for the request.

typedef struct _EASETAG_DATA

```
{
    ULONG           EaseTagKey;
    ULONG           Flags;
    ULONG           FileNameLength;
    WCHAR           FileName[1];

} EASETAG_DATA, *PEASETAG_DATA;
```

Members

EaseTagKey

This is the key to identify the reparse tag data using this structure,the key must set to 0xbba65d6f.

Flags

The resserve flags.

FileNameLength

The total length the file name in byte.

FileName

The full path file name,it can be local path or UNC path, the UNC path must start with '\\' two slash characters. It must be wide character string.

Comments

EASETAG_DATA is the data structure which use to reparse the file open to the file in EASETAG_DATA. When you create the stub file, and the tag data is following this structure, then when you or user applications open this stub file,the EaseTag filter driver will reparse the file open to the new file in kernel directly.

typedef struct _MESSAGE_SEND_DATA

```
{
    ULONG               MessageId;
    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               FileNameLength;
    WCHAR               FileName[MAX_FILE_NAME_LENGTH];
    ULONG               SidLength;
    UCHAR               Sid[MAX_SID_LENGTH];
    ULONG               DataBufferLength;
    UCHAR               DataBuffer[MAX_MESSAGE_SIZE];
    ULONG               VerificationNumber;

} MESSAGE_SEND_DATA, *PMESSAGE_SEND_DATA;
```

Members

MessageId

This is the sequential number of the transaction.

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 send notification request, it is the event type of the notification. 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.

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

VerificationNumber

The verification number to verify the data structure integerity.

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 _MESSAGE_REPLY_DATA

```
{
    ULONG           MessageId;
    ULONG           MessageType;
    ULONG           ReturnStatus;
    ULONG           FilterStatus;
    ULONG           DataBufferLength;
    UCHAR           DataBuffer[MAX_MESSAGE_SIZE];

} 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.

Comments

MESSAGE_REPLY_DATA is the data structure which return back to the filter. If you want to return the block data,you need to copy the data to the reply data buffer.

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 RegisterName,*
IN WCHAR RegisterKey)*

Parameters

RegisterName
Your register name.

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

AddFilterRule(

IN ULONG EventType,
IN WCHAR FilterMask)*

Parameters

EventType
The event type you want to register.

FilterMask
The FilterMask set the monitor 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.

Return Value
Return true if it succeeds, else return false.

Comments

AddFilterRule is the API to register the events (CREATED,CHANGED,RENAMED,DELETED) in the monitor folder.

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

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

GetFileHandleInFilter(WCHAR FileName, Handle* FileHandle);*

Parameters

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

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

CreateStubFile (
WCHAR FileName,*
LONGLONG FileSize,
ULONG FileAttributes,
ULONG TagDataLength,
BYTE TagData,*
BOOL OverwriteIfExist,
HANDLE FileHandle)*

Parameters

FileName
The full path of the file which you want to create.

FileSize
The file size you want to set for the stub file. Tt is optional if the file exist,it will use the current file size.

FileAttributes

The file attributes you want to set for the stub file. Tt is optional if the file exist,it will use the current file attribute.

TagDataLength

You can add the tag data to the stub file, the maximum length is 16*1024 byte.

TagData

You can add the tag data to the stub file.

OverwriteIfExist

The flag indicates if overwrite the file if the file exist.

FileHandle

The return file handle after the file was created.

Return Value

Return true if it succeeds, else return false.

Comments

Use this API to create the stub file, the stub file is an empty sparse file,it won't take the physical storage space.

BOOL

OpenStubFile (

WCHAR FileName,*
DWORD dwDesiredAccess,
DWORD dwShareMode,
HANDLE FileHandle)*

Parameters

FileName

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

dwDesireAccess

The requested access to the file,which can be summarized as read,write,both or neither zero. The most commonly used valuse are **GENERIC_READ**, **GENERIC_WRITE**, or both (`GENERIC_READ | GENERIC_WRITE`).

dwShareMode

The requested sharing mode of the file,it can be 0 (Share none),1(Share read),2(Share write),4(Share delete) or the combination of these values.

FileHandle

The return file handle after the file was created.

Return Value

Return true if it succeeds, else return false.

Comments

Use this API to open the stub file to file handle,if you want to get or modify the tag data,you have to use this API to get the file handle first.

BOOL

GetTagData (

HANDLE FileHandle,
ULONG TagDataLength,*
BYTE TagData)*

Parameters

FileHandle

The handle of the open stub file.

TagDataLength

The pointer to the varible for the tag data size and receive the size of the return tag data.

TagData

The pointer to the input buffer to receive the tag data.

Return Value

Return true if it succeeds, else return false.

Comments

Use this API to getthe tag data.

BOOL

AddTagData (

```
HANDLE    FileHandle,
ULONG     TagDataLength,
BYTE*     TagData)
```

Parameters

FileHandle

The handle of the open stub file.

TagDataLength

The length of the tag data.

TagData

The tag data which will add to the stub file.

Return Value

Return true if it succeeds, else return false.

Comments

Use this API to add the tag data to the stub file.

BOOL

RemoveTagData (

```
HANDLE  FileHandle,
BOOL    UpdateTimeStamp )
```

Parameters

FileHandle

The handle of the open stub file.

UpdateTimeStamp

It is optional,if it true,it will update the last write time and last access time of the stub file,or it will keep the original time stamp.

Return Value

Return true if it succeeds, else return false.

Comments

Use this API to remove the tag data.

How to use

The components

The Tiered Storage File System Filter Driver SDK includes two components (EaseTag.sys and FilterAPI.dll), The EaseTag.sys and FilterAPI.dll are different for 32bit and 64bit windows system. EaseTag.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 EaseTag" in dos prompt. To remove the filter driver from the system, call UninstallDriver() method.

Start the filter

1. Activate 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
2. After register the callback function with API RegisterMessageCallback, filter is started.

   BOOL ret = RegisterMessageCallback(FilterConnectionThreadsCount, MessageCallback, DisconnectCallback);

3. Setup the filter configuration after filter was started if needed.

C++ Example

Copy the correct version (32bit or 64bit) EaseTag.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.

C# Example

Copy the correct version (32bit or 64bit) EaseTag.sys , FilterAPI.DLL and ,EaseTag.cs to your folder. EaseTag.cs has the structures and APIs used for connecting to the filter driver.