

Contents

ファイル システム フィルター ドライバー設計ガイド
ファイル システムとフィルター ドライバーについて
ファイル システム ドライバー
ファイル システム サンプル コード
ファイル システム ドライバーのインストール
ファイル システム ドライバー用の INF ファイルの作成
ファイル システム フィルター ドライバー
ファイル システム フィルター ドライバーについて
ファイル システム フィルター ドライバーとデバイス ドライバーの類似点
ファイル システム フィルター ドライバーとデバイス ドライバーの相違点
ファイル システム ミニフィルター ドライバー
フィルター マネージャーとミニフィルター
概要
フィルター マネージャーの利点
ミニフィルター ドライバーに対するフィルター マネージャーのサポート
読み込みとアンロード
I/O 操作の処理
パラメーターの修正
ユーザー バッファーへのアクセス
ファイル名の管理
コンテキストの管理
ミニフィルター ドライバーで生成された I/O 要求
ユーザー モードとカーネル モード間のコミュニケーション
ユーザーモード ライブラリ
フィルター マネージャー操作の制御
開発とテスト ツール
レガシ フィルター ドライバーの移植ガイドライン
ミニフィルター ドライバーのインストール
ミニフィルター ドライバーのインストール

INF ファイルの作成

読み込み順序グループと高度

割り当て済み高度

ミニフィルター高度要求

再解析ポイント タグ要求

ミニフィルターの DriverEntry ルーチン

DriverEntry ルーチンの記述

ミニフィルター ドライバーの登録

フィルタリングの開始

状態を返す

ミニフィルターの FilterUnloadCallback ルーチン

FilterUnloadCallback ルーチンの登録

FilterUnloadCallback が呼び出された場合

実装

FilterUnloadCallback ルーチンの記述

通信サーバー ポートを閉じる

ミニフィルターの登録解除

グローバル クリーンアップの実行

FilterUnloadCallback ルーチンから状態を返す

操作前と操作後のコールバック ルーチン

操作前と操作後のコールバック ルーチンの記述

操作前と操作後のコールバック ルーチンの登録

ミニフィルター ドライバーでの I/O 操作のフィルタリング

操作前のコールバックの実装

操作前コールバック ルーチンの記述

I/O 操作を渡す

ミニフィルター ドライバー インスタンス スタックへ I/O 操作を渡す

FLT_PREOP_SUCCESS_WITH_CALLBACK が返される場合

FLT_PREOP_SUCCESS_NO_CALLBACK が返される場合

FLT_PREOP_SYNCHRONIZE が返される場合

操作前コールバック ルーチン内で I/O 操作を完了する

操作前コールバック ルーチン内で高速 I/O 操作を許可しない

操作前コールバック ルーチン内で高速 I/O 操作を保留にする

操作後のコールバックの実装

操作後コールバック ルーチンの記述

I/O 操作の完了処理

I/O 操作の完了処理を実行する

完了処理が安全な IRQL で実行されるようにする

操作後コールバック ルーチン内で I/O 操作を保留にする

操作後コールバック ルーチン内で I/O 操作が失敗する

I/O 操作作用のパラメーターの修正

I/O 操作のバッファ処理の実行

I/O 操作作用のバッファ処理メソッドの確認

IRP ベースまたは高速 I/O とすることが可能な操作

デバイス オブジェクト フラグに従う IRP ベースの I/O 操作

バッファ処理された I/O を常に使用する IRP ベースの I/O 操作

バッファ処理された I/O またはダイレクト I/O を常に使用しない IRP ベースの I/O 操作

IRP ベースの IOCTL と FSCTL 操作

バッファが存在しない IRP ベースの I/O 操作

I/O 操作のユーザー バッファへのアクセス

I/O 操作作用のユーザー バッファへのアクセス

操作前コールバック ルーチン内でユーザー バッファへアクセスする

操作後コールバック ルーチン内でユーザー バッファへアクセスする

コンテキストの管理

ミニフィルター ドライバーでのコンテキストの管理

コンテキストの種類の登録

コンテキストの作成

コンテキストの設定

コンテキストの取得

コンテキストの参照

コンテキストのリリース

コンテキストの削除

コンテキストの解放

ファイル システムによるコンテキストのサポート

推奨する運用方法

IRP_MJ_CREATE 操作の ECP

追加のパラメーターの作成の概要

KM ドライバーで開始された IRP_MJ_CREATE 操作に ECP をアタッチする

ミニフィルター ドライバーで ECP を使用して IRP_MJ_CREATE 操作を処理する

ユーザー定義の ECP

システム定義の ECP

ファイル システム レガシ フィルター ドライバー

レガシ フィルター ドライバーについて

レガシ フィルター ドライバーのファイル システムの概念

ドライバーのロード時期の決定方法

システム起動時のファイル システムの状態

ストレージ デバイス スタック、ストレージ ボリューム、ファイル システム スタック

ボリュームのマウント

ボリュームの認識方法

ボリュームのマウント方法

ボリュームのマウントの例

レガシ フィルター ドライバーのインストール

レガシ フィルター ドライバーのインストール

INF ファイルの作成

読み込み順序グループ

ファイル システム フィルター ドライバー クラスとクラス GUID

INF ファイルを使用したファイル システム フィルター ドライバーのインストール

INF ファイルを使用したファイル システム フィルター ドライバーのアンインストール

レガシ フィルター ドライバーの初期化

ファイル システム フィルター ドライバーの初期化

コントロール デバイス オブジェクトの作成

IRP ディスパッチ ルーチンの登録

高速 I/O ディスパッチ ルーチンの登録

FsFilter コールバック ルーチンの登録

その他必要な初期化の実行

コールバック ルーチンの登録

レジストリパス文字列のコピーの保存

状態を返す

レガシ フィルターの添付ファイル

ファイル システムまたはボリュームへのフィルターのアタッチ

フィルター デバイス オブジェクトの作成

フィルター デバイス オブジェクトのターゲット デバイス オブジェクトへのアタッチ

DO_BUFFERED_IO と DO_DIRECT_IO フラグの伝達

FILE_DEVICE_SECURE_OPEN フラグの伝達

DO_DEVICE_INITIALIZING フラグのクリア

IRP と高速 I/O

IRP と高速 I/O のフィルタリング

IRP と高速 I/O の違い

デバイス オブジェクトとレガシ フィルター ドライバー

ファイル システム フィルター ドライバーが使用するデバイス オブジェクトの種類

フィルター ドライバーのコントロール デバイス オブジェクト

ファイル システムにアタッチされるフィルター デバイス オブジェクト

ボリュームにアタッチされるフィルター デバイス オブジェクト

IRP ディスパッチ ルーチン

IRP ディスパッチ ルーチンの記述

IRP の完了

IRP を下位レベルのドライバに渡す

ディスパッチ ルーチンから返される状態

例:完了ルーチンを設定せずに IRP を渡す

ディスパッチ ルーチンに関する制約

ディスパッチ ルーチン IRQL とスレッド コンテキスト

IRP 完了ルーチン

IRP 完了ルーチンの使用

完了処理の実行方法

PendingReturned フラグの確認

完了ルーチンから返される状態

例:ディスパッチと完了間のシンプルなパススルー

完了ルーチンに関する制約

レガシ ファイル システム フィルター ドライバー内のストリーム別コンテキストの追跡

ファイル ストリーム、ストリーム コンテキスト、ストリーム別コンテキスト

ストリーム別コンテキスト構造の作成と使用

ストリーム別コンテキストの作成

ストリーム別コンテキストとファイル ストリームの関連付け

ストリーム別コンテキストの削除

レガシ ファイル システム フィルター ドライバー内のファイル別コンテキストの追跡

レガシ ファイル システム フィルター ドライバーのブロック

ネットワーク リダイレクター ドライバー

リモート ファイル システム

概要

リモート ファイル システムのコンポーネント

ネットワーク リダイレクター

ネットワーク リダイレクターについて

ネットワーク リダイレクターの基本的なアーキテクチャ

カーネルモードのネットワーク リダイレクター ドライバー

ネットワーク リダイレクターの設計とパフォーマンス

Windows NT におけるネットワーク リダイレクターの設計

Windows 2000 におけるネットワーク リダイレクターの設計

リダイレクトされたドライブのバッファー サブシステム

Oplock セマンティクス

Oplock の概要

Oplock の付与

Oplock の破損

概要

IRP_MJ_CREATE

IRP_MJ_READ

IRP_MJ_WRITE

IRP_MJ_CLEANUP

IRP_MJ_LOCK_CONTROL

IRP_MJ_SET_INFORMATION

IRP_MJ_FILE_SYSTEM_CONTROL

Oplock の破損の確認

カーネル ネットワークリダイレクター ドライバーのコンポーネント

RDBSS ドライバーとライブラリ

概要

RDBSS が提供するルーチン

RDBSS が定義するマクロ

ドライバーの登録と開始や停止の制御

プールの割り当てとフリー ルーチン

タイマーとワーカー スレッド管理

ワーク キューのディスパッチ メカニズム

診断とデバッグ

ログ記録用ルーチンとマクロ

その他のルーチン

RX_CONTEXT と IRP 管理

接続とファイル構造管理

概要

SRV_CALL 構造

NET_ROOT 構造

V_NET_ROOT 構造

FCB 構造

SRV_OPEN 構造

FOBX 構造

接続とファイル構造のロック

接続とファイル制御ブロック管理ルーチン

FCB リソース同期

バッファー処理状態の制御

低速 I/O ルーチン

名前キャッシュ管理

プレフィックス テーブル管理

消去と清掃の制御

多重 ID 管理

接続エンジン管理

カーネル ネットワーク ミニリダイレクター ドライバー

概要

カーネル ネットワーク ミニリダイレクターによって実装されるルーチン

RDBSS が使用しないルーチン

接続と名前の解決

ドライバーの開始、停止、デバイス制御

ファイル システム オブジェクトの作成と削除

ファイル システム オブジェクトの I/O ルーチン

ファイル システム オブジェクトのクエリと設定ルーチン

その他のネットワーク ミニリダイレクター ルーチン

カーネル ネットワーク リダイレクターの記述

UNC 名前付けと MUP のサポート

MUP と DFS の対話

Microsoft Windows Vista における MUP の変更点

Microsoft Windows Vista におけるトランザクションのサポート

ネットワーク リダイレクターとファイル システムの処理

ファイル システムのセキュリティに関する考慮事項

ファイル システムのセキュリティの概要

ドライバーの一般的なセキュリティに関する問題

ドライバーのセキュリティについて

セキュリティ脅威モデルを使用した設計

セキュリティ脅威モデル

ID 偽装

データの改ざん

否認

情報漏えい

サービス拒否

特権の昇格

ファイル システムを実装してセキュリティの脅威を最小限にする

概要

バッファ処理

認証と ID

アクセス制御

ハンドル管理

セキュリティのテスト

ファイル システム向けのセキュリティ機能

概要

セキュリティ記述子

ファイル システムでのセキュリティ記述子

セキュリティ識別子

アクセス マスク

アクセス制御エントリ

アクセス制御リスト

ファイル システムでの特権の管理

監査

カーネル拡張属性

ファイル システムのセキュリティに関する問題

概要

セマンティック モデルの確認

概要

作成処理

閉じるときに削除

実行可能イメージ

名前変更とハードリンク処理

ファイル情報設定処理

I/O のどちらでもない操作

ファイル システム制御処理

メディアの検証

セキュリティの検査

ファイル システムでのセキュリティ チェック

デバイス オブジェクトに対するセキュリティ記述子の適用

IRP_MJ_CREATE ディスパッチ ルーチン

IRP_MJ_CREATE でのセキュリティ チェック

IRP_MJ_CREATE での走査特権の確認

[IRP_MJ_CREATE でのその他の特殊ケースの確認](#)

[IRP_MJ_CREATE での監査の追加](#)

[IRP_MJ_CREATE でのアクセス制御リストの管理](#)

[IRP_MJ_CREATE での新規ファイルへのセキュリティの割り当て](#)

[IRP_MJ_CREATE でのクォータの処理](#)

[IRP_MJ_QUERY_SECURITY と IRP_MJ_SET_SECURITY でのセキュリティ チェック](#)

[IRP_MJ_DIRECTORY_CONTROL でのセキュリティ チェック](#)

[IRP_MJ_FILE_SYSTEM_CONTROL でのセキュリティ チェック](#)

[IRP_MJ_SET_INFORMATION でのセキュリティ チェック](#)

[偽装](#)

[処理とスレッド終了に関する問題](#)

[ファイル システム フィルター ドライバーのセキュリティに関する考慮事項](#)

[プロキシ操作](#)

[偽装](#)

[他のファイル システム フィルター ドライバーとの共存](#)

[メモリ マップ ファイル](#)

[再解析ポイント](#)

[その他の情報](#)

[Windows コンテナ用のウイルス対策の最適化](#)

[オフロード データ転送](#)

[ディスク イメージの正しい場所を見つける](#)

[重複するボリューム名がある場合のボリュームの列挙について](#)

[参照先](#)

[概要](#)

[IRP 固有の FLT_PARAMETERS](#)

[FLT_PARAMETERS \(IRP_MJ_ACQUIRE_FOR_MOD_WRITE 用\)](#)

[FLT_PARAMETERS \(IRP_MJ_ACQUIRE_FOR_SECTION_SYNCHRONIZATION 用\)](#)

[FLT_PARAMETERS \(IRP_MJ_CREATE 用\)](#)

[FLT_PARAMETERS for IRP_MJ_CREATE_MAILSLLOT](#)

[FLT_PARAMETERS for IRP_MJ_CREATE_NAMED_PIPE](#)

[FLT_PARAMETERS \(IRP_MJ_DEVICE_CONTROL と
IRP_MJ_INTERNAL_DEVICE_CONTROL 用\)](#)

[FLT_PARAMETERS \(IRP_MJ_DIRECTORY_CONTROL 用\)](#)

FLT_PARAMETERS (IRP_MJ_FAST_IO_CHECK_IF_POSSIBLE 用)

FLT_PARAMETERS (IRP_MJ_FILE_SYSTEM_CONTROL 用)

FLT_PARAMETERS (IRP_MJ_LOCK_CONTROL 用)

FLT_PARAMETERS (IRP_MJ_MDL_READ 用)

FLT_PARAMETERS (IRP_MJ_MDL_READ_COMPLETE 用)

FLT_PARAMETERS (IRP_MJ_MDL_WRITE_COMPLETE 用)

FLT_PARAMETERS (IRP_MJ_NETWORK_QUERY_OPEN 用)

FLT_PARAMETERS (IRP_MJ_PNP 用)

FLT_PARAMETERS (IRP_MJ_PREPARE_MDL_WRITE 用)

FLT_PARAMETERS (IRP_MJ_QUERY_EA 用)

FLT_PARAMETERS (IRP_MJ_QUERY_INFORMATION 用)

FLT_PARAMETERS (IRP_MJ_QUERY_OPEN 用)

FLT_PARAMETERS (IRP_MJ_QUERY_QUOTA 用)

FLT_PARAMETERS (IRP_MJ_QUERY_SECURITY 用)

FLT_PARAMETERS (IRP_MJ_QUERY_VOLUME_INFORMATION 用)

FLT_PARAMETERS (IRP_MJ_READ 用)

FLT_PARAMETERS (IRP_MJ_RELEASE_FOR_MOD_WRITE 用)

FLT_PARAMETERS (IRP_MJ_RELEASE_FOR_SECTION_SYNCHRONIZATION 用)

FLT_PARAMETERS (IRP_MJ_SET_EA 用)

FLT_PARAMETERS (IRP_MJ_SET_INFORMATION 用)

FLT_PARAMETERS (IRP_MJ_SET_QUOTA 用)

FLT_PARAMETERS (IRP_MJ_SET_SECURITY 用)

FLT_PARAMETERS (IRP_MJ_SET_VOLUME_INFORMATION 用)

FLT_PARAMETERS (IRP_MJ_SYSTEM_CONTROL 用)

FLT_PARAMETERS (IRP_MJ_VOLUME_MOUNT 用)

FLT_PARAMETERS (IRP_MJ_WRITE 用)

IRP_MJ_XXX

IRP_MJ_CLEANUP

IRP_MJ_CLOSE

IRP_MJ_CREATE

IRP_MJ_CREATE_MAILSLLOT

IRP_MJ_CREATE_NAMED_PIPE

IRP_MJ_DEVICE_CONTROL
IRP_MJ_DIRECTORY_CONTROL
IRP_MJ_FILE_SYSTEM_CONTROL
IRP_MJ_FLUSH_BUFFERS
IRP_MJ_INTERNAL_DEVICE_CONTROL
IRP_MJ_LOCK_CONTROL
IRP_MJ_PNP
IRP_MJ_QUERY_EA
IRP_MJ_QUERY_INFORMATION
IRP_MJ_QUERY_QUOTA
IRP_MJ_QUERY_SECURITY
IRP_MJ_QUERY_VOLUME_INFORMATION
IRP_MJ_READ
IRP_MJ_SET_EA
IRP_MJ_SET_INFORMATION
IRP_MJ_SET_QUOTA
IRP_MJ_SET_SECURITY
IRP_MJ_SET_VOLUME_INFORMATION
IRP_MJ_SHUTDOWN
IRP_MJ_WRITE

FSCTL_XXX

FSCTL_ADD_OVERLAY
FSCTL_DELETE_EXTERNAL_BACKING
FSCTL_DELETE_REPARSE_POINT
FSCTL_DISMOUNT_VOLUME
FSCTL_DUPLICATE_EXTENTS_TO_FILE_EX
FSCTL_ENUM_EXTERNAL_BACKING
FSCTL_ENUM_OVERLAY
FSCTL_FILE_LEVEL_TRIM
FSCTL_FIND_FILES_BY_SID
FSCTL_GET_BOOT_AREA_INFO
FSCTL_GET_EXTERNAL_BACKING

FSCTL_GET_REPARSE_POINT
FSCTL_GET_RETRIEVAL_POINTER_BASE
FSCTL_GET_RETRIEVAL_POINTERS
FSCTL_GET_WOF_VERSION
FSCTL_INVALIDATE_VOLUMES
FSCTL_IS_PATHNAME_VALID
FSCTL_IS_VOLUME_DIRTY
FSCTL_LMR_GET_LINK_TRACKING_INFORMATION
FSCTL_MARK_AS_SYSTEM_HIVE
FSCTL_MARK_VOLUME_DIRTY
FSCTL_QUERY_VOLUME_NUMA_INFO
FSCTL_OFFLOAD_READ
FSCTL_OFFLOAD_WRITE
FSCTL_OPBATCH_ACK_CLOSE_PENDING
FSCTL_OPLOCK_BREAK_ACK_NO_2
FSCTL_OPLOCK_BREAK_ACKNOWLEDGE
FSCTL_OPLOCK_BREAK_NOTIFY
FSCTL_QUERY_FILE_LAYOUT
FSCTL_QUERY_PERSISTENT_VOLUME_STATE
FSCTL_QUERY_REFS_SMR_VOLUME_INFO
FSCTL_QUERY_RETRIEVAL_POINTERS
FSCTL_REMOVE_OVERLAY
FSCTL_REQUEST_BATCH_OPLOCK
FSCTL_REQUEST_FILTER_OPLOCK
FSCTL_REQUEST_OPLOCK
FSCTL_REQUEST_OPLOCK_LEVEL_1
FSCTL_REQUEST_OPLOCK_LEVEL_2
FSCTL_SET_EXTERNAL_BACKING
FSCTL_SET_PERSISTENT_VOLUME_STATE
FSCTL_SET_REFS_SMR_VOLUME_GC_PARAMETERS
FSCTL_SET_REPARSE_POINT
FSCTL_SET_ZERO_DATA

FSCTL_SUSPEND_OVERLAY
FSCTL_UPDATE_OVERLAY
CcIsThereDirtyLoggedPages
CcSetLogHandleForFileEx
CcSetLoggedDataThreshold
CcSetReadAheadGranularityEx
CcSetReadAheadGranularityEx
READ_AHEAD_PARAMETERS
CcUnmapFileOffsetFromSystemCache
ExDisableResourceBoost
FsRtlAllocatePool
FsRtlAllocatePoolWithQuota
FsRtlEnterFileSystem
FsRtlExitFileSystem
FltAcquireResourceExclusive
FltAcquireResourceShared
FltReleaseResource
KeAcquireQueuedSpinLock
MRxCollapseOpen
MRxCreate
MRxDevFcbXXXControlFile
MRxExtendForNonCache
MRxFlush
MRxLowIOSubmitLOW[IO_OP_EXCLUSIVELOCK]
MRxLowIOSubmitLOW[IO_OP_FSCTL]
MRxLowIOSubmitLOW[IO_OP_IOCTL]
MRxLowIOSubmitLOW[IO_OP_NOTIFY_CHANGE_DIRECTORY]
MRxLowIOSubmitLOW[IO_OP_READ]
MRxLowIOSubmitLOW[IO_OP_SHAREDLOCK]
MRxLowIOSubmitLOW[IO_OP_UNLOCK]
MRxLowIOSubmitLOW[IO_OP_UNLOCK_MULTIPLE]
MRxLowIOSubmitLOW[IO_OP_WRITE]

MRxQueryDirectory
MRxQueryEaInfo
MRxQueryFileInfo
MRxQueryQuotaInfo
MRxQuerySdInfo
MRxQueryVolumeInfo
MRxSetEaInfo
MRxSetFileInfo
MRxSetFileInfoAtCleanup
MRxSetQuotaInfo
MRxSetSdInfo
MRxSetVolumeInfo
MRxShouldTryToCollapseThisOpen
MRxStop
MRxTruncate
MRxZeroExtend
RxNewMapUserBuffer
_RxAllocatePoolWithTag
_RxCheckMemoryBlock
_RxFreePool
__RxSynchronizeBlockingOperationsMaybeDroppingFcbLock
SeStopImpersonatingClient
PCOMPLETE_LOCK_IRP_ROUTINE
PFREE_FUNCTION
PUNLOCK_ROUTINE
DUAL_OPLOCK_KEY_ECP_CONTEXT
OPLOCK_KEY_ECP_CONTEXT
WCIFS_REDIRECTION_ECP_CONTEXT
FLT_FILE_NAME_OPTIONS
FILE_LOCK
ACE
SECURITY_DESCRIPTOR_CONTROL

SECURITY_INFORMATION

RxAssert

RxDbgBreakPoint

ファイル システム ドライバー 設計ガイド

2020/04/27 • [Edit Online](#)

WDK のこのセクションでは、ファイル システムおよびフィルター ドライバー (ミニフィルター) に関連する概念の情報を提供します。ドライバーで実装または呼び出しできるインターフェイスを説明する参照ページについては、[ファイル システムのプログラミング リファレンス](#)に関するページを参照してください。

ファイル システム

Windows のファイル システムは、ストレージ システム上で動作するファイル システム ドライバーとして実装されます。

Windows の各システム提供のファイル システムは、信頼性の高いデータ ストレージを与えるように設計されています。さまざまな機能でユーザーの要件を満たします。Windows で使用できる標準のファイル システムには、NTFS、ExFAT、UDF、FAT32 があります。これらの各ファイル システムの機能比較については、「[ファイル システムの機能比較](#)」をご覧ください。また、Windows Server 2012 以降のバージョンで使用する [Resilient File System \(ReFS\)](#) は、スケーラブルに大容量をサポートし、ディスク上のデータ破損を検出して、修復できます。

新しいファイル システム ドライバーを開発する必要はおそらくありません。新しいファイル システム ドライバーの要件と仕様は予測できません。そのため、この設計ガイドではファイル システムの開発については説明しません。Windows で使用できるファイル システム ドライバー以外の新しいファイル システム ドライバーを開発する必要がある場合は、サンプル コードをモデルとして使用できます (詳細については以下を参照してください)。

ファイル システム フィルター ドライバー

ファイル システム フィルター ドライバーは、ファイル システムまたは別のファイル システム フィルター ドライバーを対象にした要求をインターセプトします。要求が当初の宛先に届く前にインターセプトすることで、フィルター ドライバーは、要求の当初の宛先によって提供する機能を拡張するか、その代わりにすることができます。フィルター ドライバーの例を次に示します。

- ウイルス対策フィルター
- バックアップ エージェント
- 暗号化製品

ファイル システム フィルタリング サービスは、Windows のシステム提供の[フィルター マネージャー](#)から利用できます。フィルター マネージャーからは、フィルター ドライバーを開発するためのフレームワークが提供されます。複雑なファイル I/O をすべて管理する必要がありません。フィルター マネージャーはサードパーティ フィルター ドライバーの開発を簡単にします。また、階層を割り当てることで負荷の順序を管理するなど、レガシ フィルター ドライバー モデルに関するさまざまな問題を解決します。

フィルター マネージャー モデルに対して開発されたフィルター ドライバーはミニフィルターと呼ばれています。すべてのミニフィルター ドライバーには階層が割り当てられています。階層は、I/O スタックの他のミニフィルターに相対的にミニフィルターが読み込まれる場所を決定する一意の識別子です。階層は Microsoft によって割り当てられ、管理されます。

ファイル システムとフィルター サンプル コード

ファイル システムの開発やファイル システム フィルター ドライバーの開発のサンプルなど、いくつかの Windows ドライバーのサンプルが用意されています。完全な一覧については、「[Windows ドライバー サンプル](#)」を参照してください。

ファイル システム フィルター ドライバー 認定

ファイル システムとファイル システム フィルター ドライバーの認定情報は [Windows ハードウェア ラボ キット \(HCK\)](#) にあります。ファイル システムとファイル システム フィルター ドライバーのテストは HCK の [Filter.Driver](#) カテゴリにあります。

その他の資料

このドキュメントと上記のサンプルコードと共に、次の追加リソースを利用できます。

- Microsoft に高度の割り当てを依頼するには、ミニフィルターの高度割り当てを求める電子メールを送信してください。[ミニフィルター高度要求](#)の指示に従い、要求を送信します。
- 再解析ポイントを使用するフィルター ドライバーの ID を取得するには、「[Reparse Point Request](#)」(再解析ポイントの依頼) の手順に従います。
- [OSR](#) は、ファイル システム ミニフィルター開発用のさまざまなリソースを提供します。これには、NTFDS フォーラムなど、セミナーやコミュニティのディスカッション フォーラムなどが含まれます。

ファイルシステムのサンプルコード

2020/04/24 • [Edit Online](#)

ほとんどの場合、Windows 用の完全なファイルシステムドライバーを開発する必要はありません。新しいファイルシステムドライバーを開発する場合は、[fastfatサンプル](#)をモデルとして使用できます。ファイルシステムドライバーをインストールする方法については、「[ファイルシステムドライバー用の INF ファイルの作成](#)」を参照してください。

WDK には、ファイルシステムの開発に関する概念ドキュメントは用意されていません。

Windows でのほとんどのファイルシステム関連の開発では、システムによって提供される[フィルターマネージャー](#)と対話するファイルシステムフィルター ("ミニフィルター") ドライバーを作成する必要があります。

ファイル システム ドライバーのインストール

2020/05/05 • [Edit Online](#)

ドライバーのインストールについて

セットアップアプリケーションプログラミングインターフェイス ([setupapi.log](#)) には、Windows セットアップとドライバーのインストールを制御する機能が用意されています。これには、ファイルシステムとファイルシステムフィルタードライバーのインストールが含まれます。

インストールプロセスは INF ファイルによって制御されます。INF ファイルの詳細については、次を参照してください。

- ファイルシステムドライバー固有の INF ファイル情報については、[以下](#)を参照してください。
- 一般的なドライバーのインストール (ドライバーパッケージ、INF ファイル、ドライバーの署名に関する情報を含む) の詳細については、「[デバイスとドライバーのインストール](#)」を参照してください。

INF ファイルを作成した後、通常はセットアップアプリケーションのソースコードを記述します。セットアップアプリケーションは、ユーザーモードのセットアップ関数を呼び出して、INF ファイル内の情報にアクセスし、インストール操作を実行します。

ファイルシステムフィルタードライバーのインストールとアンインストールに関する次の情報は、ファイルシステムドライバーにも適用されます。

- [INF ファイルを使用したファイル システム フィルター ドライバーのインストール](#)
- [INF ファイルを使用したファイル システム フィルター ドライバーのアンインストール](#)

ファイル システム ドライバー用の INF ファイルの作成

ファイルシステムドライバーの INF ファイルには、Setupapi.log がドライバーのインストールに使用する指示が記載されています。INF ファイルは、ドライバーを実行するために必要なファイル、およびドライバーファイルのソースディレクトリと宛先ディレクトリを指定するテキストファイルです。INF ファイルには、Setupapi.log によってレジストリに格納されるドライバー構成情報 (ドライバーの開始の種類や読み込み順序グループなど) も含まれています。

1つの INF ファイルを作成して、複数のバージョンの Windows オペレーティングシステムにドライバーをインストールすることができます。このような INF ファイルの作成の詳細については、「[複数のプラットフォームおよびオペレーティングシステム用の Inf ファイルの作成](#)」および「[国際対応の Inf ファイルの作成](#)」を参照してください。

64ビットバージョンの Windows Vista 以降では、ファイルシステムドライバー (ファイルシステム、レガシフィルター、ミニフィルタードライバー) などの非 PnP (プラグアンドプレイ) ドライバーを含むすべてのカーネルモードコンポーネントが、読み込んで実行するために署名されている必要があります。これらのバージョンの Windows オペレーティングシステムでは、次の一覧に、ファイルシステムドライバーに関連する情報を示します。

- ファイルシステムドライバーを含む、PnP 以外のドライバーの INF ファイルに[は、製造元]または[モデル]のセクションを含める必要はありません。
- [SignTool](#)コマンドラインツール (WDK インストールディレクトリの \bin\SelfSign ディレクトリにあります) を使用して、ドライバー SYS 実行可能ファイルに "sign" を直接埋め込むことができます。パフォーマンス上の理由から、ブート開始ドライバーには、埋め込み署名が含まれている必要があります。
- INF ファイルを指定すると、[Inf2Cat](#)コマンドラインツールを使用して、ドライバーパッケージのカatalog (.cat) ファイルを作成できます。
- 管理者特権では、署名されていないドライバーを Windows Vista 以降の x64 ベースのシステムにインストールすることができます。ただし、ドライバーは署名されていないため (そのため実行) に失敗します。
- 64ビットバージョンの Windows Vista の運転署名プロセスを含む、運転署名プロセスの詳細については、「[カーネルモード](#)

[コード署名チュートリアル](#)」を参照してください。

- カスタムカーネルモードの開発ツールを含む、すべてのカーネルモードコンポーネントに署名する必要があります。詳細については、「[開発およびテスト中のドライバーへの署名 \(Windows Vista 以降\)](#)」を参照してください。

INF ファイルを使用してレジストリから情報を読み取ったり、ユーザーモードアプリケーションを起動したりすることはできません。

ファイルシステムドライバーの INF ファイルのセクション

独自のファイルシステムドライバーの INF ファイルを作成するには、次の情報をガイドとして使用します。[InfVerif](#)ツールを使用して、INF ファイルの構文を確認できます。

一般に、ファイルシステムドライバーの INF ファイルには、次のセクションが含まれています。

- [バージョン \(必須\)](#)
- [DestinationDirs \(省略可能ですが推奨\)](#)
- [SourceDisksNames \(必須\)](#)
- [SourceDisksFiles \(必須\)](#)
- [DefaultInstall \(必須\)](#)
- [DefaultInstall \(必須\)](#)
- [ServiceInstall \(必須\)](#)
- [DefaultUninstall \(省略可能\)](#)
- [DefaultUninstall. Services \(省略可能\)](#)
- [文字列 \(必須\)](#)

Version セクション (必須)

[[バージョン](#)] セクションでは、次のコード例に示すように、ドライバーのバージョン情報を指定します。

```
[Version]
Signature   = "$WINDOWS NT$"
Provider    = %Msft%
DriverVer   = 08/28/2000,1.0.0.1
CatalogFile =
```

次の表に、[[バージョン](#)] セクションでファイルシステムフィルタードライバーによって指定される値を示します。

項目	値
■	"\$WINDOWS NT \$"
■■■	独自の INF ファイルでは、Microsoft 以外のプロバイダーを指定する必要があります。
DriverVer	「 INF DriverVer ■■■ 」を参照してください。
CatalogFile	このエントリを空白のままにします。将来、署名されたドライバーの WHQL が提供したカタログファイルの名前が含まれます。

DestinationDirs セクション (省略可能ですが推奨)

[Destinationdirs](#) セクションでは、ファイルシステムドライバーファイルのコピー先のディレクトリを指定します。

このセクションとServiceinstallセクションでは、システム定義の数値を使用して、既知のシステムディレクトリを指定できます。これらの値の一覧については、「[INF DestinationDirs セクション](#)」を参照してください。次のコード例では、値 "12" は Drivers ディレクトリ (%windir%\system32\drivers) を参照します。

```
[DestinationDirs]
DefaultDestDir = 12
ExampleFileSystem.DriverFiles = 12
```

SourceDisksNames セクション (必須)

[Sourcedisksnames](#) セクションでは、使用する配布メディアを指定します。

次のコード例では、[Sourcedisksnames](#) セクションに、ファイルシステムドライバー用の1つの配布メディアが一覧表示されます。メディアの一意の識別子は1です。メディアの名前は、INF ファイルのStringsセクションで定義されている% Disk1% トークンによって指定されます。

```
[SourceDisksNames]
1 = %Disk1%
```

SourceDisksFiles セクション (必須)

[Sourcedisksfiles](#) セクションでは、コピーするファイルの場所と名前を指定します。

次のコード例では、[[Sourcedisksfiles](#)] セクションに、ファイルシステムドライバー用にコピーするファイルが一覧表示され、一意の識別子が 1 (この識別子は INF ファイルの[Sourcedisksfiles](#) セクションで定義されています) であるメディアでファイルが見つかることを指定します。

```
[SourceDisksFiles]
examplefilesystem.sys = 1
```

DefaultInstall セクション (必須)

[DefaultInstall](#) セクションでは、[CopyFiles](#) ディレクティブによって、[destinationdirs](#) セクションで指定された宛先にファイルシステムドライバーのドライバーファイルがコピーされます。

NOTE

[CopyFiles](#) ディレクティブは、カタログファイルまたは INF ファイル自体を参照することはできません。これらのファイルは、Setupapi.log によって自動的にコピーされます。

1つの INF ファイルを作成して、複数のバージョンの Windows オペレーティングシステムにドライバーをインストールすることができます。この種類の INF ファイルは、オペレーティングシステムのバージョンごとに、追加の[DefaultInstall](#)、[DefaultInstall](#)、[Defaultuninstall](#)、および[defaultuninstall. services](#) セクションを作成することによって作成されます。各セクションには、適用されるオペレーティングシステムのバージョンを指定する[装飾例](#)、. ntx86、. ntia64、または nt) が付いています。この種類の INF ファイルの作成の詳細については、「[複数のプラットフォームおよびオペレーティングシステム用の Inf ファイルの作成](#)」を参照してください。

次のコード例では、[CopyFiles](#) ディレクティブを使用して、INF ファイルの "ファイルシステム. driverfiles" セクションに一覧表示されているファイルをコピーします。

```
[DefaultInstall]
OptionDesc = %ServiceDesc%
CopyFiles = ExampleFileSystem.DriverFiles

[ExampleFileSystem.DriverFiles]
examplefilesystem.sys
```

DefaultInstall セクション (必須)

DefaultInstall セクションには、特定のドライバーのサービスが読み込まれる方法とタイミングを制御する **Addservice** ディレクティブが含まれています。

次のコード例では、**Addservice** ディレクティブによってファイルシステムサービスがオペレーティングシステムに追加されます。%ServiceName% トークンには、INF ファイルの **Strings** セクションで定義されているサービス名の文字列が含まれています。例 **Filesystem.Service** は、ファイルシステムドライバーの **Serviceinstall** セクションの名前です。

```
[DefaultInstall.Services]
AddService = %ServiceName%,ExampleFileSystem.Service
```

ServiceInstall セクション (必須)

Serviceinstall セクションでは、レジストリにサブキーまたは値の名前を追加し、値を設定します。**Serviceinstall** セクションの名前は、**DefaultInstall セクションの addservice ディレクティブ** に含まれている必要があります。

次のコード例は、ファイルシステムドライバーの **Serviceinstall** セクションを示しています。

```
[ExampleFileSystem.Service]
DisplayName = %ServiceName%
Description = %ServiceDesc%
ServiceBinary = %12%\examplefilesystem.sys
ServiceType = 2 ; SERVICE_FILE_SYSTEM_DRIVER
StartType = 1 ; SERVICE_SYSTEM_START
ErrorControl = 1 ; SERVICE_ERROR_NORMAL
LoadOrderGroup = "File System"
AddReg = ExampleFileSystem.AddRegistry
```

DisplayName エントリは、サービスの名前を指定します。前の例では、サービス名の文字列は、INF ファイルの **Strings** セクションで定義されている %ServiceName% トークンによって指定されています。

Description エントリは、サービスを説明する文字列を指定します。前の例では、この文字列は、INF ファイルの **Strings** セクションで定義されている %servicedesc% token によって指定されています。

Servicebinary エントリは、サービスの実行可能ファイルへのパスを指定します。前の例では、値 12 は Drivers ディレクトリ (%windir%\system32\drivers) を表しています。

ServiceType エントリは、サービスの種類を指定します。次の表に、**ServiceType** に使用できる値と、対応するサービスの種類を示します。

I	II
0x00000001	SERVICE_KERNEL_DRIVER (デバイスドライバーサービス)
0x00000002	SERVICE_FILE_SYSTEM_DRIVER (ファイルシステムまたはファイルシステムフィルタードライバーサービス)
0x00000010	SERVICE_WIN32_OWN_PROCESS (独自のプロセスで実行される Microsoft Win32 サービス)

I	II
0x00000020	SERVICE_WIN32_SHARE_PROCESS (プロセスを共有する Win32 サービス)

ServiceType エントリは、ファイルシステムドライバの場合は常に SERVICE_FILE_SYSTEM_DRIVER に設定する必要があります。

Starttype エントリは、サービスをいつ開始するかを指定します。次の表に、Starttype とそれに対応する開始の種類に使用できる値を示します。

I	II
0x00000000	SERVICE_BOOT_START
0x00000001	SERVICE_SYSTEM_START
0x00000002	SERVICE_AUTO_START
0x00000003	SERVICE_DEMAND_START
0x00000004	SERVICE_DISABLED

これらの開始の詳細については、「[ドライバが読み込まれるタイミング](#)を決定するには」を参照してください。

X64 ベースの Windows Vista システム以降では、ブート開始ドライバのバイナリイメージファイル (開始の種類が SERVICE_BOOT_START のドライバ) には、埋め込まれた署名が含まれている必要があります。この要件により、システムブートのパフォーマンスが最適になります。詳細については、「[カーネルモードコード署名のチュートリアル](#)」を参照してください。

ドライバが読み込まれるタイミングを、Starttype および loadordergroup のエントリがどのように決定するかについては、「[ドライバが読み込まれるタイミング](#)を確認する方法」を参照してください。

Errorcontrol エントリは、システムの起動時にサービスを開始できなかった場合に実行するアクションを指定します。次の表に、errorcontrol に使用できる値と、それに対応するエラー制御値の一覧を示します。

I	II
0x00000000	SERVICE_ERROR_IGNORE (エラーをログに記録し、システムの起動を続行します。)
0x00000001	SERVICE_ERROR_NORMAL (エラーをログに記録し、ユーザーにメッセージを表示して、システムの起動を続行します。)
0x00000002	SERVICE_ERROR_SEVERE (レジストリの [前回の制御セット] に切り替え、システムの起動を続行します。)
0x00000003	SERVICE_ERROR_CRITICAL (システムスタートアップがレジストリの正常でないコントロールセットを使用していない場合は、[正常起動時] に切り替えて、もう一度やり直してください。まだ起動できない場合は、バグチェックルーチンを実行します。システムを起動するために必要なドライバだけが、INF ファイルにこの値を指定する必要があります。)

ファイルシステムドライバの場合は、Loadordergroup エントリを常に "file system" に設定する必要があります。これは、ファイルシステムフィルタードライバまたはファイルシステムミニフィルタードライバに対して指定されているものとは異なります。この場合、Loadordergroup エントリは、ファイルシステムフィルタの読み込み順序グループのいずれかに設定されます。ファイルシステムフィルタードライバとファイルシステムミニフィルタードライバに使用される読み込み順序グループの詳細については、「[ファイルシ](#)

ステムフィルタードライバーの読み込み順序グループ」および「ミニフィルタードライバーのための負荷の増加」を参照してください。

AddReg ディレクティブは、新しくインストールされたサービスのレジストリに格納される情報を含む、1つまたは複数の INF ライターで定義されたaddregistryセクションを参照します。

NOTE

初期インストール後に INF ファイルがドライバーのアップグレードにも使用される場合、**Addregistry**セクションに含まれているエントリで、0x00000002 (FLG_ADDREG_NOCLONBER) フラグを指定する必要があります。このフラグを指定すると、後続のファイルがインストールされたときに HKLM\CurrentControlSet\Services のレジストリエントリが保持されます。次に例を示します。

```
[ExampleFileSystem.AddRegistry]
HKR,Parameters,ExampleParameter,0x00010003,1
```

DefaultUninstall セクション (省略可能)

Defaultuninstallセクションは省略可能ですが、ドライバーをアンインストールできる場合は推奨されます。このファイルには、ファイルとレジストリエントリを削除するための**Delfiles**ディレクティブと**delfiles**ディレクティブが含まれています。

次のコード例では、**delfiles**ディレクティブを使用して、INF ファイルの "ファイルシステム. driverfiles" セクションに一覧表示されているファイルを削除します。

```
[DefaultUninstall]
DelFiles    = ExampleFileSystem.DriverFiles
DelReg      = ExampleFileSystem.DelRegistry
```

Delregディレクティブは、アンインストールするサービスのレジストリから削除する情報を含む、1つまたは複数の INF ライターで定義されたdelregセクションを参照します。

DefaultUninstall. Services セクション (省略可能)

Defaultuninstall. Servicesセクションは省略可能ですが、ドライバーをアンインストールできる場合は推奨されます。ファイルシステムドライバーのサービスを削除するための**DelService**ディレクティブが含まれています。

次のコード例では、**DelService**ディレクティブを使用して、ファイルシステムドライバーのサービスをオペレーティングシステムから削除します。

```
[DefaultUninstall.Services]
DelService = %ServiceName%,0x200
```

NOTE

DelServiceディレクティブは、削除前にサービスを停止するように、常に 0x200 (SPSVCINST_STOPSERVICE) フラグを指定する必要があります。

NOTE

ファイルシステム製品には、完全にはアンインストールできない特定のクラスがあります。このような状況では、アンインストールできる製品のコンポーネントをアンインストールして、アンインストールできないコンポーネントをインストールしたままにすることができます。このような製品の例として、Microsoft 単一インスタンスストア (SIS) 機能があります。

Strings セクション (必須)

Stringsセクションでは、INF ファイルで使用される% strkey% トークンを定義します。

たとえば、ファイルシステムドライバーでは、次の文字列が INF ファイルに定義されています。

```
[Strings]
Msft      = "Microsoft Corporation"
ServiceDesc = "Example File System Driver"
ServiceName = "ExampleFileSystem"
ParameterPath = "SYSTEM\CurrentControlSet\Services\ExampleFileSystem\Parameters"
Disk1     = "Example File System Driver CD"
```

ロケール固有の文字列を追加することで、単一の国際 INF ファイルを作成できます。INF ファイルの *LanguageID* セクション。国際対応の INF ファイルの詳細については、「[国際対応の Inf ファイルの作成](#)」を参照してください。

ファイル システム ドライバー用の INF ファイルの作成

2020/05/05 • [Edit Online](#)

Windows セットアップおよびデバイスインストーラーサービス (総称して「[setupapi.log](#)」といいます) は、Windows セットアップとドライバーのインストールを制御する機能を提供します。インストールプロセスは INF ファイルによって制御されます。

ファイルシステムドライバーの INF ファイルには、Setupapi.log がドライバーのインストールに使用する指示が記載されています。INF ファイルは、ドライバーを実行するために必要なファイル、およびドライバーファイルのソースディレクトリと宛先ディレクトリを指定するテキストファイルです。INF ファイルには、Setupapi.log によってレジストリに格納されるドライバー構成情報 (ドライバーの開始の種類や読み込み順序グループなど) も含まれています。

INF ファイルとその作成方法の詳細については、「[Inf ファイルおよび inf ファイルの作成](#)」セクションと「[ディレクティブ](#)」を参照してください。ドライバーの署名に関する一般的な情報については、「[ドライバーの署名](#)」を参照してください。

1つの INF ファイルを作成して、複数のバージョンの Windows オペレーティングシステムにドライバーをインストールすることができます。このような INF ファイルの作成の詳細については、「[複数のプラットフォームおよびオペレーティングシステム用の Inf ファイルの作成](#)」および「[国際対応の Inf ファイルの作成](#)」を参照してください。

64ビットバージョンの Windows Vista 以降では、ファイルシステムドライバー (ファイルシステム、レガシフィルター、ミニフィルタードライバー) などの非 PnP (プラグアンドプレイ) ドライバーを含むすべてのカーネルモードコンポーネントが、読み込んで実行するために署名されている必要があります。これらのバージョンの Windows オペレーティングシステムでは、次の一覧に、ファイルシステムドライバーに関連する情報を示します。

- ファイルシステムドライバーを含む、PnP 以外のドライバーの INF ファイルに[は、製造元]または[モデル]のセクションを含める必要はありません。
- [SignTool](#)コマンドラインツール (WDK インストールディレクトリの \bin\SelfSign ディレクトリにあります) を使用して、ドライバー SYS 実行可能ファイルに "sign" を直接埋め込むことができます。パフォーマンス上の理由から、ブート開始ドライバーには、埋め込み署名が含まれている必要があります。
- INF ファイルを指定すると、[Inf2Cat](#)コマンドラインツールを使用して、ドライバーパッケージのカタログ (.cat) ファイルを作成できます。[WHQL](#)ロゴ署名を受け取ることができるのは、カタログファイルだけです。
- 管理者特権では、署名されていないドライバーを Windows Vista 以降の x64 ベースのシステムにインストールすることができます。ただし、ドライバーは署名されていないため (そのため実行) に失敗します。
- 64ビットバージョンの Windows Vista の運転署名プロセスを含む、運転署名プロセスの詳細については、「[カーネルモードコード署名チュートリアル](#)」を参照してください。
- カスタムカーネルモードの開発ツールを含む、すべてのカーネルモードコンポーネントに署名する必要があります。詳細については、「[開発およびテスト中のドライバーへの署名 \(Windows Vista 以降\)](#)」を参照してください。

INF ファイルを使用してレジストリから情報を読み取ったり、ユーザーモードアプリケーションを起動したりすることはできません。

INF ファイルを作成した後、通常はセットアップアプリケーションのソースコードを記述します。セットアップアプリケーションは、ユーザーモードのセットアップ関数を呼び出して、INF ファイル内の情報にアクセスし、インストール操作を実行します。

独自のファイルシステムドライバーの INF ファイルを作成するには、次の情報をガイドとして使用します。[InfVerif](#)ツールを使用して、INF ファイルの構文を確認できます。

一般に、ファイルシステムドライバーの INF ファイルには、次のセクションが含まれています。

- バージョン (必須)

- DestinationDirs (省略可能ですが推奨)
- SourceDisksNames (必須)
- SourceDisksFiles (必須)
- DefaultInstall (必須)
- DefaultInstall (必須)
- ServiceInstall (必須)
- DefaultUninstall (省略可能)
- DefaultUninstall.Services (省略可能)
- 文字列 (必須)

Version セクション (必須)

[バージョン] セクションでは、次のコード例に示すように、ドライバーのバージョン情報を指定します。

```
[Version]
Signature      = "$WINDOWS NT$"
Provider       = %Msft%
DriverVer      = 08/28/2000,1.0.0.1
CatalogFile =
```

次の表に、[バージョン] セクションでファイルシステムフィルタードライバーによって指定される値を示します。

項目	値
■	"\$WINDOWS NT \$"
■■■	独自の INF ファイルでは、Microsoft 以外のプロバイダーを指定する必要があります。
DriverVer	「 INF DriverVer ■■■」を参照してください。
CatalogFile	このエントリを空白のままにします。将来、署名されたドライバーの WHQL が提供したカタログファイルの名前が含まれます。

DestinationDirs セクション (省略可能ですが推奨)

Destinationdirs セクションでは、ファイルシステムドライバーファイルのコピー先のディレクトリを指定します。

このセクションとServiceinstallセクションでは、システム定義の数値を使用して、既知のシステムディレクトリを指定できます。これらの値の一覧については、「 INF DestinationDirs」セクションを参照してください。次のコード例では、値 "12" は Drivers ディレクトリ (%windir%\system32\drivers) を参照します。

```
[DestinationDirs]
DefaultDestDir = 12
ExampleFileSystem.DriverFiles = 12
```

SourceDisksNames セクション (必須)

Sourcedisksnames セクションでは、使用する配布メディアを指定します。

次のコード例では、Sourcedisksnamesセクションに、ファイルシステムドライバー用の1つの配布メディアが一覧表示されます。メディアの一意の識別子は1です。メディアの名前は、INF ファイルのStringsセクションで定義されている% Disk1% トークンによっ

て指定されます。

```
[SourceDisksNames]
1 = %Disk1%
```

SourceDisksFiles セクション (必須)

Sourcedisksfiles セクションでは、コピーするファイルの場所と名前を指定します。

次のコード例では、[**Sourcedisksfiles**] セクションに、ファイルシステムドライバー用にコピーするファイルが一覧表示され、一意の識別子が 1 (この識別子は INF ファイルの**Sourcedisksfiles** セクションで定義されています) であるメディアでファイルが見つかることを指定します。

```
[SourceDisksFiles]
examplefilesystem.sys = 1
```

DefaultInstall セクション (必須)

DefaultInstall セクションでは、**CopyFiles** ディレクティブによって、**destinationdirs** セクションで指定された宛先にファイルシステムドライバーのドライバーファイルがコピーされます。

NOTE

CopyFiles ディレクティブは、カタログファイルまたは INF ファイル自体を参照することはできません。これらのファイルは、Setupapi.log によって自動的にコピーされます。

1つの INF ファイルを作成して、複数のバージョンの Windows オペレーティングシステムにドライバーをインストールすることができます。この種類の INF ファイルは、オペレーティングシステムのバージョンごとに、追加の**DefaultInstall**、**DefaultInstall**、**Defaultuninstall**、および**defaultuninstall. services** セクションを作成することによって作成されます。各セクションには、適用されるオペレーティングシステムのバージョンを指定する**装飾**(例、. nt86、. ntia64、または nt) が付いています。この種類の INF ファイルの作成の詳細については、「**複数のプラットフォームおよびオペレーティングシステム用の Inf ファイルの作成**」を参照してください。

次のコード例では、**CopyFiles** ディレクティブを使用して、INF ファイルの "ファイルシステム. driverfiles" セクションに一覧表示されているファイルをコピーします。

```
[DefaultInstall]
OptionDesc = %ServiceDesc%
CopyFiles = ExampleFileSystem.DriverFiles

[ExampleFileSystem.DriverFiles]
examplefilesystem.sys
```

DefaultInstall セクション (必須)

DefaultInstall セクションには、特定のドライバーのサービスが読み込まれる方法とタイミングを制御する**Addservice** ディレクティブが含まれています。

次のコード例では、**Addservice** ディレクティブによってファイルシステムサービスがオペレーティングシステムに追加されます。% ServiceName% トークンには、INF ファイルの**Strings** セクションで定義されているサービス名の文字列が含まれています。例 **Filesystem. Service** は、ファイルシステムドライバーの**Serviceinstall** セクションの名前です。

```
[DefaultInstall.Services]
AddService = %ServiceName%,ExampleFileSystem.Service
```

ServiceInstall セクション (必須)

Serviceinstallセクションでは、レジストリにサブキーまたは値の名前を追加し、値を設定します。Serviceinstallセクションの名前は、**DefaultInstall セクションのaddservice ディレクティブ**に含まれている必要があります。

次のコード例は、ファイルシステムドライバのServiceinstallセクションを示しています。

```
[ExampleFileSystem.Service]
DisplayName    = %ServiceName%
Description    = %ServiceDesc%
ServiceBinary  = %12%\examplefilesystem.sys
ServiceType    = 2 ;      SERVICE_FILE_SYSTEM_DRIVER
StartType      = 1 ;      SERVICE_SYSTEM_START
ErrorControl    = 1 ;      SERVICE_ERROR_NORMAL
LoadOrderGroup = "File System"
AddReg         = ExampleFileSystem.AddRegistry
```

DisplayNameエントリは、サービスの名前を指定します。前の例では、サービス名の文字列は、INF ファイルのStringsセクションで定義されている% ServiceName% トークンによって指定されています。

Descriptionエントリは、サービスを説明する文字列を指定します。前の例では、この文字列は、INF ファイルのStringsセクションで定義されている% servicedesc% token によって指定されています。

Servicebinaryエントリは、サービスの実行可能ファイルへのパスを指定します。前の例では、値12は Drivers ディレクトリ (%windir%\system32\drivers) を表しています。

ServiceTypeエントリは、サービスの種類を指定します。次の表に、**ServiceType**に使用できる値と、対応するサービスの種類を示します。

I	II
0x00000001	SERVICE_KERNEL_DRIVER (デバイスドライバサービス)
0x00000002	SERVICE_FILE_SYSTEM_DRIVER (ファイルシステムまたはファイルシステムフィルタドライバサービス)
0x00000010	SERVICE_WIN32_OWN_PROCESS (独自のプロセスで実行される Microsoft Win32 サービス)
0x00000020	SERVICE_WIN32_SHARE_PROCESS (プロセスを共有する Win32 サービス)

ServiceTypeエントリは、ファイルシステムドライバの場合は常に SERVICE_FILE_SYSTEM_DRIVER に設定する必要があります。

Starttypeエントリは、サービスをいつ開始するかを指定します。次の表に、**Starttype**とそれに対応する開始の種類に使用できる値を示します。

I	II
0x00000000	SERVICE_BOOT_START
0x00000001	SERVICE_SYSTEM_START
0x00000002	SERVICE_AUTO_START
0x00000003	SERVICE_DEMAND_START

I	II
0x00000004	SERVICE_DISABLED

これらの開始の種類の詳細については、「[ドライバが読み込まれるタイミング](#)を決定するには」を参照してください。

X64 ベースの Windows Vista システム以降では、ブート開始ドライバのバイナリイメージファイル (開始の種類が SERVICE_BOOT_START のドライバ) には、埋め込まれた署名が含まれている必要があります。この要件により、システムブートのパフォーマンスが最適になります。詳細については、「[カーネルモードコード署名のチュートリアル](#)」を参照してください。

ドライバが読み込まれるタイミングを、Starttypeおよびloadordergroupのエントリがどのように決定するかについては、「[ドライバが読み込まれるタイミングを確認する方法](#)」を参照してください。

Errorcontrolエントリは、システムの起動時にサービスを開始できなかった場合に実行するアクションを指定します。次の表に、errorcontrolに使用できる値と、それに対応するエラー制御値の一覧を示します。

I	II
0x00000000	SERVICE_ERROR_IGNORE (エラーをログに記録し、システムの起動を続行します。)
0x00000001	SERVICE_ERROR_NORMAL (エラーをログに記録し、ユーザーにメッセージを表示して、システムの起動を続行します)。
0x00000002	SERVICE_ERROR_SEVERE (レジストリの [前回の制御セット] に切り替え、システムの起動を続行します。)
0x00000003	SERVICE_ERROR_CRITICAL (システムスタートアップがレジストリの正常でないコントロールセットを使用していない場合は、[正常起動時] に切り替えて、もう一度やり直してください。まだ起動できない場合は、バグチェックルーチンを実行します。システムを起動するために必要なドライバだけが、INF ファイルにこの値を指定する必要があります。)

ファイルシステムドライバの場合は、Loadordergroupエントリを常に "file system" に設定する必要があります。これは、ファイルシステムフィルタードライバまたはファイルシステムミニフィルタードライバに対して指定されているものとは異なります。この場合、Loadordergroupエントリは、ファイルシステムフィルタの読み込み順序グループのいずれかに設定されます。ファイルシステムフィルタードライバとファイルシステムミニフィルタードライバに使用される読み込み順序グループの詳細については、「[ファイルシステムフィルタードライバの読み込み順序グループ](#)」および「[ミニフィルタードライバのための負荷の増加](#)」を参照してください。

AddReg ディレクティブは、新しくインストールされたサービスのレジストリに格納される情報を含む、1つまたは複数の INF ライターで定義されたaddregistryセクションを参照します。

注 初期インストール後に INF ファイルを使用してドライバをアップグレードする場合は、addregistryセクションに含まれているエントリで、0x00000002 (FLG_ADDREG_NOCLOBBER) フラグを指定する必要があります。このフラグを指定すると、後続のファイルがインストールされたときに HKLM\CurrentControlSet\Services のレジストリエントリが保持されます。次に例を示します。

```
[ExampleFileSystem.AddRegistry]
HKR,Parameters,ExampleParameter,0x00010003,1
```

DefaultUninstall セクション (省略可能)

Defaultuninstallセクションは省略可能ですが、ドライバをアンインストールできる場合は推奨されます。このファイルには、ファイルとレジストリエントリを削除するためのDelfilesディレクティブとdelfilesディレクティブが含まれています。

次のコード例では、delfilesディレクティブを使用して、INF ファイルの "ファイルシステム. driverfiles" セクションに一覧表示されているファイルを削除します。

```
[DefaultUninstall]
DelFiles    = ExampleFileSystem.DriverFiles
DelReg      = ExampleFileSystem.DelRegistry
```

Delregディレクティブは、アンインストールするサービスのレジストリから削除する情報を含む、1つまたは複数の INF ライターで定義された**delreg**セクションを参照します。

DefaultUninstall. Services セクション (省略可能)

Defaultuninstall. Servicesセクションは省略可能ですが、ドライバーをアンインストールできる場合は推奨されます。ファイルシステムドライバーのサービスを削除するための**DelService**ディレクティブが含まれています。

次のコード例では、**DelService**ディレクティブを使用して、ファイルシステムドライバーのサービスをオペレーティングシステムから削除します。

```
[DefaultUninstall.Services]
DelService = %ServiceName%,0x200
```

NOTE

DelServiceディレクティブは、削除前にサービスを停止するように、常に 0x200 (SPSVCINST_STOPSERVICE) フラグを指定する必要があります。

NOTE

ファイルシステム製品には、完全にはアンインストールできない特定のクラスがあります。このような状況では、アンインストールできる製品のコンポーネントをアンインストールして、アンインストールできないコンポーネントをインストールしたままにすることができます。このような製品の例として、Microsoft 単一インスタンスストア (SIS) 機能があります。

Strings セクション (必須)

Stringsセクションでは、INF ファイルで使用される% strkey% トークンを定義します。

たとえば、ファイルシステムドライバーでは、次の文字列が INF ファイルに定義されています。

```
[Strings]
Msft      = "Microsoft Corporation"
ServiceDesc = "Example File System Driver"
ServiceName = "ExampleFileSystem"
ParameterPath = "SYSTEM\CurrentControlSet\Services\ExampleFileSystem\Parameters"
Disk1     = "Example File System Driver CD"
```

ロケール固有の文字列を追加することで、単一の国際 INF ファイルを作成できます。INF ファイルの*LanguageID*セクション。国際対応の INF ファイルの詳細については、「[国際対応の Inf ファイルの作成](#)」を参照してください。

ファイル システム フィルター ドライバーについて

2020/04/24 • [Edit Online](#)

Windows 上のファイルシステムフィルタードライバー

ファイルシステムフィルタードライバーは、ファイルシステムの動作に値を追加したり、ファイルシステムの動作を変更したりするためのオプションのドライバーです。ファイルシステムフィルタードライバーは、Windows executive の一部として実行されるカーネルモードのコンポーネントです。

ファイルシステムフィルタードライバーは、1つまたは複数のファイルシステムまたはファイルシステムボリュームの i/o 操作をフィルター処理できます。フィルターは、ドライバーの性質によっては、*ログ記録*、*監視*、*変更*、または *回避* することを意味します。ファイルシステムフィルタードライバーの一般的なアプリケーションには、ウイルス対策ユーティリティ、暗号化プログラム、階層型記憶域管理システムなどがあります。

Windows には、次の2つのファイルシステムフィルターモデルがあります。

- **ミニフィルターモデル**。ミニフィルターでは、システムによって提供されるフィルターマネージャーサポートを使用します。これにより、フィルターの開発が簡素化されます。
- **レガシファイルシステムフィルターモデル**

NOTE

最適な信頼性とパフォーマンスを得るには、従来のファイルシステムフィルタードライバーではなく、フィルターマネージャーをサポートする **ファイルシステムミニフィルタードライバー** を使用します。レガシドライバーをミニフィルタードライバーに移植する方法については、「**レガシフィルタードライバーを移植するためのガイドライン**」を参照してください。

ファイルシステムフィルタードライバーはデバイスドライバーではありません

デバイスドライバーは、特定のハードウェア i/o デバイスを制御するソフトウェアコンポーネントです。たとえば、dvd 記憶装置ドライバーは DVD ドライブを制御します。

これに対し、ファイルシステムフィルタードライバーは、1つまたは複数のファイルシステムと連携して、ファイル i/o 操作を管理します。これに該当する操作は次のとおりです。

- ファイルとディレクトリの作成、開始、終了、および列挙
- ファイル、ディレクトリ、およびボリュームの情報の取得と設定
- ファイルデータの読み取りと書き込み

また、ファイルシステムフィルタードライバーは、キャッシュ、ロック、スパーズファイル、ディスククォータ、圧縮、セキュリティ、回復性、再解析ポイント、ボリュームマウントポイントなどのファイルシステム固有の機能をサポートしている必要があります。

ファイルシステムフィルタードライバーとデバイスドライバーの類似点と相違点の詳細については、次を参照してください。

- **ファイルシステムフィルタードライバーとデバイスドライバーの類似点**
- **ファイルシステムフィルタードライバーとデバイスドライバーの違い**

ファイル システム フィルター ドライバーとデバイス ドライバーの類似点

2020/04/24 • [Edit Online](#)

Microsoft Windows オペレーティングシステムのファイルシステムフィルタードライバーとデバイスドライバーは、次の点で似ています。

- 類似の構造体

デバイスドライバーと同様に、ファイルシステムフィルタードライバーには [Driverentry](#)、[Dispatch](#)、および [i/o 完了](#) ルーチンがあります。デバイスドライバーが呼び出したのと同じカーネルモードルーチンの多くを呼び出し、関連付けられているデバイス (つまり、ファイルシステムボリューム) の i/o 要求をフィルター処理します。

- 同様の機能

- ファイルシステムフィルタードライバーとデバイスドライバーは i/o システムの一部であるため、i/o [要求パケット](#)(irp) を受信して操作します。
- デバイスドライバーと同様に、ファイルシステムフィルタードライバーは独自の Irp を作成し、下位レベルのドライバーに送信することもできます。
- どちらの種類のドライバーも、さまざまなシステムイベントの通知を (コールバック関数を使用して) 登録できます。

- その他の類似点

- デバイスドライバーと同様に、ファイルシステムフィルタードライバーは、[I/o 制御コード](#)(ioctl) の概要を受け取ることができます。ファイルシステムフィルタードライバーは、[ファイルシステム制御コード](#)(FSCTLs) を受信して定義することもできます。
- デバイスドライバーと同様に、ファイルシステムフィルタードライバーは、システムの起動時に読み込まれるように構成することも、システムの起動プロセスが完了した後に読み込むように構成することもできます。

ファイル システム フィルター ドライバーとデバイス ドライバーの相違点

2020/04/24 • [Edit Online](#)

Microsoft Windows オペレーティングシステムのファイルシステムフィルタードライバーとデバイスドライバーは、次の点で異なります。

- 電源管理なし

ファイルシステムフィルタードライバーはデバイスドライバーではないため、ハードウェアデバイスを直接制御しないため、[IRP_MJ_の電源](#) 要求を受信しません。代わりに、電力 Irp はストレージデバイススタックに直接送信されます。ただし、まれに、ファイルシステムフィルタードライバーによって電源管理が妨げられることがあります。このため、ファイルシステムフィルタードライバーは、Driverentryルーチンで IRP_MJ_電源のディスパッチルーチンを登録する必要がありません。また、[poxxx](#)ルーチンを呼び出すことはできません。

- WDM なし

ファイルシステムフィルタードライバーを Windows Driver Model (WDM) ドライバーにすることはできません。Microsoft [Windows Driver Model](#)は、デバイスドライバーのみを対象としています。

- AddDevice または StartIo はありません

ファイルシステムフィルタードライバーはデバイスドライバーではないため、ハードウェアデバイスを直接制御しないので、[AddDevice](#)または[StartIo](#)ルーチンを指定しないでください。

- 作成されたさまざまなデバイスオブジェクト

ファイルシステムフィルタードライバーとデバイスドライバーは両方ともデバイスオブジェクトを作成しますが、作成するデバイスオブジェクトの数と種類は異なります。

デバイスドライバーは、デバイスを表す物理デバイスオブジェクトと機能デバイスオブジェクトを作成します。プラグアンドプレイ (PnP) マネージャーは、デバイスドライバーによって作成されたすべてのデバイスオブジェクトを含むグローバルデバイスツリーを構築し、管理します。ファイルシステムフィルタードライバーによって作成されたデバイスオブジェクトは、このデバイスツリーに含まれていません。

ファイルシステムフィルタードライバーは、物理的または機能しているデバイスオブジェクトを作成しません。代わりに、コントロールデバイスオブジェクトを作成し、デバイスオブジェクトをフィルター処理します。コントロールデバイスオブジェクトは、システムおよびユーザーモードアプリケーションに対するフィルタードライバーを表します。フィルターデバイスオブジェクトは、特定のファイルシステムまたはボリュームをフィルター処理する実際の作業を実行します。通常、ファイルシステムフィルタードライバーは、1つのコントロールデバイスオブジェクトと1つまたは複数のフィルターデバイスオブジェクトを作成します。

- その他の相違点

- ファイルシステムフィルタードライバーはデバイスドライバーではないため、[ダイレクトメモリアクセス \(DMA\)](#)は実行されません。
- デバイスフィルタードライバーは、ターゲットデバイスの関数ドライバーの上または下にアタッチできますが、ファイルシステムフィルタードライバーは、ターゲットファイルシステムドライバーの上にはアタッチできません。したがって、デバイスドライバーの用語では、ファイルシステムフィルタードライバーは上位フィルターに限らず、フィルターを下げることはできません。

フィルター マネージャーの概念

2020/04/24 • [Edit Online](#)

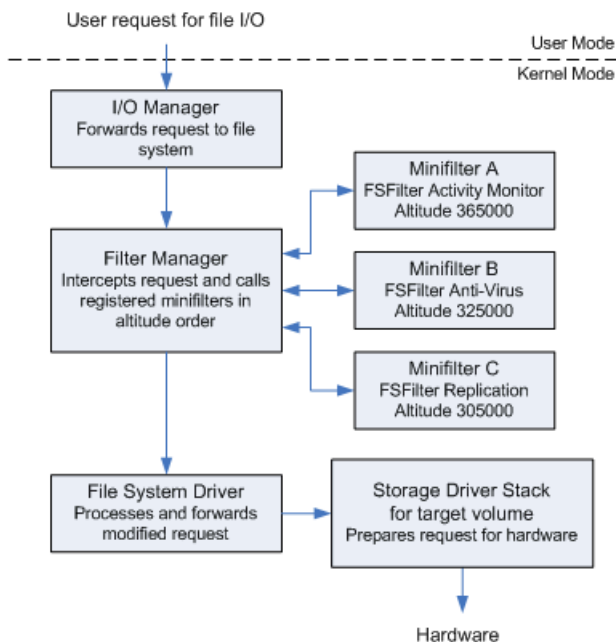
フィルターマネージャーは、従来のファイルシステムフィルターモデルに準拠するカーネルモードドライバーです。ファイルシステムフィルタードライバーで一般的に必要なとされる機能を実装し、公開します。この機能を利用することで、サードパーティの開発者は、従来のファイルシステムフィルタードライバーよりも簡単に開発できるミニフィルタードライバーを作成できます。これにより、高品質で堅牢なドライバーを生成しながら、開発プロセスが短縮されます。

フィルターマネージャーは Windows と共にインストールされますが、ミニフィルタードライバーが読み込まれた場合にのみアクティブになります。フィルターマネージャーは、ターゲットボリュームのファイルシステムスタックにアタッチします。ミニフィルタードライバーはファイルシステムスタックに間接的にアタッチします。フィルタマネージャーに登録すると、ミニフィルタードライバーによってフィルター処理される i/o 操作が選択されます。

ミニフィルタードライバーは、特定の順序でアタッチされます。添付ファイルの順序は、高度と呼ばれる一意の識別子によって決まります。特定のボリューム上の特定の高度にミニフィルタードライバーを添付することを、ミニフィルタードライバーのインスタンスといいます。

ミニフィルタードライバーの高度を使用すると、ミニフィルタードライバーのインスタンスは、他のミニフィルタードライバーインスタンスとは関係なく、常に適切な場所に読み込まれます。また、フィルターマネージャーがミニフィルタードライバーを呼び出して、i/o を処理する順序を決定します。階層は Microsoft によって割り当てられ、管理されます。

次の図は、フィルターマネージャーと3つのミニフィルタードライバーを使用した単純な i/o スタックを示しています。



ミニフィルタードライバーは、IRP ベースの i/o 操作だけでなく、高速 i/o およびファイルシステムフィルター (FSFilter) コールバック操作をフィルター処理できます。フィルター処理の対象として選択された i/o 操作ごとに、ミニ[操作コールバックルーチン](#)、postoperation コールバックルーチン、またはその両方を登録できます。i/o 操作を処理する場合、フィルターマネージャーは、その操作に対して登録されているミニフィルタードライバーごとに適切なコールバックルーチンを呼び出します。このコールバックルーチンが戻ると、フィルターマネージャーは、操作に対して登録された次のミニフィルタードライバーの適切なコールバックルーチンを呼び出します。

たとえば、上記の3つのすべてのミニフィルタードライバーが同じ i/o 操作に対して登録されている場合、フィルターマネージャーは、上位から下位 (A、B、C) の高度な順序で preoperation コールバックルーチンを呼び出し、その後の処理のために i/o 要求を次の下位のドライバーに転送します。フィルターマネージャーは、完了のために i/o 要求を受信すると、各ミニフィルタードライバーの postoperation コールバックルーチンを逆順 (C、B、A) の逆の順序で呼び出します。

レガシフィルタードライバーとの相互運用性を確保するために、フィルターマネージャーは、複数の場所にあるファイルシステムの i/o スタックにフィルターデバイスオブジェクトをアタッチできます。フィルターマネージャーの各デバイスオブジェクトは、フレームと呼ばれます。従来のフィルタードライバーの観点から見ると、各フィルターマネージャーフレームは、別のレガシフィルタードライバーにすぎません。

各フィルターマネージャーフレームは、高度な範囲を表します。フィルターマネージャーでは、既存のフレームを調整したり、新しいフレームを作成して、ミニフィルタードライバーが適切な場所に接続されるようにすることができます。

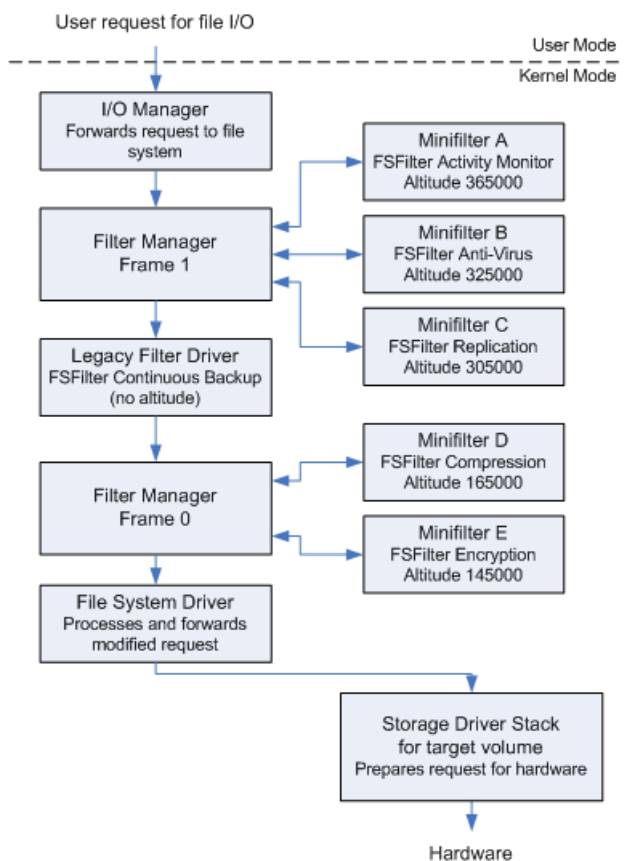
フィルターマネージャーは、接続されている2つのレガシフィルターの間にフィルターマネージャーフレームが既に存在しない限り、ミニフィルターをアタッチできません。ミニフィルターをレガシフィルターの上にアタッチすることを想定している場合は、2番目にアタッチされたレガシフィルターの存在に応じて、その下にアタッチできます。レガシフィルターの下に接続するためのミニフィルターは、レガシフィルターよりも上にアタッチすることができます。

IMPORTANT

レガシフィルターとミニフィルターの相互運用性を常に確認するか、レガシフィルターをミニフィルターに置き換えることを検討してください。詳細については、「[レガシフィルタードライバーを移植するためのガイドライン](#)」を参照してください。

ミニフィルタードライバーがアンロードされ、再読み込みされると、そのドライバーは、アンロードされた同じフレーム内の同じ高度に再読み込みされます。

次の図は、2つのフィルターマネージャーフレーム、ミニフィルタードライバーインスタンス、およびレガシフィルタードライバーを使用した単純な i/o スタックを示しています。



フィルター マネージャー モデルの利点

2020/04/24 • [Edit Online](#)

フィルターマネージャーモデルでは、既存のレガシフィルタードライバーモデルよりも次の利点があります。

- **DAX ボリュームに接続する機能。**レガシファイルシステムフィルタードライバーは、直接アクセス (DAX) ボリュームにアタッチできません。
- **フィルターの読み込み順序をより細かく制御できます。**レガシフィルタードライバーとは異なり、ミニパスドライバーはいつでも読み込み、その高度によって決定される適切な場所に接続できます。
- **システムの実行中にアンロードできます。**システムの実行中にアンロードできない従来のフィルタードライバーとは異なり、ミニフィルタードライバーはいつでもアンロードできます。また、必要に応じて、それ自体がアンロードされないようにすることができます。フィルタマネージャは、すべてのミニフィルタドライバ添付ファイルの安全な削除を同期し、ミニフィルタドライバがアンロードされた後に完了した操作を処理します。
- **必要な操作のみを処理する機能。**フィルターマネージャーでは、フィルター処理に使用する i/o 操作 (IRP ベース、高速 i/o、または FSFilter) の種類をミニフィルタードライバーで選択できるコールバックモデルが使用されます。ミニフィルタードライバーは、コールバックルーチンが登録されている i/o 要求のみを受信します。ミニフィルタードライバーは、一意の preoperation または postoperation コールバックルーチン、あるいはその両方を登録できます。また、ページング i/o や キャッシュ i/o など、特定の種類の操作を無視できます。
- **カーネルスタックをより効率的に使用します。**フィルターマネージャーは、使用するカーネルスタックの量を減らすように最適化されており、コールバックモデルにより、スタックへのミニフィルタードライバーの影響が大幅に減少します。フィルターマネージャーは、スタック内の下位のドライバーによってのみ見られるフィルターによる i/o をサポートすることで、再帰 i/o の影響を軽減します。
- **より少ない冗長コード。**フィルターマネージャーを使用すると、複数のミニフィルタードライバーで使用する名前の生成とキャッシュファイル名のインフラストラクチャを提供するなど、さまざまな方法でミニフィルタードライバーに必要なコードの量を減らすことができます。フィルターマネージャーはボリュームにアタッチし、ボリュームが使用可能になったときにミニフィルタードライバーに通知します。フィルターマネージャーは、マルチプロセッサシステムをサポートするように最適化されています。これにより、ロックが効率的になり、エラーが発生しにくくなります。
- **複雑さが軽減されます。**フィルターマネージャーは、名前付け、コンテキスト管理、ユーザーモードとカーネルモード間の通信、ファイルシステム間のマスクの違いなど、一般的な機能のサポートルーチンを提供することで、i/o 要求のフィルター処理を簡略化します。フィルターマネージャーは、保留中の lrp、列挙、ファイルシステムスタックへのアタッチなど、ミニフィルタードライバーに代わって特定のタスクを処理します。
- **新しい操作の追加が容易になりました。**ミニフィルタードライバーは処理する i/o 操作に対してのみ登録されるため、新しい操作のサポートをフィルターマネージャーに追加して、既存のミニフィルタードライバーを中断することはできません。
- **複数のプラットフォームのサポートの強化。**フィルタマネージャーをサポートする任意のバージョンの Windows で、ミニフィルタードライバーを実行できます。実行時に使用できない i/o 操作をミニフィルタードライバーが登録した場合、フィルターマネージャーは、その操作に対してミニフィルタードライバーを呼び出しません。ミニフィルタードライバーは、関数が使用可能かどうかをプログラムによって判断できます。また、フィルターマネージャー構造は拡張可能であるように設計されています。
- **ユーザーモードアプリケーションのサポートが強化されています。**フィルターマネージャーは、ユーザーモードのサービスと、ミニフィルタードライバーを操作するプログラムに共通の機能を提供します。フィルターマネージャーのユーザーモードドライブラリである Filterlib.dll を使用すると、ユーザーモードサービスまたはコントロールプログラムとミニフィルタードライバーの間の通信が可能になります。Filterlib.dll には、管理ツールのインターフェイスも用意されています。

ロードとアンロード

2020/04/24 • [Edit Online](#)

読み込みとアンロードについて

ミニフィルタードライバーは、システムの実行中にいつでも読み込むことができます。ミニフィルタードライバーの **INF ファイル** で、SERVICE_BOOT_START、SERVICE_SYSTEM_START、または SERVICE_AUTO_START のドライバーの開始の種類が指定されている場合、従来のフィルタードライバーとの相互運用性をサポートするために、ファイルシステムフィルタードライバーの既存の読み込み順序グループ定義に従ってミニフィルタードライバーが読み込まれます。システムの実行中は、サービス開始要求 (sc start、net start、または service Api) を介して、または明示的な読み込み要求 (fltmc load、**Fltloadfilter**、または **filterload**) を介してミニフィルタードライバーを読み込むことができます。

ミニフィルタードライバーの **Driverentry** ルーチンはミニフィルタードライバーが読み込まれるときに呼び出されます。そのため、ミニフィルタードライバーは、ミニフィルタードライバーのすべてのインスタンスに適用される初期化を実行できます。その **Driverentry** ルーチンでは、ミニフィルタードライバーは **fltregisterfilter** を呼び出して、フィルターマネージャーと **fltregisterfilter** にコールバックルーチンを登録します。これにより、フィルターマネージャーに対して、ミニフィルタードライバーを開始する準備ができたことが通知されます。ボリュームにアタッチし、i/o 要求をフィルター処理しています。

ミニフィルタードライバーインスタンスは、ミニフィルタードライバーのインストールに使用される **INF ファイル** で定義されています。ミニフィルタードライバーの INF ファイルでは、既定のインスタンスを定義する必要があります。また、追加のインスタンスを定義することもできます。これらの定義は、すべてのボリュームに適用されます。各インスタンス定義には、インスタンス名、その高度、インスタンスを自動的に接続できるかどうかを示すフラグ、手動で、またはその両方を含めることができます。既定のインスタンスは、フィルターマネージャーがミニフィルタードライバーのマウントおよびインスタンスセットアップコールバックルーチンを正しい順序で呼び出すように、ミニフィルタードライバーの順序を設定するために使用されます。既定のインスタンスは、呼び出し元がインスタンス名を指定していない場合に、明示的な添付ファイル要求でも使用されます。

フィルターマネージャーは、ボリュームがマウントされた後の最初の作成操作で **Instancesetupcallback** ルーチンを呼び出すことによって、使用可能なボリュームについてミニフィルタードライバーに自動的に通知します。これは、フィルターマネージャーがシステムの起動時に既存のボリュームを列挙するときに、**Fltstartfiltering** が戻る前に発生することがあります。また、実行時に、ボリュームがマウントされたとき、または明示的な添付ファイルの要求 (fltmc attach、**Fltattachvolume**、または **filterattach**) の結果として発生することもあります。

ミニフィルタードライバーインスタンスは、ミニフィルタードライバーがアンロードされたとき、インスタンスがアタッチされているボリュームがマウント解除されたとき、または明示的なデタッチ要求 (fltmc detach、**FltDetachVolume**、または **filterdetach**) の結果として破棄されます。ミニフィルタードライバーによって **InstanceQueryTeardownCallback** ルーチンが登録されると、**Filterdetach** または **FltDetachVolume** を呼び出すことによって明示的なデタッチ要求が失敗する可能性があります。破棄は次のように進行します。

- ミニフィルタードライバーによって **InstanceTeardownStartCallback** コールバックルーチンが登録されている場合、フィルターマネージャーは破棄プロセスの開始時にそれを呼び出します。このルーチンでは、ミニフィルタードライバーがすべての保留中の操作を完了するか、ミニフィルタードライバーによって生成された i/o 要求などの他の処理を完了し、新しい作業項目のキューを停止します。
- インスタンスの破棄中、現在実行中の preoperation または postoperation コールバックルーチンは通常の処理を続行し、postoperation コールバックを待機しているすべての i/o 要求は "ドレイン" またはキャンセルされ、ミニフィルタードライバーは、完了するまで通常の処理を続行します。
- ミニフィルタードライバーによって **InstanceTeardownCompleteCallback** ルーチンが登録されている場合、未処理の i/o 操作がすべて完了した後に、フィルターマネージャーがこのルーチンを呼び出します。このルーチンでは、ミニフィルタードライバーによって、開いているファイルがすべて閉じられます。
- インスタンスへの未処理の参照がすべて解放されると、フィルターマネージャーは残りのコンテキストを削除し、インスタンスは

完全に破棄されます。

システムの実行中は、サービス停止要求 (sc stop、net stop、または service Api) を介して、または明示的なアンロード要求 (fltmc unload、[Fltunloadfilter](#)、または[filterunload](#)) を介してミニフィルタードライバーをアンロードできます。

ミニフィルタードライバーの[FilterUnloadCallback](#)ルーチンは、ミニフィルタードライバーがアンロードされるときに呼び出されます。このルーチンは、開いている通信サーバーのポートを閉じ、[Fltunregisterfilter](#)を呼び出して、必要なクリーンアップを実行します。このルーチンの登録は任意です。ただし、ミニフィルタードライバーで[FilterUnloadCallback](#)ルーチンが登録されていない場合は、ミニフィルタードライバーをアンロードできません。このルーチンの詳細については、「[FilterUnloadCallback ルーチンの記述](#)」を参照してください。

ミニフィルタードライバーを読み込んでアンロードするためのフィルターマネージャールーチン

フィルターマネージャーには、明示的な読み込み要求とアンロード要求に対する次のサポートルーチンが用意されています。これらのルーチンは、ユーザーモードまたはカーネルモードから発行できます。

- [FilterLoad](#)
- [FilterUnload](#)
- [FltLoadFilter](#)
- [FltUnloadFilter](#)

インスタンスのセットアップと破棄のためにコールバックルーチンを登録および登録解除するには、次のルーチンを使用します。

- [FltRegisterFilter](#)
- [FltStartFiltering](#)
- [FltUnregisterFilter](#)

インスタンスのセットアップ、破棄、アンロードを行うためのミニフィルタードライバーのコールバックルーチン

次のミニフィルタードライバーコールバックルーチンは、[Fltregisterfilter](#)にパラメーターとして渡される[FLT_REGISTRATION](#)構造体のメンバーとして格納されます。

*****	*****
InstanceSetupCallback	PFLT_INSTANCE_SETUP_CALLBACK
InstanceQueryTeardownCallback	PFLT_INSTANCE_QUERY_TEARDOWN_CALLBACK
InstanceTeardownStartCallback	PFLT_INSTANCE_TEARDOWN_CALLBACK
InstanceTeardownCompleteCallback	PFLT_INSTANCE_TEARDOWN_CALLBACK
FilterUnloadCallback	PFLT_FILTER_UNLOAD_CALLBACK

I/O 操作の処理

2020/04/24 • [Edit Online](#)

フィルターマネージャーを使用すると、ミニフィルタードライバーの i/o 操作の処理が簡単になります。レガシフィルタードライバーは、すべての i/o 要求を次の下位のドライバーに正しく渡す必要があります。また、レガシフィルタードライバーが要求に関連する処理を行うかどうかに関係なく、保留中の要求、同期、および i/o の完了を正しく処理する必要があります。は、処理する必要がある i/o 操作に対してのみ登録します。

特定の i/o 操作では、フィルターマネージャーは、その操作に対して**事前操作コールバック**ルーチンが登録されているミニフィルタードライバーのみを呼び出します。フィルターマネージャーは、ミニフィルタードライバーに代わって特定の IRP メンテナンスタスクを処理します。たとえば、パラメーターを次のスタック位置にコピーし、IRP Pendingreturned れたフラグを伝達することができます。

事前操作コールバックルーチンでは、ミニ操作コールバックルーチンから適切な値を返すことによって、i/o 操作に必要な処理がすべてミニフィルタードライバーによって行われます。たとえば、完了ルーチンを使用せずに、IRP を次の下位のドライバーに転送する場合、ミニフィルタードライバーは、FLT_PREOP_SUCCESS__コールバックなしで返されます。完了ルーチン (i/o 操作のミニフィルタードライバーの postoperation コールバックルーチン) で同じ操作を実行するには、ミニフィルタードライバーが__コールバックで FLT_PREOP_SUCCESS_を返します。

事前操作コールバックルーチンでは、**FltQueueDeferredIoWorkItem**を呼び出すことによって、必要に応じてミニフィルタードライバーがワーカースレッドに操作をキューできます。この操作を実行すると、ミニ操作コールバックルーチンから保留中の FLT_PREOP_PENDING が返され、i/o 操作が保留中であることが示されます。また、ミニフィルタードライバーは、要求の処理を完了または再開します。処理を再開するには、ミニフィルタードライバーがワーカースレッドから**Fltcompletependedpreoperation**を呼び出します。

ミニフィルタードライバーが、処理対象の未処理の i/o 操作のインスタンスごとのキャンセルセーフキューを保持する必要がある場合、そのようなキューを設定するには、*Instancesetupcallback*ルーチンで**FltCbdqInitialize**を呼び出し、必要に応じて preoperation コールバックルーチンで**FltCbdqInsertIo**を呼び出して、i/o 操作をキューに挿入します。

フィルターマネージャーは、低フィルタドライバー (レガシフィルターおよびミニフィルタードライバー) が完了処理を完了したときに、ミニフィルタードライバーの i/o 操作に対してミニ**操作コールバック**ルーチンを呼び出します。

Postoperation コールバックルーチンでは、ミニフィルタードライバーは**Fltdocompletion Processingwhensafe**を呼び出して、完了処理が安全な IRQL で実行されるようにすることができます。または、**FltQueueDeferredIoWorkItem**を呼び出すことによって、必要に応じて、操作の完了処理をワーカースレッドにキューすることができます。この操作を実行すると、ミニフィルタードライバーは、postoperation コールバックルーチンから必要な_処理_より多くの_を_返します。これにより、i/o 操作のフィルターマネージャーの完了処理が中断されます。完了処理を再開するために、ミニフィルタードライバーはワーカースレッドから**Fltcompletependedpostoperation**を呼び出します。

フィルターマネージャーは、i/o 操作ではなく、ミニフィルタードライバーまたはミニフィルタードライバーインスタンスに関連付けられた作業項目である "汎用" 作業項目のキューのサポートを提供します。ミニフィルタードライバーは、**Fltqueuegenericworkitem**を呼び出して、作業項目をシステム作業キューに挿入できます。このルーチンは、**Exqueueworkitem**などのルーチンに似ています。たとえば、**FltAllocateGenericWorkItem**を呼び出すことによって割り当てられた作業項目は再利用できます。ただし、フィルターマネージャーでは、未処理の作業項目がまだ処理されている間はミニフィルタードライバーまたはミニフィルタードライバーインスタンスをアンロードできないため、Fltqueuegenericworkitemはミニフィルタードライバーを使用の方が安全です。

フィルターマネージャーでは、便宜的ロック (oplock) 操作もサポートされています。Oplock 操作の場合、ミニフィルタードライバーは**Fltinitializeoplock**および**FltOplockFsctrl**としてこのようなフィルターマネージャールーチンを使用できます。これは、FsRtlInitializeOplockおよびFsRtlOplockFsctrlルーチンと同等です。ファイルシステムおよびレガシフィルタードライバーによって使用されます。

I/O 操作を処理するためのフィルターマネージャールーチン

フィルターマネージャーには、**preoperation**および**postoperation**コールバックルーチンでの保留中の i/o に対する次のサポート

ルーチンが用意されています。

[FltCompletePendedPostOperation](#)

[FltCompletePendedPreOperation](#)

[Fltdo補完 Processingwhensafe](#)

次のルーチンは、事前操作および postoperation コールバックルーチンで作業項目をキューに置いたときに使用されます。

[FltAllocateDeferredIoWorkItem](#)

[FltAllocateGenericWorkItem](#)

[FltFreeDeferredIoWorkItem](#)

[FltFreeGenericWorkItem](#)

[FltQueueDeferredIoWorkItem](#)

[FltQueueGenericWorkItem](#)

次のルーチンは、キャンセルセーフキューのサポートを提供します。

[FltCbdqDisable](#)

[FltCbdqEnable](#)

[FltCbdqInitialize](#)

[FltCbdqInsertIo](#)

[FltCbdqRemovelIo](#)

[FltCbdqRemoveNextIo](#)

次のルーチンでは、oplock のサポートが提供されます。

[FltCheckOplock](#)

[FltCurrentBatchOplock](#)

[FltInitializeOplock](#)

[FltOplockFsctrl](#)

[FltOplockIsFastIoPossible](#)

[FltUninitializeOplock](#)

I/O 操作を処理するためのミニフィルタードライバーのコールバックルーチン

次のコールバックルーチンは、ミニフィルタードライバーが処理する i/o 操作の種類ごとに、[FLT_操作_登録](#) 構造体に格納されます。

XXXXXXXXXX	XXXXXXXXXXXX
<i>PostOperation</i>	PFLT_POST_OPERATION_CALLBACK
<i>PreOperation</i>	PFLT_PRE_OPERATION_CALLBACK

パラメーターの修正

2020/04/24 • [Edit Online](#)

ミニフィルタードライバーでは、ターゲットインスタンス、ターゲットファイルオブジェクト、バッファアドレスやメモリ記述子リスト (MDL) アドレスなどの操作固有のパラメーターなど、i/o 操作に関連付けられている特定のパラメーターを変更できます。ミニフィルタードライバーは、通常、i/o 要求に対して事前操作コールバックのパラメーターを変更します。ミニフィルタードライバーでパラメーターが変更された場合、パラメーターが変更されたことをフィルターマネージャーに通知するには、[FltSetCallbackDataDirty](#) を呼び出す必要があります。また、preoperation コールバックから渡されたコンテキストの変更を記録して、ポスト操作コールバックで使用するようにする必要があります。

ミニフィルタードライバーは、preoperation コールバックで操作を完了したとき、または postoperation コールバックで操作が失敗したときに、操作の i/o 状態を変更することができます (状態_成功をエラー状態に変更するなど)。この場合、[FltSetCallbackDataDirty](#) を呼び出す必要はありません。

パラメーターの変更の詳細については、「[I/O 操作のパラメーターの変更](#)」を参照してください。

ミニフィルタードライバーは、i/o 要求のバッファフィールドを独自のバッファに置き換えることによって、バッファのスワップを行うことができます。このようなミニフィルタードライバーは、i/o 要求の MDL およびバッファフィールドの同期を維持する役割を担います。フィルターマネージャーは、バッファがシステムバッファであるかどうかを示すために、[FLT_callback_データ](#) 構造で、[FLTFL_CALLBACK_DATA_SYSTEM_BUFFER](#) フラグを設定します。その場合、ミニフィルタードライバーは、非ページブルから交換バッファを割り当て、MDL フィールドを NULL に設定する必要があります。それ以外の場合は、ページングされたプールまたは非ページブルからバッファを割り当てることができ、ミニフィルタードライバーでは常に MDL を作成して設定する必要があります。(高速な i/o 操作の場合は、ページングされたプールまたは非ページブルから新しいバッファを割り当て、MDL を NULL にする必要があります)。ミニフィルタードライバーは、置き換えられるバッファまたは MDL を解放しないようにする必要があります。また、コールバックデータ構造に正常に挿入された MDL を解放することはできません (フィルターマネージャーによって、ミニフィルタードライバーの代わりに MDL が解放されます)。MDL またはバッファに変更を加えた後、ミニフィルタードライバーは [FltSetCallbackDataDirty](#) を呼び出す必要があります。

ミニフィルタードライバーは、バッファを交換するすべての操作に対して postoperation コールバックを登録する必要があります。このコールバックルーチンでは、ミニフィルタードライバーは、割り当てられたバッファを解放する必要があります。フィルターマネージャーは、ミニフィルタードライバーが [FltRetainSwappedBufferMdlAddress](#) を呼び出す場合を除き、MDL を解放します。この場合、ミニフィルタードライバーによって、MDL が解放されます。ミニフィルタードライバーは、[FltGetSwappedBufferMdlAddress](#) を呼び出して、preoperation コールバックでバッファセットの MDL を取得できます。

バッファをスワップした操作中にミニフィルタードライバーがアンロードされた場合、操作を "ドレイン" することはできません。代わりに、操作は取り消され、フィルターマネージャーは、ミニフィルタードライバーをアンロードする前に操作が完了するのを待機します。

バッファを交換するミニフィルタードライバーの例については、[SwapBuffers サンプル](#)を参照してください。

パラメーターを変更するためのフィルターマネージャールーチン

フィルターマネージャーには、preoperation および postoperation コールバックルーチンで i/o 操作パラメーターを変更するための次のサポートルーチンが用意されています。

[Fltclearの Datadirty](#)

[FltIsCallbackDataDirty](#)

[FltSetCallbackDataDirty](#)

次のルーチンでは、バッファのスワップがサポートされています。

[FltGetSwappedBufferMdlAddress](#)

[FltRetainSwappedBufferMdlAddress](#)

ユーザー バッファへのアクセス

2020/04/24 • [Edit Online](#)

バッファやメモリ記述子リスト (MDLs) など、特定の i/o 操作に固有のすべてのパラメーターは、**FLT_parameters** 共用体で定義されています。この共用体は、i/o 操作を表す **FLT_CALLBACK_データ** 構造体の `lop` メンバーを介してアクセスされる、**FLT_IO_パラメーター_ブロック** 構造体に含まれています。フィルターマネージャーとミニフィルタードライバーはどちらも、**FLT_コールバック_データ** 構造体を使用して、i/o 操作を開始および処理します。

また、**FLT_PARAMETERS** 共用体には、その操作に使用されるバッファリングメソッド (バッファ、ダイレクト i/o、バッファなしまたは直接 i/o) に固有の、IRP ベースの操作のパラメーター定義も含まれています。また、非 IRP ベースの操作 (fast i/o および `FsFilter` コールバックルーチン) のパラメーター定義も含まれています。

ミニフィルタードライバーは、**FltDecodeParameters** を呼び出して、i/o 操作の MDL アドレス、バッファポインター、バッファ長、および必要なアクセスパラメーターへのポインターを取得できます。これにより、これらのパラメーターを複数の i/o 操作にわたってアクセスするヘルパールーチンでこれらのパラメーターの位置を検索する `switch` ステートメントを使用して、ミニフィルタードライバーが保存されます。

ユーザーバッファを含む i/o 操作を処理する場合、ミニフィルタードライバーは常に、使用可能な場合は MDL を使用する必要があります。その場合、ミニフィルタードライバーは **MmGetSystemAddressForMdlSafe** を呼び出して、MDL のシステムアドレスを取得し、システムアドレスを使用してユーザーバッファにアクセスします。

バッファアドレスだけが使用可能な場合、ミニフィルタードライバーは、`try/except` ブロック内のバッファにアクセスする試行を常に囲む必要があります。ミニフィルタードライバーが、同期されていない `postoperation` コールバックルーチン内のバッファにアクセスする必要がある場合、または i/o 操作がワーカースレッドにポストされた場合は、**FltLockUserBuffer** を呼び出すことによって、ミニフィルタードライバーでもユーザーバッファをロックする必要があります。この関数は、i/o 操作の種類に基づいて、ロックされたバッファに適用する適切なアクセス方法を決定し、ロックされたページを指す MDL を作成します。

ユーザー バッファにアクセスするためのフィルターマネージャールーチン

フィルターマネージャーには、`preoperation` および `postoperation` コールバックルーチンでユーザーバッファにアクセスするための次のサポートルーチンが用意されています。

FltDecodeParameters

FltLockUserBuffer

ファイル名の管理

2020/04/24 • [Edit Online](#)

フィルターマネージャーを使用すると、レガシフィルタードライバーがファイル名を取得して管理するために必要な作業の多くが不要になります。名前が要求されると、フィルターマネージャーは、現在の操作に対して適切な形式 (正規化された名前、開かれた名前、または短い名前) で、参照カウント構造内の名前を提供します。

ミニパスのドライバーは、[FltGetDestinationFileNameInformation](#) を呼び出して、名前を変更するか、NTFS ハードリンクを作成するファイルまたはディレクトリの完全な宛先パス名を作成できます。この名前は、正規化またはオープンファイル形式で返すことができます。

フィルターマネージャーでは、ファイル名のトンネリングによって無効になる正規化されたファイル名情報を取得するため [FltGetTunneledName](#) ルーチンも用意されています。

パフォーマンスを向上させるために、フィルターマネージャーは、システム内のすべてのミニフィルタードライバー間で共有される名前をキャッシュに配置します (可能な場合)。ミニフィルタードライバーは、キャッシュまたはファイルシステムのいずれか、またはその両方に対してクエリを実行できます。

名前空間を変更するミニフィルタードライバーでは、コールバックルーチンを登録して名前のクエリ操作 (上位ミニフィルタードライバーによる名前の生成または正規化要求など) をインターセプトすることによって、名前プロバイダーのフィルターマネージャーのサポートを利用できます。

名前管理のフィルターマネージャールーチン

フィルターマネージャーでは、次の名前管理ルーチンがサポートされています。

[FltGetDestinationFileNameInformation](#)

[FltGetFileNameInformation](#)

[FltGetFileNameInformationUnsafe](#)

[FltGetTunneledName](#)

[FltParseFileNameInformation](#)

[FltReleaseFileNameInformation](#)

名前管理用のミニフィルタードライバーコールバックルーチン

次のコールバックルーチンは、[FLT_登録](#) 構造体に格納されています。この構造は、名前空間を変更するミニフィルタードライバーの [Fltregisterfilter](#) にパラメーターとして渡されます。

*****	*****
<i>GenerateFileNameCallback</i>	PFLT_GENERATE_FILE_NAME
<i>NormalizeContextCleanupCallback</i>	PFLT_NORMALIZE_CONTEXT_CLEANUP
<i>NormalizeNameComponentCallback</i>	PFLT_NORMALIZE_NAME_COMPONENT

コンテキストの管理

2020/04/24 • [Edit Online](#)

フィルターマネージャーを使用すると、ミニフィルタードライバーを使用して、i/o 操作間で状態を保持するためにコンテキストをオブジェクトに関連付けることができます。コンテキストを持つことのできるオブジェクトには、ボリューム、インスタンス、ストリーム、およびストリームハンドルがあります。

サードパーティのファイルシステムでは、**FSRTL_ADVANCED_fcb_header** 構造体 (**FSRTL_COMMON_fcb_header** 構造体ではなく) を使用して、ストリームおよびストリームハンドルコンテキストで適切に動作する必要があります。

コンテキストは、ページングされたプールまたは非ページプールから割り当てることができます。ただし、ボリュームコンテキストは、非ページプールから割り当てする必要があります。

すべての未処理の参照が解放されると、コンテキストは自動的に解放されます。ミニフィルタードライバーによってコンテキストクリーンアップコールバックルーチンが定義されている場合、コンテキストが解放される前に、フィルターマネージャーがルーチンを呼び出します。

フィルターマネージャーは、関連付けられたオブジェクトが削除されたとき、インスタンスがデタッチされたとき、およびミニフィルタードライバーがアンロードされたときに、コンテキストの削除を行います。

ミニフィルタードライバーが1つのボリュームにつき1つのインスタンスのみをサポートしている場合は、パフォーマンスを向上させるために、ボリュームコンテキストではなくインスタンスコンテキストを使用します。また、ストリームまたはストリームハンドルコンテキスト内にミニフィルタードライバーインスタンスコンテキストへのポインターを格納することで、パフォーマンスを向上させることもできます。

コンテキストは、ページングファイルまたは次の操作中にはサポートされません。

- 作成要求の事前操作処理
- クローズ要求の postoperation 処理
- IRP_MJ_ネットワーク_クエリ_オープン要求の処理

コンテキストを使用するミニフィルタードライバーの例については、CTX サンプルを参照してください。

コンテキスト管理のフィルターマネージャールーチン

フィルターマネージャーには、コンテキストを作成、登録、および設定するための次のサポートルーチンが用意されています。

[FltAllocateContext](#)

[FltRegisterFilter](#)

[FltSetInstanceContext](#)

[FltSetStreamContext](#)

[FltSetStreamHandleContext](#)

[FltSetVolumeContext](#)

コンテキストサポートをクエリするために、次のルーチンが用意されています。

[FltSupportsStreamContexts](#)

[FltSupportsStreamHandleContexts](#)

コンテキストを取得および参照するために、次のルーチンが用意されています。

[FltGetContexts](#)

[FltGetInstanceContext](#)

[FltGetStreamContext](#)

[FltGetStreamHandleContext](#)

[FltGetVolumeContext](#)

[FltReferenceContext](#)

コンテキストの解放と削除には、次のルーチンが用意されています。

[FltDeleteContext](#)

[FltDeleteInstanceContext](#)

[FltDeleteStreamContext](#)

[FltDeleteStreamHandleContext](#)

[FltDeleteVolumeContext](#)

[FltReleaseContext](#)

[FltReleaseContexts](#)

コンテキスト管理用のミニフィルタードライバーコールバックルーチン

次のコールバックルーチンは、コンテキストを管理するミニフィルタードライバーの[Fltregisterfilter](#)にパラメーターとして渡される[FLT_登録](#)構造体に格納されます。

*****	*****
<i>ContextAllocateCallback</i>	PFLT_CONTEXT_ALLOCATE_CALLBACK
<i>ContextCleanupCallback</i>	PFLT_CONTEXT_CLEANUP_CALLBACK
<i>ContextFreeCallback</i>	PFLT_CONTEXT_FREE_CALLBACK

ミニフィルター ドライバーで生成された I/O 要求

2020/04/24 • [Edit Online](#)

ミニフィルタードライバーは、現在のボリュームまたは別のボリューム上にあるミニフィルタードライバー独自のインスタンスから、IRP ベースの i/o 要求を生成して送信することができます。生成された i/o は、指定したインスタンスの下、およびファイルシステムによって接続された、ミニフィルタードライバーインスタンスとレガシフィルタードライバーによってのみ表示されます。これにより、レガシフィルタードライバーモデルの再帰 i/o に関連する多くの問題が解決されます。レガシフィルタードライバーによって生成される i/o 要求は、最上位のドライバーから順にファイルシステムスタック全体を通過する必要があります。

フィルターマネージャーは、未処理のすべての i/o 操作が完了するまでミニフィルタードライバーをアンロードしません。

ミニフィルタードライバーによって生成される i/o 要求のフィルターマネージャールーチン

フィルターマネージャーには、ファイルの作成、開く、読み取り、および書き込みを行うための次のサポートルーチンが用意されています。

[FltClose](#)

[FltCreateFile](#)

[FltCreateFileEx](#)

[FltReadFile](#)

[FltWriteFile](#)

再解析ポイントの設定と削除には、次のサポートルーチンが用意されています。

[FltTagFile](#)

[FltUntagFile](#)

I/O 要求を生成するために、次のサポートルーチンが用意されています。

[FltAllocateCallbackData](#)

[FltFreeCallbackData](#)

[FltPerformAsynchronousIo](#)

[FltPerformSynchronousIo](#)

[FltReuseCallbackData](#)

ファイルオープン要求をキャンセルし、i/o 要求を再発行するために、次のサポートルーチンが用意されています。

[FltCancelFileOpen](#)

[FltReissueSynchronousIo](#)

フィルターマネージャーには、次の汎用ルーチンも用意されています。

[FltDeviceIoControlFile](#)

[FltFlushBuffers](#)

[FltFsControlFile](#)

[FltQueryInformationFile](#)

FltQuerySecurityObject

FltQueryVolumeInformationFile

FltSetInformationFile

FltSetSecurityObject

ユーザー モードとカーネル モード間のコミュニケーション

2020/04/24 • [Edit Online](#)

フィルタマネージャーは、通信ポートを使用したユーザーモードとカーネルモード間の通信をサポートしています。ミニフィルタードライバーは、通信ポートオブジェクトに適用するセキュリティ記述子を指定することによって、ポートのセキュリティを制御します。通信ポートを介した通信はバッファリングされないため、高速で効率的です。ユーザーモードのアプリケーションまたはサービスは、ミニ転送ドライバーからのメッセージにตอบสนองして双方向通信を行うことができます。

ミニフィルタードライバーは、通信サーバーのポートを作成するときに、そのポートでの着信接続のリッスンを開始します。ユーザーモードの呼び出し元がポートに接続しようとする、フィルタマネージャーは、新しく作成された接続を処理するハンドルを使用して、ミニフィルタードライバーの [ConnectNotifyCallback](#) ルーチンを呼び出します。フィルタマネージャーは制御を取り戻すと、ユーザーモードの呼び出し元を、接続に対するユーザーモードの呼び出し元のエンドポイントを表す別のファイルハンドルに渡します。このハンドルを使用すると、i/o 完了ポートをリスナーポートに関連付けることができます。

接続は、ユーザーモードの呼び出し元が、ポートのセキュリティ記述子で指定されている十分なアクセス権を持っている場合にのみ受け入れられます。ポートに接続するたびに、独自のメッセージキューとプライベートエンドポイントが取得されます。

いずれかのエンドポイント (カーネルまたはユーザー) を閉じると、その接続が終了します。ユーザーモードの呼び出し元がエンドポイントへのハンドルを閉じると、フィルタマネージャーはミニフィルタードライバーの接続 [解除ルーチン](#) を呼び出し、ミニフィルタードライバーが接続のハンドルを閉じることができるようにします。

通信サーバーのポートを閉じると、新しい接続が防止されますが、既存の接続は終了しません。フィルタマネージャーは、ミニフィルタードライバがアンロードされたときに既存の接続を終了します。

ユーザーモードとカーネルモード間の通信用フィルタマネージャールーチン

フィルタマネージャーには、ユーザーモードアプリケーションと通信するためのカーネルモードミニフィルタードライバーの次のサポートルーチンが用意されています。

[FltCloseClientPort](#)

[FltCloseCommunicationPort](#)

[FltCreateCommunicationPort](#)

[FltSendMessage](#)

ユーザーモードアプリケーションがミニフィルタードライバーと通信するには、次のサポートルーチンが用意されています。

[FilterConnectCommunicationPort](#)

[FilterGetMessage](#)

[FilterReplyMessage](#)

[FilterSendMessage](#)

ユーザーモードとカーネルモード間の通信用のミニフィルタードライバーコールバックルーチン

次のミニフィルタードライバーコールバックルーチンは、パラメーターとして [FltCreateCommunicationPort](#) に渡されます。

*****	*****
<i>ConnectNotifyCallback</i>	PFLT_CONNECT_NOTIFY

名称	名称
切断 Notifycallback	PFLT_DISCONNECT_NOTIFY
MessageNotifyCallback	PFLT_MESSAGE_NOTIFY

ユーザーモード ライブラリ

2020/04/24 • [Edit Online](#)

フィルターマネージャーのユーザーモードインターフェイスは、フィルタードライバーを含む製品に共通の機能を提供します。ユーザーモードライブラリは *Fltlib.dll* です。アプリケーションには、ヘッダーファイル *Fltuser.h* と *Fltuser* ストラクチャー *.h* が含まれています。また、*fltuser* にリンクしています。

これらのユーザーモードインターフェイスを使用すると、ミニフィルタードライバーとユーザーモードサービスまたはコントロールプログラムとフィルタードライバーの間の通信を全般的に制御できます。ユーザーモードインターフェイスには、フィルター、ボリューム、およびインスタンスの列挙を可能にする管理ツール用のインターフェイスも用意されています。

ミニフィルターの場合、ユーザーモード通信 Api は管理者特権を必要としません。代わりに、ミニフィルターでは、ポートで定義されている **ACL** を使用して必要な特権を定義します。

フィルターマネージャーのユーザーモードライブラリルーチン

フィルターマネージャーには、ミニフィルタードライバーの読み込みとアンロードに使用するユーザーモードアプリケーションの次のサポートルーチンが用意されています。

FilterLoad

FilterUnload

ミニフィルタードライバーとインスタンスハンドルの作成と終了には、次のサポートルーチンが用意されています。

FilterClose

FilterCreate

FilterInstanceClose

FilterInstanceCreate

ミニフィルタードライバーインスタンスのアタッチとデタッチには、次のサポートルーチンが用意されています。

FilterAttach

FilterAttachAtAltitude

FilterDetach

フィルター、ボリューム、およびインスタンスを列挙するために、次のサポートルーチンが用意されています。

FilterFindFirst

FilterFindNext

FilterInstanceFindFirst

FilterInstanceFindNext

FilterVolumeFindFirst

FilterVolumeFindNext

FilterVolumeInstanceFindFirst

FilterVolumeInstanceFindNext

情報を照会するために、次のサポートルーチンが用意されています。

[FilterGetDosName](#)

[FilterGetInformation](#)

[FilterInstanceGetInformation](#)

ユーザー操作によって開始される通信には、次のサポートルーチンが用意されています。

[FilterConnectCommunicationPort](#)

[FilterSendMessage](#)

ミニフィルタードライバーによって開始される通信に応答するために、次のサポートルーチンが用意されています。

[FilterGetMessage](#)

[FilterReplyMessage](#)

フィルター マネージャー操作の制御

2020/04/24 • [Edit Online](#)

Windows Vista は、REG によって制御されるより前のバージョンの Windows 上のフィルター マネージャーの操作 `_DWORD AttachWhenLoaded` レジストリ値が次のキーに格納されます。

```
HKLM\System\CurrentControlSet\Services\FltMgr
```

ときに `AttachWhenLoaded` 設定が 0 の場合にフィルター マネージャー アタッチできないのすべてのボリューム フィルター マネージャーとミニフィルター ドライバーに登録されるまで。ときに `AttachWhenLoaded` フィルター マネージャー接続のすべてのボリュームのブート時に、1 に設定されます。

既定値 `AttachWhenLoaded` は Windows XP Service Pack 2 (SP2) 以降では 0 です。

既定値 `AttachWhenLoaded` は Windows Server 2003 Service Pack 1 (SP1) および以降のバージョンの 1。

`AttachWhenLoaded` レジストリ値は、Windows Vista に存在しないと、Windows の以前のバージョンにのみ適用されます。

ミニフィルター ドライバーは、Windows Vista より前の Windows にインストールするときにソフトウェアのインストーラーを設定する必要があります `AttachWhenLoaded` このレジストリ値が 1 に現在設定されていない場合は 1 にします。場合の以前の値 `AttachWhenLoaded` が 0 の場合、インストーラーはミニフィルター ドライバーのインストール後に、システムを再起動する必要があります。

開発とテスト ツール

2020/04/24 • [Edit Online](#)

IFS Kit for Windows Server 2003 SP1 とで、Windows Driver Kit (WDK) Windows Vista 以降、このセクションで説明されているフィルター マネージャー ツールが提供されます。

ミニフィルター ドライバー開発者向けに汎用のカーネル モード開発を使用することをお勧めもと、ドライバー固有のルールと PREfast などのツールをテストします。

Fltmc.exe コントロール プログラム

Fltmc.exe コントロール プログラムは、一般的なミニフィルター ドライバーの管理操作のコマンド ライン ユーティリティです。開発者は、読み込みとミニフィルター ドライバーをアンロード、ミニフィルター ドライバーをボリュームにアタッチまたはボリュームからのデタッチに Fltmc.exe を使用して、ミニフィルター ドライバー、インスタンス、およびボリュームを列挙できます。

! fltkd デバッガー拡張機能

! で fltkd デバッガーの拡張機能の提供、[Windows デバッグ](#)ツール。よく使用されるコマンドを以下に示します。

!!!!	!!
!cbd	フィルター マネージャーと同等の! irp
! █	指定されたフィルターに関する詳細な情報を一覧表示
! █	すべてのアタッチされたミニフィルター ドライバーを一覧します。
! █	リストのすべてのマネージャーのフレームのフィルターし、ミニフィルター ドライバーの接続
! █	指定したインスタンスに関する詳細な情報を一覧表示
! █	指定したボリュームに関する詳細な情報を一覧表示
! █	すべてのボリュームを一覧表示し、ミニフィルター ドライバーのインスタンスに接続されています。

デバッグの詳細については、一般的なエラーをキャッチする多数のアサート ステートメントを含む、Fltmgr.sys のデバッグ バージョンとミニフィルター ドライバーをテストします。

フィルターの検証方法

フィルターの検証ツールは、[I/O の検証](#)オプションDriver Verifierミニフィルター ドライバーの使用状況のフィルター マネージャーの機能を検証します。フィルター検証ツールは、フィルター マネージャーと共にインストールされます。開発者は、Driver Verifier でミニフィルター ドライバーを開発常にする必要があり、フィルター検証機能を有効にします。

フィルター検証ツールを使用するには、ミニフィルター ドライバーの名前を指定し、Driver Verifier (Verifier.exe) で I/O の検証オプションを有効にします。フィルター マネージャーとミニフィルター ドライバーの登録時に検証を開始します。

フィルター検証ツールは、ミニフィルター ドライバーでは次の使用法を検証します。

- パラメーターと呼び出し元のコンテキストの正しい使用
- Preoperation と postoperation コールバック ルーチンから適切な戻り値
- コールバック データのパラメーターに、首尾一貫した変更

フィルター検証ツールは、次のフィルター マネージャー オブジェクトを追跡します。

- コンテキスト
- コールバックのデータ構造体
- キューに置かれた作業項目
- NameInformation 構造体
- ファイル オブジェクト
- フィルター オブジェクト
- インスタンス オブジェクト
- Volume オブジェクト

レガシ フィルター ドライバーの移植ガイドライン

2020/04/24 • [Edit Online](#)

開発者は、フィルタードライバーの機能を強化し、システムの信頼性を向上させるために、従来のフィルタードライバーをフィルターマネージャーモデルに移植することをお勧めします。経験豊富な開発者は、従来のフィルタードライバーをミニフィルタードライバーに移植するのは比較的簡単です。Microsoft のフィルタードライバー開発者は、次の方法をお勧めします。

- 信頼性の高い回帰テストスイートから開始して、レガシフィルタードライバーと移植されたミニフィルタードライバーの動作を検証します。
- ミニフィルタードライバーシェルを作成し、従来のフィルタードライバーからミニフィルタードライバーに機能を体系的に移動します。たとえば、添付ファイルが動作していることを確認し、一度に1つの操作を移植して、各操作の後にテストします。
- ユーザーモード/カーネルモードの通信が最後に変更されたので、既存のツールを使用してミニフィルタードライバーをテストすることができます。
- PRefast を使用してコンパイルし、ドライバー検証ツールの [検証ツール i/o 検証] オプションを使用してテストします。

移植プロセスでは、フィルターマネージャーの機能を最大限に活用するために、すべてのレガシフィルタードライバーコードを確認する必要があります。特に、次の点に注意してください。

- IRP ベースの i/o および高速 i/o 操作は、必要に応じて同じ操作を行うことができます。これにより、コードの重複を減らすことができます。
- 操作に登録すると、ミニフィルタードライバーは、すべてのページング i/o とキャッシュされた i/o を無視するように明示的に選択できます。これにより、コードをチェックする必要がなくなります。
- インスタンス通知は、アタッチ/デタッチロジックを大幅に簡素化します。
- ミニフィルタードライバーで処理する必要がある操作に対してのみ登録します。他のすべてを無視できます。
- フィルターマネージャーのコンテキストと名前管理のサポートを活用してください。
- 非再帰的 i/o を発行するためのフィルターマネージャーサポートを利用します。
- レガシフィルタードライバーとは異なり、ミニフィルタードライバーはローカル変数に依存して、preoperation 処理から postoperation 処理までのコンテキストを維持することはできません。操作状態を格納するには、ルックアサイドリストを割り当てることを検討してください。
- 名前またはコンテキストで終了した場合は、必ず参照を解放してください。
- ユーザーモードの完了ポートは、キューを構築するための強力な手法を追加します。1つの名前付きポートへの接続は1つしか必要ありません。

次の表に、レガシフィルタードライバーでの一般的な操作と、それらの操作をフィルターマネージャーモデルにマップする方法を示します。

レガシ フィルター ドライバー	フィルターマネージャーモデル
XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX

XXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXX
完了ルーチンのないパススルー操作	ミニフィルタードライバーがこの種類の i/o 操作に対して機能しない場合は、この操作に対して preoperation または postoperation コールバックルーチンを登録しないでください。 それ以外の場合は、この操作に登録されている preoperation コールバックルーチンから FLT_PREOP_SUCCESS_NO_CALLBACK を返します。 「FLT_PREOP_SUCCESS_NO_CALLBACK を返す」を参照してください。
完了ルーチンを使用したパススルー操作	Preoperation コールバックルーチンから FLT_PREOP_SUCCESS_WITH_CALLBACK を返します。 「FLT_PREOP_SUCCESS_WITH_CALLBACK を返す」を参照してください。
Preoperation コールバックルーチンでの保留操作	必要に応じて FltLockUserBuffer を呼び出して、ワークスレッドでアクセスできるように、すべてのユーザバッファが適切にロックされていることを確認します。 FltAllocateDeferredIoWorkItem や FltQueueDeferredIoWorkItem などのサポートルーチンを呼び出してワークスレッドに作業をキューに置いてください。 Preoperation コールバックルーチンから FLT_PREOP_PENDING を返します。 I/o 操作をフィルターマネージャーに返す準備ができたなら、Fltcompletependingpreoperation を呼び出します。 「プリ操作コールバックルーチンでの I/o 操作の保留」を参照してください。
Postoperation コールバックルーチンでの保留操作	Preoperation コールバックルーチンで、FltLockUserBuffer を呼び出して、ワークスレッドでアクセスできるように、ユーザバッファが適切にロックされていることを確認します。 FltAllocateGenericWorkItem や Fltqueuegenericworkitem などのサポートルーチンを呼び出して、ワークスレッドに作業をキューに置いてください。 Postoperation コールバックルーチンから FLT_POSTOP_MORE_PROCESSING_REQUIRED を返します。 I/o 操作をフィルターマネージャーに返す準備ができたなら、Fltcompletependingpostoperation を呼び出します。 「Postoperation コールバックルーチンでの I/o 操作の保留」を参照してください。
操作を同期する	Preoperation コールバックルーチンから FLT_PREOP_SYNCHRONIZE を返します。 「FLT_PREOP_SYNCHRONIZE を返す」を参照してください。

XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX
Preoperation コールバックルーチンで操作を完了します。	操作の最後の操作の状態と、FLT_CALLBACK_DATA構造体の<i>iostatus</i>メンバーの情報を設定します。 Preoperation コールバックルーチンから FLT_PREOP_COMPLETE を返します。 「プリ操作コールバックルーチンでの I/O 操作の完了」を参照してください。
Preoperation コールバックルーチンで保留された後に操作を完了します	操作の最後の操作の状態と、FLT_CALLBACK_DATA構造体の<i>iostatus</i>メンバーの情報を設定します。 I/O 操作を処理するワーカースレッドからFltcompleteFLT_PREOP_COMPLETEDedpreoperationを呼び出し、コールアウト状態パラメーターとしてを渡します。 「プリ操作コールバックルーチンでの I/O 操作の完了」を参照してください。
完了ルーチンですべての完了作業を実行します	Postoperation コールバックルーチンから FLT_POSTOP_FINISHED_PROCESSING を返します。 「Postoperation コールバックルーチンの記述」を参照してください。
安全な IRQL での作業の完了	Postoperation コールバックルーチンからFltdo!Processingwhensafeを呼び出します。 「完了処理が安全な IRQL で実行されるようにする」を参照してください。
完了ルーチンからイベントを通知します	この操作の preoperation コールバックルーチンから FLT_PREOP_SYNCHRONIZE を返します。 フィルターマネージャーは、IRQL <= APC_LEVEL で、preoperation コールバックルーチンと同じスレッドコンテキストで postoperation コールバックルーチンを呼び出します。 「FLT_PREOP_SYNCHRONIZE を返す」を参照してください。
成功した作成操作の失敗	作成操作の postoperation コールバックルーチンからFltcancelfileopenを呼び出します。 操作のFLT_CALLBACK_DATA構造体の<i>iostatus</i>メンバーに、適切なエラー NTSTATUS 値を設定します。 FLT_POSTOP_FINISHED_PROCESSING を返します。 「Postoperation コールバックルーチンでの I/O 操作の失敗」を参照してください。
I/O 操作の高速 I/O パスを介して I/O を禁止する	操作の preoperation コールバックルーチンから FLT_STATUS_DISALLOW_FAST_IO を返します。 「事前操作コールバックルーチンでの高速 I/O 操作の禁止」を参照してください。

XXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX
I/O 操作のパラメーターを変更する	操作のFLT_CALLBACK_DATA構造体のiopbメンバーの変更されたパラメーター値を設定します。 FLT_CALLBACK_DATA 構造体のlostatusメンバーの内容を変更した場合を除き、FltSetCallbackDataDirtyを呼び出して、FLT_CALLBACK_DATA 構造をダーティとしてマークします。 「I/O 操作のパラメーターの変更」を参照してください。
操作のユーザーバッファをロックします	「 I/O 操作のユーザーバッファへのアクセス 」で説明されている手法とガイドラインを使用します。

ミニフィルター ドライバーのインストール

2020/04/24 • [Edit Online](#)

Microsoft Windows XP と以降のオペレーティング システムでは、INF ファイルと、インストール アプリケーションを使用して、ミニフィルター ドライバーをインストールする必要があります。(Windows 2000 と以前のオペレーティング システムでミニフィルター ドライバーがよくインストール サービス コントロール マネージャーによって)。

今後、ミニフィルター ドライバーに関する Windows ハードウェア認定キットの要件に対応するために必要とする INF ベースのインストールが必要です。「INF ベースのインストール」の意味をのみファイルをコピーして、レジストリの情報を格納する INF ファイルを使用する必要がありますに注意してください。INF ファイルのみを使用して、全体の製品をインストールする必要はありませんし、提供する必要はありません、「[インストールを右クリックして](#)」ドライバーのオプション。

このセクションの内容:

[ミニフィルター ドライバーの INF ファイルを作成します。](#)

[ロード順序グループとミニフィルター ドライバーの高度](#)

[高度が割り当てられています。](#)

[ミニフィルターの高度の要求](#)

[ポイント タグの要求を再解析します。](#)

ミニフィルター ドライバー用の INF ファイルの作成

2020/04/24 • [Edit Online](#)

ファイル システム ミニフィルター ドライバーの INF ファイルには、次のセクションでは、一般に含まれています。

バージョン (必須)

DestinationDirs (推奨されるが、省略可能)

DefaultInstall (必須)

DefaultInstall.Services (必須)

ServiceInstall (必須)

AddRegistry (必須)

DefaultUninstall (省略可能)

DefaultUninstall.Services (省略可能)

文字列 (必須)

注 以降では、Windows Vista x64 ベース システム、すべてのカーネル モード コンポーネントでは、ファイル システム ドライバー (ファイル システム、従来のフィルターおよびミニフィルター ドライバー) などの非 PnP (プラグ アンド プレイ) ドライバーを含む必要があります署名します。読み込みと実行します。このシナリオでは、次の一覧には、ファイル システム ドライバーに関連する情報が含まれています。

- ファイル システム ドライバーなど、非 PnP ドライバーの INF ファイルを格納する必要はありません[製造元]または[モデル]セクション。
- [SignTool](#) にある、コマンド ライン ツール、\bin\WDK のインストール ディレクトリの SelfSign ディレクトリを直接「記号を埋め込む」を使用するドライバー SYS実行可能ファイルです。パフォーマンス上の理由から、ブート開始ドライバーが埋め込みの署名を含める必要があります。
- INF ファイルでは、指定された、[Inf2cat](#)コマンド ライン ツールは、ドライバー パッケージのカタログ (.cat) ファイルを作成するために使用できます。カタログ ファイルのみを受信できる[WHQL](#)ロゴ署名します。
- 管理者特権で未署名のドライバを Windows Vista 以降の x64 ベース システムでインストールもできます。ドライバーを読み込む (およびため、実行する) に失敗するただし、署名されていないためです。
- ドライバーの署名の詳細については、次を参照してください。[ドライバーの署名](#)します。
- 運転署名プロセスの詳細については、次を参照してください。[カーネル モード コード署名のチュートリアル](#)します。
- カスタム カーネル モード開発ツールを含むすべてのカーネル モード コンポーネントに署名する必要があります。詳細については、次を参照してください。[開発およびテスト \(Windows Vista 以降\) の中にドライバーの署名](#)します。

バージョンのセクション (必須)

[バージョン](#)セクションでは指定クラスと GUID ミニフィルター ドライバーの種類によって決定される次のコード例に示すようにします。

```
[Version]
Signature    = "$WINDOWS NT$"
Class        = "ActivityMonitor"
ClassGuid    = {b86dff51-a31e-4bac-b3cf-e8cfe75c9fc2}
Provider     = %Msft%
DriverVer    = 10/09/2001,1.0.0.0
CatalogFile =
```

次の表は、値で、ファイル システム ミニフィルター ドライバーを指定する必要があります、[バージョン](#)セクション。

「	[I]
■	「\$WINDOWS NT \$」
Class	参照してください ファイル システム フィルター ドライバーのクラスとクラス Guid します。
ClassGuid	参照してください ファイル システム フィルター ドライバーのクラスとクラス Guid します。
Provider	INF ファイルでは、Microsoft 以外のプロバイダーを指定してください。
DriverVer	参照してください INF DriverVer ■■■■します。
CatalogFile	署名されているウイルス対策ミニフィルター ドライバーについては、このエントリには、WHQL が指定したカタログ ファイルの名前が含まれています。その他のミニフィルター ドライバーこのエントリは空白にする必要があります。詳細については、の説明を参照して、CatalogFileエントリ INF ■■■■ ■■■■します。

DestinationDirs セクション (推奨されるが、省略可能)

[DestinationDirs](#) セクション ミニフィルター ドライバーとアプリケーション ファイルのコピー先ディレクトリを指定します。

このセクションで、ServiceInstall セクションで、システム定義の数値でよく知られているシステムのディレクトリを指定することができます。これらの値の一覧は、次を参照してください。 [INF DestinationDirs セクション](#)します。次のコード例で、値 12 がドライバのディレクトリを参照 (%windir%\system32\ドライバー)、値 10 が Windows ディレクトリ (%windir%) を指します。

```
[DestinationDirs]
DefaultDestDir = 12
Minispy.DriverFiles = 12
Minispy.UserFiles  = 10,FltMgr
```

(必須) DefaultInstall セクション

[DefaultInstall](#) セクションで、[CopyFiles](#) ディレクティブ ミニフィルター ドライバーのドライバー ファイル、ユーザー アプリケーションのコピーファイルで指定されている変換先を [DestinationDirs](#) セクション。

注 、[CopyFiles](#) ディレクティブが、カタログ ファイルまたは INF ファイル自体を参照する必要があります。SetupAPI では、これらのファイルが自動的にコピーします。

Windows オペレーティング システムの複数のバージョンでは、ドライバーをインストールする 1 つの INF ファイルを作成することができます。この種類の INF ファイルを作成するには追加作成して [DefaultInstall](#)、[DefaultInstall.Services](#)、

DefaultUninstall、およびDefaultUninstall.Servicesセクションの各オペレーティング システムのバージョン。各セクションのラベルでは、[装飾](#)(.ntx86、.ntia64、または .nt など) を適用するオペレーティング システムのバージョンを指定します。この種類の INF ファイルを作成する方法の詳細については、次を参照してください。[INF ファイルを複数のプラットフォームやオペレーティング システムを作成する](#)します。

次のコード例を示しています、典型的な **DefaultInstall** セクション。

```
[DefaultInstall]
OptionDesc = %MinispyServiceDesc%
CopyFiles = Minispy.DriverFiles, Minispy.UserFiles
```

(必須) DefaultInstall.Services セクション

DefaultInstall.Services セクションが含まれています、**AddService** 方法とタイミングを制御するディレクティブのサービスを次のコード例に示すように、特定のドライバーが読み込まれます。

```
[DefaultInstall.Services]
AddService = %MinispyServiceName%,Minispy.Service
```

(必須) ServiceInstall セクション

ServiceInstallセクションには、ドライバー サービスを読み込むために使用される情報が含まれています。MiniSpy サンプル ドライバーのこのセクションの名前は"Minispy.Service"次のコード例に示すように。名前、**ServiceInstall**にセクションを表示する必要があります、**AddService** ディレクティブで、**DefaultInstall.Services**セクション。

```
[Minispy.Service]
DisplayName = %MinispyServiceName%
Description = %MinispyServiceDesc%
ServiceBinary = %12%\minispy.sys
ServiceType = 2 ; SERVICE_FILE_SYSTEM_DRIVER
StartType = 3 ; SERVICE_DEMAND_START
ErrorControl = 1 ; SERVICE_ERROR_NORMAL%
LoadOrderGroup = "FSFilter Activity Monitor"
AddReg = Minispy.AddRegistry
Dependencies = FltMgr
```

ServiceTypeエントリは、サービスの種類を指定します。ミニフィルター ドライバーが 2 の値を指定する必要があります (サービス_ファイル_システム_ドライバー)。詳細については、**ServiceType**エントリを参照してください [INF AddService ディレクティブ](#)します。

StartTypeエントリでは、サービスを開始するタイミングを指定します。次の表に、可能な値**StartType**とそれに対応する型を開始します。

[1]	「
0x00000000	SERVICE_BOOT_START
0x00000001	SERVICE_SYSTEM_START
0x00000002	SERVICE_AUTO_START
0x00000003	SERVICE_DEMAND_START

[1]	“
0x00000004	SERVICE_DISABLED

開始の詳細については、「ドライバーを開始型」を参照してください[何を決定しますと、ドライバーが読み込まれる](#)します。

LoadOrderGroup エントリがフィルター マネージャーとミニフィルター ドライバーと従来のファイル システム フィルター ドライバーの間の相互運用性を確認する必要がある情報を提供します。指定する必要があります、LoadOrderGroupを開発しているミニフィルター ドライバーの種類に適した値。ロード順序グループを選択するを参照してください。[ロード順序グループとミニフィルタードライバーの高度](#)します。

指定する必要がありますに注意してください、LoadOrderGroup 値ミニフィルター ドライバーのスタートアップの種類は、サービスではない場合でも、_ブート_を開始します。ミニフィルター ドライバーは、この方法で、従来のファイル システム フィルター ドライバーと異なります。

注 フィルター マネージャーのStartType値はサービス_ブート_を開始してそのLoadOrderGroup値は FSFilter インフラストラクチャ。これらの値は、フィルター マネージャーが、ミニフィルター ドライバーが読み込まれる前に常に読み込まれていることを確認します。

方法の詳細についてはStartTypeとLoadOrderGroup エントリが特定のドライバーが読み込まれるときを参照してください[何を決定しますと、ドライバーが読み込まれる](#)します。

注 ミニフィルター ドライバーは、従来のファイル システム フィルター ドライバーとは異なり、StartTypeとLoadOrderGroup 値は場所を特定できませんミニフィルター ドライバーミニフィルターのインスタンスのスタックにアタッチします。この場所はミニフィルター インスタンスが指定されている高度によって決まります。

ErrorControl エントリは、システムの起動時に開始するサービスが失敗した場合に実行されるアクションを指定します。ミニフィルター ドライバーが 1 の値を指定する必要があります (サービス_エラー_通常)。詳細については、ErrorControl エントリを参照してください [INF AddService ディレクティブ](#) します。

AddReg ディレクティブを指すライター定義されている 1 つまたは複数の INF AddRegistry セクションでは、新しくインストールされたため、レジストリに格納される情報を含むサービス。ミニフィルター ドライバーを使用して、AddRegistry セクション ミニフィルター ドライバーのインスタンスを定義して、既定のインスタンスを指定します。

依存関係 エントリを任意のサービスまたはドライバーが依存するロード順序グループの名前を指定します。ミニフィルター ドライバーをすべて FltMgr、これはフィルター マネージャーのサービスの名前を指定してください。

(必須) AddRegistry セクション

AddRegistry セクション キーと値をレジストリに追加します。ミニフィルター ドライバーを使用して、AddRegistry セクション ミニフィルターのインスタンスを定義して、既定のインスタンスを指定します。フィルター マネージャー ミニフィルター ドライバーの新しいインスタンスを作成するたびに、この情報が使用されます。

MiniSpy サンプル ドライバーで、次のAddRegistryで %strkey% トークン定義と共に、セクション、[文字列](#) セクションで、次の 3 つのインスタンスを定義します。MiniSpy サンプル ドライバーの既定のインスタンスとしてその 1 つと呼びます。

```
[Minispy.AddRegistry]
HKR,%RegInstancesSubkeyName%,%RegDefaultInstanceValueName%,0x00000000,%DefaultInstance%
HKR,%RegInstancesSubkeyName%\ "%Instance1.Name%,%RegAltitudeValueName%,0x00000000,%Instance1.Altitude%
HKR,%RegInstancesSubkeyName%\ "%Instance1.Name%,%RegFlagsValueName%,0x00010001,%Instance1.Flags%
HKR,%RegInstancesSubkeyName%\ "%Instance2.Name%,%RegAltitudeValueName%,0x00000000,%Instance2.Altitude%
HKR,%RegInstancesSubkeyName%\ "%Instance2.Name%,%RegFlagsValueName%,0x00010001,%Instance2.Flags%
HKR,%RegInstancesSubkeyName%\ "%Instance3.Name%,%RegAltitudeValueName%,0x00000000,%Instance3.Altitude%
HKR,%RegInstancesSubkeyName%\ "%Instance3.Name%,%RegFlagsValueName%,0x00010001,%Instance3.Flags%
```

DefaultUninstall セクション (省略可能)

DefaultUninstall セクションは省略可能ですが、ドライバーをアンインストールする場合はお勧めします。含まれている

DelFiles としてディレクティブを次のコード例に示すように、ファイルおよびレジストリのエントリを削除します。

```
[DefaultUninstall]
DelFiles    = Minispy.DriverFiles, Minispy.UserFiles
DelReg      = Minispy.DelRegistry
```

DefaultUninstall.Services セクション (省略可能)

DefaultUninstall.Services セクションは省略可能ですが、ドライバーをアンインストールする場合はお勧めします。含まれている **DelService** MiniSpy サンプル ドライバーから次のコード例に示すように、ミニフィルター ドライバーのサービスを削除するためのディレクティブ。

注、**DelService** ディレクティブは、SPSVCINST を常に指定する必要があります_STOPSERVICE フラグ (0x00000200) が削除される前に、サービスを停止します。

```
[DefaultUninstall.Services]
DelService = Minispy,0x200
```

文字列のセクション (必須)

文字列 セクションは、INF ファイルで使用されている各 %strkey% トークンを定義します。

1 つの国際 INF ファイルを作成するにはロケールに固有の追加を作成して**文字列**。LanguageID INF ファイルのセクション。国際対応の INF ファイルの詳細については、次を参照してください。 [International INF ファイルの作成](#)です。

次のコード例を示しています、典型的な**文字列** セクション。

```
[Strings]
Msft                = "Microsoft Corporation"
MinispyServiceDesc  = "Minispy mini-filter driver"
MinispyServiceName  = "Minispy"
RegInstancesSubkeyName = "Instances"
RegDefaultInstanceValueName = "DefaultInstance"
RegAltitudeValueName  = "Altitude"
RegFlagsValueName    = "Flags"

DefaultInstance      = "Minispy - Top Instance"
Instance1.Name        = "Minispy - Middle Instance"
Instance1.Altitude    = "370000"
Instance1.Flags       = 0x1 ; Suppress automatic attachments
Instance2.Name        = "Minispy - Bottom Instance"
Instance2.Altitude    = "365000"
Instance2.Flags       = 0x1 ; Suppress automatic attachments
Instance3.Name        = "Minispy - Top Instance"
Instance3.Altitude    = "385000"
Instance3.Flags       = 0x1 ; Suppress automatic attachments
```

ミニフィルター ドライバー用のロード順序グループと高度

2020/04/24 • [Edit Online](#)

Windows では、システムの起動時に読み込まれるファイルシステムフィルタードライバーとミニフィルタードライバーに対して、一連の専用の読み込み順序グループを使用します。

レガシファイルシステムフィルタードライバーは、既存のファイルシステムドライバースタックの一番上にもみアタッチでき、スタックの途中でアタッチすることはできません。このため、従来のファイルシステムフィルタードライバーでは、ドライバーおよび読み込み順序グループの開始の種類が重要です。以前のフィルタードライバーが読み込まれるため、ファイルシステムのドライバースタックに接続することができます。

ドライバーは、ドライバーの開始の種類に基づいて最初に読み込まれます。これは、システムの起動フェーズを表します。開始の種類の詳細については、「[ドライバーの起動の種類](#)」を[参照してください](#)。スタートアップの種類がサービス_起動_を指定するすべてのファイルシステムフィルタードライバーとミニフィルタードライバーは、スタートアップの種類がサービス_システム_開始またはサービス_自動_開始するドライバーの前に読み込まれます。スタートアップの種類は、ミニフィルタードライバーをインストールするために使用される INF ファイルの ServiceInstall セクションのstarttypeエントリによって指定されます。[開始の種類] カテゴリごとに、[読み込み順序] グループによって、ファイルシステムフィルタードライバーとミニフィルタードライバーがいつ読み込まれるかが決定されます。

ミニフィルタードライバーは、いつでも読み込むことができます。従来のファイルシステムフィルタードライバーとの相互運用性を実現するために、ミニフィルタードライバーでは、負荷順序グループの概念が必要です。すべてのミニフィルタードライバーには、高度と呼ばれる一意の識別子が必要です。ミニフィルタードライバーの高度は、ミニフィルタードライバーが読み込まれるときに、i/o スタック内の他のミニフィルタードライバーとの相対的な位置を定義します。高度は、10進数として解釈される無限精度の文字列です。数値の高度が低いミニフィルタードライバーは、数値が大きいミニフィルタードライバーの下に i/o スタックに読み込まれます。

各読み込み順序グループには、高度な範囲が定義されています。ミニフィルタードライバーへの高度な割り当ては、Microsoft によって管理されます。ミニフィルタードライバーの高度を要求するには、に電子メールメッセージを送信して、割り当てられているかどうかを確認 fsfcomm@microsoft.com ます。

ミニフィルタードライバーでは、読み込み順序グループを表す高度な範囲の高度な値を指定する必要があります。ミニフィルタードライバーの高度値は、ミニフィルタードライバーのインストールに使用される INF ファイルの文字列セクションのインスタンス定義で指定されます。インスタンス定義は、[FLT_REGISTRATION](#) 構造体の[instancesetupcallback](#)ルーチンの呼び出しで指定することもできます。ミニフィルタードライバーには、複数のインスタンスと高度を定義できます。これらのインスタンス定義は、すべてのボリュームに適用されます。

次のルールでは、開始の種類と読み込み順序のグループについて、ミニフィルタードライバーがいつ読み込まれるかを決定します。

- 特定の開始の種類と負荷の順序のグループを指定するミニフィルタードライバーは、その開始の種類と負荷の順序グループの他のファイルシステムフィルタードライバーおよびミニフィルタードライバーと同時に読み込まれます。
- 各読み込み順序グループ内で、ファイルシステムフィルタードライバーとミニフィルタードライバーは、通常、ランダムな順序で読み込まれます。これにより、ドライバーがインストールされた順序に基づいてドライバーが読み込まれます。
- ファイルシステムフィルタードライバーまたはミニフィルタードライバーによって、読み込み順序グループが指定されていない場合は、同じスタートアップの種類の中のすべてのドライバーが読み込み順序グループを指定した後に読み込まれます。

次の表に、システム定義の読み込み順序グループとミニフィルタードライバーの高度な範囲を示します。各読み込み順序グループについて、[読み込み順序グループ] 列には、ミニフィルターの INF ファイルの ServiceInstall セクションのLoadordergroupエントリでそのグループに指定する値が含まれています。[高度範囲] 列には、特定の負荷順序グループの高度な範囲が含まれます。ミニフィルタードライバーでは、適切な負荷順序のグループまたはグループ内の Microsoft からの高度な割り当てを要求する

必要があります。

読み込み順序グループと高度な範囲は、スタックに表示されているとおりに一覧表示されます。これは、読み込まれた順序の逆です。

グループ名	範囲	説明
フィルター	420000-429999	このグループは、Windows 2000 以前で使用できたフィルター読み込み順序グループと同じです。このグループは最後に読み込まれるため、ファイルシステムから最も遠いものがアタッチされます。
上にある FSFilter	400000-409999	このグループは、他のすべての FSFilter の種類の上にアタッチする必要があるフィルタードライバ用に用意されています。
FSFilter 利用状況モニター	360000-389999	このグループには、ファイル i/o を監視および報告するフィルタードライバが含まれています。
FSFilter の削除の取り消し	340000-349999	このグループには、削除されたファイルを回復するフィルターが含まれています。
FSFilter ウイルス対策	320000-329999	このグループには、ファイル i/o 中にウイルスを検出して駆除するフィルタードライバが含まれています。
FSFilter レプリケーション	300000-309999	このグループには、ファイルデータをリモートサーバーにレプリケートするフィルタードライバが含まれています。
FSFilter 連続バックアップ	280000-289999	このグループには、ファイルデータをバックアップメディアにレプリケートするフィルタードライバが含まれています。
FSFilter コンテンツスクリーナー	260000-269999	このグループには、特定のファイルまたはファイルコンテンツを作成できないようにするフィルタードライバが含まれています。
FSFilter クォータの管理	240000-249999	このグループには、拡張ファイルシステムのクォータを提供するフィルタードライバが含まれています。
FSFilter システムの回復	220000-229999	このグループには、システムの復元 (SR) フィルターなど、オペレーティングシステムの整合性を維持する操作を実行するフィルタードライバが含まれています。

機能名	機能 ID	説明
FSFilter クラスターファイルシステム	200000-209999	このグループには、ネットワーク経由でファイルサーバーのメタデータを提供する製品で使用されるフィルタードライバーが含まれています。
FSFilter HSM	180000-189999	このグループには、階層型記憶域の管理を実行するフィルタードライバーが含まれています。
FSFilter イメージング	170000-175000	このグループには、仮想名前空間を提供する ZIP 形式のフィルタードライバーが含まれています。 この読み込みグループは、Windows Vista 以降のバージョンのオペレーティングシステムで使用できます。
FSFilter の圧縮	160000-169999	このグループには、ファイルデータの圧縮を実行するフィルタードライバーが含まれています。
FSFilter の暗号化	140000-149999	このグループには、ファイル i/o 中にデータを暗号化および暗号化解除するフィルタードライバーが含まれています。
FSFilter の仮想化	13万 ~ 139999	このグループには、Windows Vista で追加された承認されていないユーザー (LUA) フィルタードライバーなど、ファイルパスを仮想化するフィルタードライバーが含まれています。 この読み込みグループは、Windows Vista 以降のバージョンのオペレーティングシステムで使用できます。
FSFilter の物理クォータ管理	120000-129999	このグループには、物理ブロック数を使用してクォータを管理するフィルタードライバーが含まれています。
FSFilter 開いているファイル	100000-109999	このグループには、既に関いているファイルのスナップショットを提供するフィルタードライバーが含まれています。
FSFilter セキュリティ言え	80000-89999	このグループには、ロックダウンおよび拡張アクセス制御リスト (Acl) を適用するフィルタードライバーが含まれています。
FSFilter のコピー保護	60000-69999	このグループには、メディア上の帯域外データをチェックするフィルタードライバーが含まれています。

XXXXXXXXXX	XXXX	II
下位の FSFilter	40000-49999	このグループは、他のすべての FSFilter の種類の下に接続する必要があるフィルタードライバ用に用意されています。
FSFilter システム	20000-29999	内部使用のために予約済みです。
FSFilter インフラストラクチャ		内部使用のために予約済みです。このグループは最初に読み込まれるため、ファイルシステムに最も近い場所にアタッチされます。

割り当て済み高度

2020/04/24 • [Edit Online](#)

フィルターマネージャーモデルに対して開発されたファイルシステムミニパスのドライバーは、ファイルシステムスタック内に存在する他のミニフィルターとの相対的な位置を定義する高度と呼ばれる一意の識別子を持っている必要があります。ミニフィルターの高度は、ミニフィルターの要件と負荷の順序のグループに基づいて Microsoft によって割り当てられます。

ミニフィルターの高度な数値を要求するには、「[ミニフィルターの高度な要求](#)」を参照してください。

このページの発行日の時点で割り当てられた標高割り当ては、次の各読み込み順序グループについて以下の一覧に記載されています (注: このページは、1年あたり1-2 回更新されます)。

42万-429999: フィルター

⌘	⌘	COMPANY
ntoskrnl.exe	425500	マイクロソフト
ntoskrnl.exe	425000	マイクロソフト

40万-409999: FSFilter Top

⌘	⌘	COMPANY
wcnfs.sys	409900	マイクロソフト
bindflt	409800	マイクロソフト
cldflt	409500	マイクロソフト
iorate	409010	マイクロソフト
ioqos.sys	409000	マイクロソフト
fsdepends.sys に依存します	407000	マイクロソフト
sftredir	406000	マイクロソフト
dfs-r	405000	マイクロソフト
VeeamFCT.sys	404920	Veeam ソフトウェア
tracker	404910	Acronis
csvnsflt	404900	マイクロソフト
csvflt	404800	マイクロソフト
Uev. AgentDriver.sys	404710	マイクロソフト

氏	氏	COMPANY
AppvVfs	404700	マイクロソフト
CCFFilter .sys	404600	マイクロソフト
U-Agentdrv .sys	402110	膨大な制限 GmbH
mrigflt	402100	最重要ソフトウェア Ltd.
dciogrd	402010	Datacloak Tech
Dewdrv .sys	402000	Dell テクノロジ
zsusbstorfilt	401910	Zshield Inc.
eaw .sys	401900	Raytheon サイバーソリューション
TVFsfiler	401800	TrustView
KKDiskProtector	401700	Goldmsg
AgentComm	401600	Trustwave 持株 Inc.
rvsavd	401500	CJSC Returnil ソフトウェア
DGMinFlt	401410	デジタルガーディアン Inc.
dgdmk	401400	Verdasys Inc.
tusbstorfilt	401300	SimplyCore の場合
pcgenfam .sys	401200	Soluto
atr sdfw .sys	401100	Altiris
tpfilter .sys	401000	RedPhone のセキュリティ
MBIG2Prot	400920	Malwarebytes Inc.
mbamwatchdog .sys	400900	Malwarebytes Corporation
DSESafeCtrlDrv	400803	上海 Eff-ソフト IT
edevmon	400800	ESET spol. s r.o.
vmwflstor .sys	400700	VMware, Inc.
TsQBDrv .sys	400600	Tencent テクノロジ
PolyPortFlt	400490	PolyPort Inc.

氏名	氏名	COMPANY
Dscdriver .sys	400300	Dell Technologies Inc.

36万-389999: FSFilter 利用状況モニター

氏名	氏名	COMPANY
klboot	389510	Kaspersky ラボ
klfdefsf	389500	Kaspersky ラボ
SeRdr. sys	389450	rhipe オーストラリア Pty
storagedrv .sys	389400	SMTechnology Co.
path8ft	389320	Telefónica デジタル
NgScan	389310	Acronis
icrlmonitor	389300	工業技術
gibepcore .sys	389290	グループ-IB LTD.
enmon	389280	OpenText Corp
RansomDetect	389270	WidgetNuri Corp
cbfsfilter2017	389260	モバイルコンテンツ管理
CBFSFilter2017	389250	SecureLink Inc.
cbfsfilter2017	389245	南京 Geomarking
NWEDriver .sys	389240	Dell テクノロジ
cytmon. sys	389230	Cytrence Inc.
SophosED	389220	Sophos
.Sys	389210	Somma Inc.
IFS64	389200	Ashampoo 開発
TSTFsReDir	389192	ThinScale Tech
Tstreグリーンディレクトリ .sys	389191	ThinScale Tech
TSTFilter	389190	ThinScale Tech
VrnsFilter .sys	389180	Varonis Ltd.

氏名	氏名	COMPANY
slb_guard.sys	389175	Lenovo 北京
lrtp.sys	389170	Lenovo 北京
ipcomfltr	389160	Bluzen Inc.
SvCBT	389150	Spハーソフトテクノロジ
mbamshuriken.sys	389140	Malwarebytes
ContainerMonitor	389130	水色のセキュリティ
cmflt	389125	Certero
SaMFlt	389120	DreamCrafts
RuiMinispy	389117	このように
RuiFileAccess	389115	このように
システムの	389113	このように
. Sys	389111	このように
windd.sys	389110	Comae 技術
cbfsfilter2017	389105	ネットワークでの base
taobserveflt	389100	ThinAir Labs Inc.
bsrfsflt	389096	Man テクノロジ Inc.
fsrfilter.sys	389094	Man テクノロジ Inc.
vollock.sys	389092	Man テクノロジ Inc.
drbdlock	389090	Man テクノロジ Inc.
dcfsgrd	389085	Datacloak Tech
hsmIltmon	389080	Hitachi ソリューション
AternityRegistryHook	389070	Aternity Ltd.
MozyNextFilter	389068	Carbonite Inc.
MozyCorpFilter	389067	Carbonite Inc.
MozyEntFilter	389066	Carbonite Inc.

氏名	氏名	COMPANY
MozyOEMFilter	389065	Carbonite Inc.
MozyEnterpriseFilter	389064	Carbonite Inc.
MozyProFilter	389063	Carbonite Inc.
MozyHomeFilter	389062	Carbonite Inc.
BDSFilter.sys	389061	Carbonite Inc.
CSBFilter.sys	389060	Carbonite Inc.
ChemometecFilter	389050	ChemoMetec
SentinelMonitor	389040	SentinelOne
DhWatchdog	389030	マイクロソフト
edrsensor.sys	389025	Bitdefender SRL
bdprivmon	389022	Bitdefender SRL
NpEtw.sys	389020	お金
OczMiniFilter フィルター.sys	389010	OCZ ストレージ
ielcp	389004	Intel Corporation
IESlp	389002	Intel Corporation
IntelCAS	389000	Intel Corporation
boxifier	388990	Kenubi
SamsungRapidFSFltr	388980	NVELO Inc.
drsfile	388970	MRY います。
CbFltFs4	388966	Simopro テクノロジ
CrUnCopy	388964	Shenzhen CloudRiver
aictracedrv_am.sys	388960	AI コンサルティング
fiopolicyfilter	388954	SanDisk Inc.
sodatpfl.sys	388951	SODATSW spol. s r.o.
sodatpfl.sys	388950.2	SODATSW

№	№	COMPANY
fcontrol.sys	388950	SODATSW spol. s r.o.
qfilter.sys	388940	クォーラムラボ
Redlight.sys	388930	Trustware Ltd.
.eps	388920	Lumension
VHDTrack	388915	会社 onis Inc.
VHDDelta	388912	Niriva の場合
FSTrace	388910	Niriva の場合
YahooStorage	388900	Yahoo 日本 Corporation
KeWF.sys	388890	KEBA AG
epregflt	388888	チェックポイントソフトウェア
epklib.sys	388886	チェックポイントソフトウェア
zsfprt	388880	k4solution Co.
dsflt	388876	cEncrypt
bfaccess.sys	388872	bitFence Inc.
xcpl.sys	388870	X-クラウドシステム
DFMFilter.sys	388867	ManageEngine Zoho
DCFAFilter.sys	388866	ManageEngine Zoho
RMPHVMonitor.sys	388865	ManageEngine Zoho
FAPMonitor.sys	388864	ManageEngine Zoho
EaseFIt	388860	EaseVault テクノロジ Inc.
rpq ウォッチャー	388855	最高のセキュリティ
sieflt	388852	迅速な回復テクノロジ Pvt. 社.
cssdIp	388851	迅速な回復テクノロジ Pvt. 社.
cssdIp	388850	CoSoSys
INISBDrv64	388840	Initech Inc.

№	№	COMPANY
トレース .sys	388831	Fitsec Ltd.
SandDriver	388830	Fitsec Ltd.
dskmn	388820	Honeycomb テクノロジ
offsm. sys	388811	Jiransoft Co, Ltd.
xkfsfd .sys	388810	Jiransoft Co, Ltd.
J、sys	388808	Jiransoft Co, Ltd.
JKPPXK. sys	388807	Jiransoft Co, Ltd.
JKPPPF .sys	388806	Jiransoft Co, Ltd.
J、sys	388805	Jiransoft Co, Ltd.
pcpifd .sys	388800	Jiransoft Co, Ltd.
NNTInfo .sys	388790	新しいネットテクノロジーが制限されています
FsMonitor .sys	388780	IBM
CVCBT .sys	388770	CommVault Systems, Inc.
AwareCore	388760	TaaSera Inc.
laFS. sys	388750	NetworkProfi Ltd.
fsnk	388740	SoftPerfect な研究
RGNT .sys	388730	HFN Inc.
fltRs329	388720	セキュリティで保護された地球 Inc.
ospmon	388710	SC ODEKIN SOLUTIONS の SRL
edsigk .sys	388700	Enterprise Data Solutions, Inc.
fiometer	388692	Fusion-io
dcSnapRestore .sys	388690	Fusion-io
fam. sys	388680	迅速な回復テクノロジー Pvt. 社.
vidderfs	388675	Vidder Inc.
Tritiumfltr	388670	Tritium Inc.

氏名	氏名	COMPANY
HexisFSMonitor	388660	Hexis サイバーソリューション
BlackbirdFSA	388650	BeyondTrust Inc.
TMUMS	388642	Trend Micro Inc.
hfileflt	388640	Trend Micro Inc.
TMUMH.sys	388630	Trend Micro Inc.
AcDriver.sys	388620	Trend Micro, Inc.
SakFile	388610	Trend Micro, Inc.
SakMFile	388600	Trend Micro, Inc.
rsfdrv.sys	388580	Clonix Co
alcapture.sys	388570	エアロックデジタル Pty Ltd.
kmNWCH	388560	IGLOO SOFTWARE SECURITY, Inc.
ISIRMFmon	388550	アルプスのシステム統合 CO。
EsProbe.sys	388542	Stormshield
heimdall	388540	Arkoon のネットワークセキュリティ
thetta	388532	Syncopate
thetta	388531	Syncopate
thetta	388530	Syncopate
DTPL.sys	388520	CONNECT SHIFT LTD.
CyOptics.sys	388514	Cylance Inc.
CyProtectDrv32	388510	Cylance Inc.
CyProtectDrv64	388510	Cylance Inc.
tbfsflt	388500	3番目のやバケツリレー
IvAppMon	388491	Ivanti
LDSecDrv.sys	388490	LANDESK ソフトウェア
SGResFlt	388480	Samsung SDS Ltd.

№	№	COMPANY
CwMem2k64	388470	ApSoft
axftdrv.sys	388460	Axact Pvt Ltd.
RMDiskMon.sys	388450	Qingdao の Anmei ネットワークテクノロジー Co。
diskactmon.sys	388440	Qingdao の Anmei ネットワークテクノロジー Co。
Codex.sys	388430	いいね。
CatMF.sys	388420	Logichron Inc.
RW7FsFit	388410	PJSC KP VTI
aswSP.sys	388401	Avast ソフトウェア
aswFsBlk.sys	388400	ALWIL ソフトウェア
ThreatStackFIM	388380	脅威スタック
BOsCmFit	388370	Barkly は Inc. を保護します。
BOsFsFiltr	388370	Barkly は Inc. を保護します。
Asgard	388365	KNET EOOD
FeKern.sys	388360	焼討
libwamf	388350	OPSWAT Inc.
SZEDRDrv.sys	388346	安全です。
szシャード	388345	安全です。
szpcmdrv	388341	安全です。
szdfmdrv.sys	388340	安全です。
szdfmdrv_usb.sys	388331	安全です。
sprtdrv	388330	安全です。
SWFsFiltrv2	388321	お金
SWFsFiltr	388320	お金
flashaccelfs	388310	ネットワークアプライアンス

氏名	氏名	COMPANY
changelog	388300	ネットワークアプライアンス
stcvsm.sys	388250	StorageCraft Tech
システム	388240	ITSTATION Inc.
fshs	388222	F-セキュリティ保護
fshs	388221	F-セキュリティ保護
fsatp.sys	388220	F-セキュリティ保護
SecdoDriver.sys	388210	Secdo
TGFSMF	388200	Tetraglyph テクノロジ
evscase.sys	388100	3月の h Software Ltd.
VSScanner.sys	388050	VoodooSoft
HDRansomOffDrv	388044	He・g の防御
HDCorrelateFDrv	388042	He・g の防御
HDFFileMon.sys	388040	He・g の防御
tsifilemon.sys	388012	インターコム Inc.
MarSpy.sys	388010	インターコム Inc.
AGSysLock.sys	388002	AppGuard の場合
AGSecLock.sys	388001	AppGuard の場合
BrnFileLock.sys	388000	Blue ねじ山 Networks
BrnSecLock.sys	387990	Blue ねじ山 Networks
LCmPrintMon	387978	YATEM Co. Ltd.
LCgAdMon	387977	YATEM Co. Ltd.
LCmAdMon	387976	YATEM Co. Ltd.
LCgFileMon.sys	387975	YATEM Co. Ltd.
LCmFile	387974	YATEM Co. Ltd.
LCgFile.sys	387972	YATEM Co. Ltd.

№	№	COMPANY
LCmFileMon	387970	YATEM Co. Ltd.
IridiumSwitch	387960	# O
scensemon	387950	AppiXoft
ruaff	387940	RUNEXY
bbfilter .sys	387930	derivo GmbH
Bfmon	387920	Baidu (香港特别行政区) 限定
bdsysmon	387912	Baidu Online ネットワーク
BdRdFolder .sys	387910	Baidu (北京)
ml、.sys	387901	RUNEXY
pscff .sys	387900	Weing、Ltd.、
fcnotify .sys	387880	TCXA Ltd.
aaf. sys	387860	Actifio Inc.
gddcv	387840	G データソフトウェア AG
wgfile	387820	オレンジ色の WERKS Inc.
zesfsmf .sys	387800	社
uamflt	387700	Sevtechnotrans
ehdrv .sys	387600	ESET、 spol. s r.o.
DattoFSF	387560	Datto Inc.
RubrikFileAudit	387552	Rubrik Inc.
FileSystemCBT	387550	Rubrik Inc.
Snilog .sys	387500	Systemneeds 必要性, Inc.
tss	387400	Tiversa Inc
LmDriver .sys	387390	ソフト Kft.
WDCFilter .sys	387330	Interset Inc.
altcbt .sys	387320	Altaro Ltd.

II	II	COMPANY
bapfecpt	387310	BitArmor システム、Inc.
bamfltr	387300	BitArmor システム、Inc.
TrustedEdgeFfd .sys	387200	FileTek, Inc.
MRxGoogle	387100	Google, Inc.
YFSDR。SYS.DATABASES	387010	Yokogawa R & L Corp
YFSD.SYS.DATABASES	387000	Yokogawa R & L Corp
YFSRD sys	386990	Yokogawa R & L Corp
psgfoctrl	386990	Yokogawa R & L Corp
psgdflt	386980	Yokogawa R & L Corp
USBDPH。SYS.DATABASES	386901	高度なコンピューティングを開発するためのセンター
pecfilter	386900	C-DAC ハイデラバード
G/FCB. sys	386810	インカインターネット Co。
GKPFCB64	386810	インカインターネット Co。
32ビットの TkPcFtCb	386800	インカインターネット Co., Ltd.
TkPcFtCb64 (64 ビット)	386800	インカインターネット Co., Ltd.
bmregdrv .sys	386731	Yandex の場合
bmfdrv .sys	386730	Yandex の場合
CarbonBlackK	386720	Bit9 Inc.
ScAuthFSflt	386710	セキュリティコードの例
ScAuthloDrv .sys	386700	セキュリティコードの例
mfeaskm	386610	McAfee Inc.
mfencfilter .sys	386600	McAfee
WinFLAHdrv	386540	NewSoftwares。net, Inc.
WinFLAdrv. sys	386530	NewSoftwares。net, Inc.
WinDBdrv .sys	386520	NewSoftwares。net, Inc.

氏 名	氏 名	COMPANY
WinFLdrv .sys	386510	NewSoftwares。net, Inc.
WinFPdrv. sys	386500	NewSoftwares。net, Inc.
varpffmon. sys	386486	Varlook Ltd.
SkyWPDrv	386435	スカイ Co., Ltd.
SkyRGDrv	386431	スカイ Co., LTD.
SkyAMDrv	386430	スカイ Co., LTD.
SheedSelfProtection. sys	386421	SheedSoft Ltd.
arta	386420	SheedSoft Ltd.
ApexSqlFilterDriver .sys	386410	ApexSQL の場合
stflt	386400	Xacti
tbrdrv .sys	386390	クローラーグループ
WinTeonMiniFilter フィルター	386320	Dmitry Stefankov
ワイパー	386310	Dmitry Stefankov
DevMonMiniFilter システム	386300	Dmitry Stefankov
VMWVvpfsd	386200	VMware, Inc.
RTOLogon (名前の変更)	386200	VMware, Inc.
Code42Filter	386190	Code42
ConduantFSFltr	386180	Conduant Corporation
KtFSFilter .sys	386170	鍵視テクノロジー
FileGuard. sys	386140	RES ソフトウェア
NetGuard. sys	386130	RES ソフトウェア
RegGuard. sys	386120	RES ソフトウェア
ImgGuard	386110	RES ソフトウェア
AppGuard. sys	386100	RES ソフトウェア
自分の Idiskfs .sys	386030	このように

氏名	氏名	COMPANY
minitr	386020	保護されたネットワーク
cpepmon	386010	チェックポイントソフトウェア
CGWMF.sys	386000	NetIQ
ISRegFlt	385990	Flexera Software Inc.
ISRegFlt64	385990	Flexera Software Inc.
shdlpSf	385970	Comtrue テクノロジ
ctrPAMon	385960	Comtrue テクノロジ
shdlpMedia.sys	385950	Comtrue テクノロジ
immflex	385910	Immidio B.V.
StegoProtect	385900	Stegosystems Inc.
brfilter.sys	385890	Bromium Inc.
BrCow_x_x_x_x.sys	385889	Bromium Inc.
BemK	385888	Bromium Inc.
secRMM.sys	385880	Squadra テクノロジ
dgfilter	385870	DataGravity Inc.
WFP_MRT.sys	385860	焼討アイ Inc.
klrsps	385815	Kaspersky ラボ
klnsr	385810	Kaspersky ラボ
TaniumRecorderDrv	385800	Tanium
CdsgFsFilter	385700	CRU データセキュリティグループ
mssecflt	385600	マイクロソフト
Backupreader	385500	マイクロソフト
MsixPackagingToolMonitor	385410	マイクロソフト
AppVMon	385400	マイクロソフト
DpmFilter.sys	385300	マイクロソフト

氏名	氏名	COMPANY
Procmon11	385200	マイクロソフト
minispy-Top	385100	マイクロソフト
fdrtrace	385001	マイクロソフト
filetrace .sys	385000	マイクロソフト
uwfreg .sys	384910	マイクロソフト
uwfs .sys	384900	マイクロソフト
locksmith. sys	384800	マイクロソフト
winload .sys	384700	マイクロソフト
SFPMonitor .sys-Top	383350	SonicWall Inc.
FilrDriver .sys	383340	マイクロフォーカス
rwの場合	383330	Rackware
airship-filter	383320	エアウェアテクノロジー Ltd.
AeFilter	383310	Faronics Corporation
QQProtect. sys	383300	Tencent (Shenzhen)
QQProtectX64	383300	Tencent (Shenzhen)
KernelAgent32	383260	ゾーン
WRDWIZFILEPROT.SYS.DATABASES	383251	WardWiz
WRDWIZREGPROT.SYS.DATABASES	383250	WardWiz
groundling32	383200	Dell Secureworks
groundling64	383200	Dell Secureworks
avgtpx86	383190	平均テクノロジー CS-CZ
avgtpx64	383190	平均テクノロジー CS-CZ
DataNow_Driver .sys	383182	AppSense Ltd.
UcaFltDriver	383180	AppSense Ltd.
YFSD2	383170	Yokogawa Corpration

氏名	氏名	COMPANY
Kisknl	383160	王国
MWatcher.sys	383150	Neowiz Corporation
tsifilemon.sys	383140	インターコム Inc.
FIM.sys	383130	elQnetworks Inc.
cFSfdrv	383120	Chaewool
ajfsprot	383110	分析 tik Jena AG
isafermon	383100	40u-cSMS
kfac	383000	北京 CA-JinChen ソフトウェア Co。
GUMHFilter.sys	382910	Glarysoft Ltd.
PsAcFileAccessFilter.sys	382902	富士通のソフトウェア
FJGSDis2	382900	富士通限定
secure_os.sys	382890	富士通のソーシャルサイエンス
ibr2fsk	382880	富士通工学
FJSeparettiFilterRedirect	382860	富士通限定
Fsw31rj1	382855	富士通限定
da_ctl.sys	382850	富士通限定
zqFilter.sys	382800	magrasoft Ltd.
ntps_fa.sys	382700	NTP ソフトウェア
sConnect.sys	382600	I/O データデバイス
AdaptivaClientCache32	382500	Adaptiva
AdaptivaclientCache64	382500	Adaptiva
phantomd	382440	Veramine Inc.
GoFSMF.sys	382430	Gorizonty Rosta Ltd.
SWCommFltr	382420	SnoopWall の場合
atflt	382410	Atlansys ソフトウェア

氏	氏	COMPANY
amfd.sys	382400	Atlansys ソフトウェア
iSafeKrnI	382390	Elex Tech Inc.
iSafeKrnIMon	382380	Elex Tech Inc.
Qutmdrv.sys	382320	360ソフトウェア (北京)
360box	382310	Qihoo 360
360fsflt	382300	北京 Qihoo テクノロジ Co。
scred	382210	ソフトキャンブ Co。
PDGenFam.sys	382200	Soluto LTD.
MCFileMon64 (x64 システム)	382100	Sumitomo 電気
MCFileMon32 (x32 systems)	382100	Sumitomo 電気
RyGuard	382050	SHENZHEN UNNOO Information のテクニカル Co。
FileShareMon	382040	SHENZHEN UNNOO Information のテクニカル Co。
ryfilter	382030	SHENZHEN UNNOO Information のテクニカル Co。
secufile	382020	Shenzhen Unnoo LTD.
XiaobaiFs	382010	Shenzhen Unnoo LTD.
XiaobaiFsR	382000	Shenzhen Unnoo LTD.
TWBDCFilter	381910	Trustwave
VPDrvNt	381900	AhnLab, Inc.
eetd32	381800	Entrust Inc.
eetd64	381800	Entrust Inc.
dnaFSMonitor.sys	381700	Dtex システム
2000の iwhlp2	381610	InfoWatch
XP の iwhlpxp	381610	InfoWatch
Vista の iwhlp	381610	InfoWatch

氏名	氏名	COMPANY
iwdmfs	381600	InfoWatch
IronGateFD	381500	rubysoft
MagicBackupMonitor	381400	マジック Softworks, Inc.
Sonar .sys	381337	IKARUS のセキュリティ
IPFilter	381310	Jinfengshuntai
MSpy	381300	Ladislav Zezula
. sys	381200	3月の h Software Ltd.
qfmon	381190	品質 Corporation
flyfs	381160	NEC ソフト
serfs	381150	NEC ソフト
hdrfs	381140	NEC ソフト
UVMCIFSF	381130	NEC Corporation
ICFClientFlt	381120	NEC システムテクノロジー、Ltd.
IccFileIoAd	381110	NEC システムテクノロジー、Ltd.
IccFilterAudit. sys	381100	NEC システムテクノロジー
IccFilterSc	381090	InfoCage
Sefo. sys-Top	381010	お金
mtsvcdf .sys	381000	境界
SDDrvLdr	380970	の aksander Lebiadzevich
SQLsafeFilterDriver .sys	380901	Idera ソフトウェア
IderaFilterDriver .sys	380900	Idera
cbfsfilter2017	380850	SN Systems Ltd.
xhunter1	380800	Wellbialcom
iGuard	380720	i-ガード SAS
cbflfts4	380715	Nomadesk

氏名	氏名	COMPANY
cbftfs4	380710	バックアップシステム Ltd.
PkgFilter.sys	380700	スケーラブルなソフトウェア Inc.
snimg.sys	380600	Softnext テクノロジ
SK	380520	ヒートソフトウェア
cbfsfilter2017	380515	キット Ltd.
mpxmon	380510	肯定的テクノロジー
filenamevalidator	380502	Infotecs
KC3	380500	Infotecs
PLPOffDrv.sys	380492	SK Infosec 共同
ISFPDrv.sys	380491	SK Infosec 共同
ionmonwdrv	380490	SK Infosec 共同
Sefo.sys-中間	380480	お金
sagntft	380470	ShinNihonSystec Co
VrExpDrv	380460	Hauri Inc.
srminifilterdrv.sys	380450	Citrix システム
zzpensys	380440	Jshan Jing an
tedrdrv.sys	380430	Palo Alto Networks
fangcloud_autolock_driver.sys	380420	Hangzhou Yifangyun
FASDriver	380410	技術研究
CbSampleDrv.sys	380020	マイクロソフト
CbSampleDrv.sys	380010	マイクロソフト
CbSampleDrv.sys	38万	マイクロソフト
簡略化した担当者	371100	マイクロソフト
変更、.sys	370160	マイクロソフト
delete_ft.sys	370150	マイクロソフト

№	№	COMPANY
SmbResilFilter	370140	マイクロソフト
usbtest	370130	マイクロソフト
NameChanger	370120	マイクロソフト
failMount .sys	370110	マイクロソフト
failAttach .sys	370100	マイクロソフト
stest	370090	マイクロソフト
cdo .sys	370080	マイクロソフト
ctx	370070	マイクロソフト
fmm .sys	370060	マイクロソフト
cancelSafe	370050	マイクロソフト
メッセージ .sys	370040	マイクロソフト
パススルー	370030	マイクロソフト
nullFilter .sys	370020	マイクロソフト
ntest. sys	370010	マイクロソフト
minispy- 中間	37万	マイクロソフト
AvaPsFD	368540	Avanite 制限付き
isecureflt	368530	iSecure Ltd.
SFPMonitor .sys- 中央	368520	SonicWall Inc.
wats_se .sys	368510	Fujian Shen
secure_os_mf .sys	368500	HAURI
FileMonitor	368470	Cygna Labs
asiofms .sys	368460	テクノロジーを奨励する
cbfsfilter2017	368450	絶対ソフトウェア
FileHubAgent	368440	SmartFile の場合
pfracdrv	368430	NURILAB

№	№	COMPANY
nrcomgrdki	368420	NURILAB
nrcomgrdka	368420	NURILAB
nrpmonki	368410	NURILAB
nrpq monka. sys	368410	NURILAB
nravwka	368400	NURILAB
bhkavki	368390	NURILAB
bhkavka. sys	368390	NURILAB
docvmonk 者	368380	NURILAB
docvmonk64	368380	NURILAB
InvProtectDrv	368370	Invincea
InvProtectDrv64	368370	Invincea
browserMon. sys	368360	Adtrustmedia
SfdFilter .sys	368350	Sandoll 通信
phdcbtdrv .sys	368340	PHD Virtual Tech Inc.
sysdiag .sys	368330	HeroBravo テクノロジ
WntGPDrv. sys	368327	Winicssec Ltd.
edrdrv .sys	368325	Nurd Yaza.s. m
CmdCwagt	368322	Comodo Security Solutions Inc.
cfrmd. sys	368320	Comodo Security Solutions Inc.
repdrv	368310	ビジョンソリューション
repmon	368300	ビジョンソリューション
cwofflineFlt32	368200	Quantum Corporation。
cwofflineFlt64	368200	Quantum Corporation。
DsDriver .sys	368100	ディスクソフトウェアのワープ
nlcbbelpx86	368000	NetLib

氏名	氏名	COMPANY
nIcbhelpx64	368000	NetLib
nIcbhelpi64	368000	NetLib
wbfilter	367950	ホワイトボックスのセキュリティ
LRAgentMF	367900	LogRhythm Inc.
Drwebfwflt	367810	医師 Web
EventMon. sys	367800	医師 Web
dsfltfs .sys	367760	DigitalSense Co
soidriver .sys	367750	Sophos Plc
drvhookcsmf	367700	GrammaTech, Inc.
drvhookcsmf_amd64 .sys	367700	GrammaTech, Inc.
RevoNetDriver. sys	367650	J のコミュニケーション Co。
avipbb .sys	367600	AGmbH
FileSightMF	367500	PA ファイルの視野
csaam .sys	367400	Cisco システム
FSMon	367300	1mill
filefilter. sys	367100	北京 Zhong Jiaxin Computer テクノロジ Co., Ltd.
iiscache .sys	367000	マイクロソフト
nowonmf	366993	Diskeeper Corporation
dktlfsmf .sys	366992	Diskeeper Corporation
DKDrv .sys	366991	Diskeeper Corporation
DKRtWrt-XPSP3 の temp fix	366990	Diskeeper Corporation
HBFSFltr	366980	Diskeeper Corporation
xoiv8x64	366940	Arcserve
xomfcbt8x64	366930	CA
KmxAgent .sys	366920	CA

氏名	氏名	COMPANY
KmxFile.sys	366910	CA
KmxSbx	366900	CA
PointGuardVistaR32	366810	Futex
PointGuardVistaR64	366810	Futex
Pointguの場合	366800	Futex
PointGuardVista64F	366800	Futex
vintmfs	366789	ConduSivTechnologies
hiofs.sys	366782	ConduSiv テクノロジ
intmfs	366781	ConduSivTechnologies
excfs	366780	ConduSivTechnologies
zampit_ml.sys	366700	Zampit
TenRSafe2	366669	Tencent テクノロジ
tesxporter	366667	Tencent テクノロジ
tesxnginx	366666	Tencent テクノロジ
rflog.sys	366600	AppStream, Inc.
csmon	366582	CyberSight Inc.
mumdi	366540	ZenmuTech Inc.
LivedriveFilter	366500	Livedrive Internet Ltd.
regmonex.sys	366410	Tranxition Corp
TXRegMon	366400	Tranxition Corp
SDVFilter.sys	366300	Soliton Systems K.K.
eLock2FSCTLDriver	366210	これはテクノロジーの Inc. です。
msiodrv4	366200	Centennial Software Ltd.
mmPsy32	366110	Resplendence プロジェクト
mmPsy64	366110	Resplendence プロジェクト

氏名	氏名	COMPANY
rrMon32	366100	Resplendence プロジェクト
rrMon64	366100	Resplendence プロジェクト
cvsflt	366000	3月の h Software Ltd.
ktsyncfsflt	365920	KnowledgeTree Inc.
nvmon	365900	NetVision, Inc.
SnDacs	365810	の場合
SnExequota	365800	の場合
llfilter .sys	365700	SecureAxis ソフトウェア
hafsnc	365660	HA Unix Pt
DgeDriver	365655	Dell Software Inc.
BWFSDrv .sys	365650	Quest ソフトウェア Inc.
CAADFlt	365601	Quest ソフトウェア Inc.
QFAPFlt	365600	Quest ソフトウェア
XendowFLT	365570	Credant テクノロジ
fmdrive	365500	Cigital, Inc.
EGMinFlt	365400	WhiteCell Software Inc.
it2reg	365315	Soliton システム
it2drv	365310	Soliton システム
solitkm	365300	Soliton システム
pgpwdefs .sys	365270	Symantec
GEProtection. sys	365260	Symantec
diflt	365260	Symantec Corp.
sysMon	365250	Symantec
ssrfsf	365210	Symantec
emxdrv2	365200	Symantec

氏	氏	COMPANY
reghook.sys	365150	Symantec
spbbcdrv.sys	365100	Symantec
bhdrv86	365100	Symantec
bhdrv64	365100	Symantec
symevnt.sys	365090	Broadcom
symevnt32	365090	Broadcom
SISIPSEFileFilter	365010	Symantec
symevent	365000	Symantec
wrpfv.sys	364900	マイクロソフト
Upguシヤード	364810	UpGuard
usbl_ifsfltr.sys	364800	SecureAxis
ntfsf	364700	Sun & ムーン
BssAudit.sys	364600	ByStorm
GPMiniFilter フィルター.sys	364500	Kalpataru
AlfaFF	364400	Mg-alfa
FSAFilter.sys	364300	ScriptLogic
GcfFilter.sys	364200	GemacmbH
FFCFILT.SYS.DATABASES	364100	FFC 限定
msnfsflt	364000	マイクロソフト
mblmon	363900	Packeteer
amsfilter	363800	Axur 情報 (Sec)。
rswctrl.sys	363713	Douzone Bizon Co
mcstrg.sys	363712	Douzone Bizon Co
fmkkc	363711	Douzone Bizon Co
nmlhssrv01	363710	Douzone Bizon Co

II	II	COMPANY
equ8_helper .sys	363705	Int3 Software AB
strapvista .sys (廃止)	363700	AvSoft テクノロジ
SAFE-Agent	363636	安全-Cyberdefense
EstPrmon	363610	ESTsoft corp。
Estprp-64 ビット	363610	ESTsoft corp。
EstRegmon	363600	ESTsoft corp。
EstRegp. sys-64 ビット	363600	ESTsoft corp。
agfsmon	363530	TechnoKom Ltd.
NlxFF .sys	363520	OnGuard Systems のシステム
サハラ	363511	Saf の終了
サンタ .sys	363510	Saf の終了
vfdrv .sys	363500	Viewfinfin
topdogfsfilt	363450	ManTech
xhunter64	363400	Wellbia.com
アン未確認	363390	Wellbia.com
AuditFlt	363313	Ionx Solutions p
SPIMiniFilter	363300	ソフトウェアのいいね。
mracdrv .sys	363230	Mail.Ru
BEDaisy	363220	BattlEye イノベーション
MPKernel .sys	363210	Lovelace Network Tech
Net、Ctrl. sys	363200	リンク co。
NetAccCtrl64	363200	リンク co。
bzsenth. sys	363140	BiZone のお金
hpreg	363130	HP
QMON. sys	363122	Qualys Inc.

氏名	氏名	COMPANY
qfimdvr .sys	363120	Qualys Inc.
QDocumentREF. sys	363110	BicDroid Inc.
dsfemon	363100	トポロジ Ltd.
AmznMon	363030	アマゾンウェブサービス Inc.
iothorfs	363020	ioScience
ctamflt	363010	ComTrade
psi? sys	363000	SharpCrafters
QmlInspec. sys	362990	北京 QiAnXin Tech。
GagSecurity	362970	北京 Shu Yan サイエンス
Cybカーネルトラッカー	362960	CyberArk ソフトウェア
filemon	362950	Temasoft S.R.L.
Scalは .sys	362940	Sogou Ltd.。
ep_minifilter .sys	362930	ForcePoint の参照。
klifks	362902	Kaspersky ラボ
klifaa .sys	362901	Kaspersky ラボ
Klifsm	362900	Kaspersky ラボ
スポットライト .sys	362870	Cigent テクノロジ Inc.
nxrmflt	362860	NextLabs
大規模な .sys	362850	PolyLogyx の場合
AALProtect	362840	AlphaAntiLeak
egnfsflt	362830	Egnyte Inc.
RsFlt	362820	Redstor 制限
CentrifyFSF	362810	Centrify Corp
Sefo. sys-Bottom	362800	お金
proggerdriver	362790	WaikatoLink Ltd.

氏名	氏名	COMPANY
SFPMonitor .sys-Bottom	362700	SonicWall Inc.
minispy-Bottom	361000	マイクロソフト

34万-349999: FSFilter の削除の取り消し

氏名	氏名	COMPANY
BSSFIt	346000	ブルーシューズソフトウェアの靴
ThinIO	345900	ThinScale テクノロジ
hmpalert. sys	345800	右へ
nsffkmd64	345700	NetSTAR Inc.
nsffkmd32	345700	NetSTAR Inc.
xbのフィルター (.sys)	345600	Zrxb
ifileguard. sys	345500	I-O DATA DEVICE, INC.
undelex32	345400	Resplendence プロジェクト
undelex64	345400	Resplendence プロジェクト
starmon	345300	Kantowitz Engineering, Inc.
mxRCycle	345200	Avanquest
UdFilter .sys	345100	Diskeeper Corporation
it2prtc	345040	Soliton Systems K.K.
このようになります。	345030	Soliton Systems K.K.
システム	345020	Soliton Systems K.K.
このようになります。	345010	Soliton Systems K.K.
システム	345000	Soliton スマート秒
DCVPr .sys	340700	セキュリティースター GmbH

32万-329998: FSFilter ウイルス対策

氏名	氏名	COMPANY
ReveFltMgr	329350	ウイルス対策

氏名	氏名	COMPANY
ReveProcProtection	329340	ウイルス対策
zwPxeSvr .sys	329330	SecureLink Inc.
ZTE Atom. sys	329320	SecureLink Inc.
wscm. sys	329310	富士通のソーシャルサイエンス
IMFFilter .sys	329300	IObit 情報技術
CSFlt	329290	ConeSecurity Inc.
オシリスの .sys	329240	バイナリ防御システム
ospfile_mini .sys	329230	OKUMA Corp
SoftFilterxxx .sys	329222	WidgetNuri Corp
RansomDefensexxx	329220	WidgetNuri Corp
RanPodFS	329210	システムの管理
ksfsflt	329200	北京の王国
DeepInsFS	329190	Deep 性質
AppCheckD .sys	329180	CheckMAL Inc.
spellmon	329170	SpellSecurity
ホワイトリスト (d)	329160	Meidensha Corp
reアク tor. sys	329150	Reアク Ta Ltd.
SE46Filter	329140	テクノロジーの対応
FileScan	329130	NPcore Ltd.
ECATDriver .sys	329120	ジャパン
pfkrnl	329110	FXSEC LTD.
E鳥フィルター .sys	329100	非表示のリフレックス
EdnemFsFilter	329090	ダコタ州大学
b9kernel	329050	Bit9 Inc.
eeCtrl	329010	symantec

№	№	COMPANY
消しゴム .sys (廃止)	329010	symantec
SRTSP. sys	329000	symantec
SRTSPIT-ia64 システム	329000	symantec
SRTSP64.SYS-x64 システム	329000	symantec
a2ertpx86	328920	Emsi Software GmbH
a2ertpx64	328920	Emsi Software GmbH
a2gffx86-x86	328910	Emsi Software GmbH
a2gffx64-x64	328910	Emsi Software GmbH
a2gffi64-IA64	328910	Emsi Software GmbH
a2acc	328900	Emsi Software GmbH
x64 システムでの a2acc64	328900	Emsi Software GmbH
フライトレコーダー	328850	Malwarebytes Corp.
si32_file .sys	328810	Scargo Inc.
si64_file .sys	328810	Scargo Inc.
mbam .sys	328800	Malwarebytes Corp.
lnvscenter. sys	328780	Lenovo
Enigmafilのドライバー (.sys)	328770	Enigoft Oft
KUBWKSP	328750	Netlor SAS
hcp_kernel_acq .sys	328740	参照アクションポイント
SegiraFlt	328730	Segira の場合
wdocsafe	328722	Cheetah Mobile Inc.
lbprotect	328720	Cheetah Mobile Inc.
eamonm .sys	328700	ESET、 spol. s r.o.
klam	328650	Kaspersky ラボ
MaxProc64	328620	セキュリティで保護されたソフトウェアの最大数

氏名	氏名	COMPANY
MaxProtector。sys	328610	セキュリティで保護されたソフトウェアの最大数
maxcryptmon	328601	セキュリティで保護されたソフトウェアの最大数
SDActMon	328600	セキュリティで保護されたソフトウェアの最大数
TmKmSnsr	328550	Trend Micro Inc.
fileflt	328540	Trend Micro Inc.
TmEsFlt	328530	Trend Micro Inc.
TmEyes. sys	328520	Trend Micro Inc.
tmevtmgr	328510	Trend Micro Inc.
tmpreflt	328500	傾向
vcMFilter .sys	328400	SGRI Co., LTD.
SAFsFilter	328300	ライトスピードシステム Inc.
vsepflt	328200	VMware, Inc.
VFileFilter. sys (名前が変更されました)	328200	VMware, Inc.
sfavflt	328130	テクノロジーの sang
sfavflt	328120	テクノロジーの sang
drivesentryfilterdriver2lite	328100	ドライブエントリ Inc.
WdFilter .sys	328010	マイクロソフト
mpFilter .sys	328000	マイクロソフト
vrSDetri	327801	ETRI
vrSDetrix	327800	ETRI
AhkSvPro .sys	327720	Ahkun。
AhkUsbFW. sys	327710	Ahkun。
AhkAMFlt	327700	Ahkun。
majoradvapi	327680	北京 Majorsec
PSINPROC。SYS.DATABASES	327620	パンダのセキュリティ

氏 名	社 名	COMPANY
PSINFILE。SYS.DATABASES	327610	パンダのセキュリティ
amfsm .sys-Windows XP/2003 x64	327600	パンダのセキュリティ
amm8660-Windows Vista x86	327600	パンダのセキュリティ
amm6460-Windows Vista x64	327600	パンダのセキュリティ
ADSpiderDoc	327550	Digitalonnet
BkavAutoFlt	327542	Bkav Corporation
BkavSdFlt	327540	Bkav Corporation
easyanticheat	327530	EasyAntiCheat ソリューション
5nine. cbt. .sys	327520	5nine ソフトウェア Inc.
caavFltr	327510	コンピューターのアソシエーション
ino_fltr .sys	327500	コンピューターのアソシエーション
SECOne_USB .sys	327426	GRGBanking 機器
SECOne_Proc10 .sys	327424	GRGBanking 機器
SECOne_REG10 .sys	327422	GRGBanking 機器
SECOne_FileMon10 .sys	327420	GRGBanking 機器
WCSDriver	327410	ホワイトクラウドのセキュリティ
360qpesv	327404	360ソフトウェア (北京)
dsark	327402	Qihoo 360
360avflt	327400	Qihoo 360
sciptflt	327334	SECUI Corporation
scifsflt	327333	SECUI Corporation
ANVfsm .sys	327310	Arcdo
CDrRSFlt	327300	Arcdo
mfdriver .sys	327250	Imperva Inc.
EPSMn .sys	327200	中

氏	氏	COMPANY
TxFileFilter. sys	327160	北京 Venus
VTsSysFlt	327150	北京 Venus
TesMon. sys	327130	Tencent
QQSysMonX64	327125	Tencent
QQSysMon. sys	327120	Tencent
TSysCare	327110	Shenzhen Tencent コンピューターシステムの 企業限定
TFsFlt	327100	Shenzhen Tencent コンピューターシステムの 企業限定
avmf. sys	327000	Authentium
BDFileDefend	326916	Baidu (北京)
BDsdKit .sys	326914	Baidu オンラインネットワークテクノロジー (北京) Co。
bd0003	326912	Baidu オンラインネットワークテクノロジー (北京) Co。
Bfilter. sys	326910	Baidu (香港特別行政区) 限定
NeoKerbyFilter	326900	NeoAutus
egnfsflt	326830	Egnyte Inc.
RsFlt	326820	Redstor 制限
trpmnflt	326810	TRAPMINE A.S.
PLGFltr	326800	Paretologic
WrdWizSecure64	326730	WardWiz
wrdwizscanner	326720	WardWiz
AshAvScan .sys	326700	Ashampoo GmbH & · KG
Zyfm	326666	ZhengYong InfoTech LTD.
csaav. sys	326600	Cisco システム
oavfm. sys	326550	HSM IT-サービス GmbH
SegMD	326520	Segurmatica

氏名	氏名	COMPANY
SegMP	326510	Segurmatica
SegF	326500	Segurmatica
eeh	326400	eEye デジタルセキュリティ
eeyehv64	326400	eEye デジタルセキュリティ
CpAvFilter.sys	326311	CodeProof テクノロジ Inc.
CpAvKernel.sys	326310	CodeProof テクノロジ Inc.
NovaShield	326300	テクノロジ、Inc.
SheedAntivirusFilterDriver	326290	SheedSoft Ltd.
bSyirmf	326260	BLACKFORT のセキュリティ
bSysp.sys	326250	BLACKFORT のセキュリティ
bSymfdm	326240	BLACKFORT のセキュリティ
bSywl	326235	BLACKFORT のセキュリティ
bSyrtm.sys	326230	BLACKFORT のセキュリティ
bSyaed.sys	326220	BLACKFORT のセキュリティ
bSyar	326210	BLACKFORT のセキュリティ
BdFileSpy	326200	BullGuard
npxgd	326160	インカインターネット Co。
npxgd64	326160	インカインターネット Co。
tkpl2k	326150	インカインターネット Co。
tkpl2k64	326150	インカインターネット Co。
GKFF.sys	326140	インカインターネット Co。
GKFF64	326140	インカインターネット Co。
tkdac2k	326130	インカインターネット Co。
tkdacxp	326130	インカインターネット Co。
tkdacxp64	326130	インカインターネット Co。

㊦	㊦	COMPANY
tksp2k	326120	インカインターネット Co。
tkspxp	326120	インカインターネット Co。
tkspxp64	326120	インカインターネット Co。
tkfsft	326110	インカインターネット Co, Ltd.
tkfsft64-64 ビット	326110	インカインターネット Co, Ltd.
tkfsavxp-32 ビット	326100	インカインターネット Co, Ltd.
tkfsavxp64-64 ビット	326100	インカインターネット Co, Ltd.
SMDrvNt	326040	AhnLab, Inc.
ATamptNt	326030	AhnLab, Inc.
V3Flt2k	326020	AhnLab, Inc.
V3MifiNt	326010	Ahnlab
V3lft2k	326000	Ahnlab
V3lftmNt (Old name)	326000	Ahnlab
ArfMonNt .sys	325990	Ahnlab
AhnRghLh	325980	Ahnlab
AszFltNt	325970	Ahnlab
OMFltLh .sys	325960	Ahnlab
V3Flu2k	325950	Ahnlab
TfFregNt .sys	325940	AhnLab Inc.
AdcVcsNT	325930	Ahnlab
vcdriv	325820	Greatsoft Corp. Ltd.
vcreg .sys	325810	Greatsoft Corp. Ltd.
vchle. sys	325800	Greatsoft Corp. Ltd.
NxFsMon	325700	Novatix Corporation
AntiLeakFilter	325600	個人開発者 (Soft3304)

氏名	氏名	COMPANY
NanoAVMF	325510	パンダソフトウェア
shldflt	325500	パンダソフトウェア
nprosec.sys	325410	Norman ASA
nregsec.sys	325400	Norman ASA
issregistry.sys	325300	IBM
THFilter.sys	325200	Sybonic Systems Inc.
pervac.sys	325100	PerSystems SA
avgmfx86	325000	AVG Grisoft
avgmfx64	325000	AVG Grisoft
avgmfi64	325000	AVG Grisoft
avgmfrs (廃止)	325000	AVG Grisoft
FortiAptFilter.sys	324930	Fortinet Inc.
fortimon2	324920	Fortinet Inc.
fortirmon.sys	324910	Fortinet Inc.
fortishield.sys	324900	Fortinet Inc.
mscan-rt	324800	SecureBrain Corporation
sysdiag.sys	324600	セキュリティの強化
agentrtm64	324510	WINS CO. 社
rswmon.sys	324500	WINS CO. 社
mwfsmltr	324420	マイクロワールドソフトウェアサービス Pvt. 社.
gtkdrv	324410	GridinSoft の場合
GbpKm.sys	324400	ガス Tecnologia
crnsysm.sys	324310	Coranti Inc.
crncache32	324300	Coranti Inc.
crncache64	324300	Coranti Inc.

№	№	COMPANY
システム	324242	TEHTRI-セキュリティ
drwebfwft	324210	医師 Web
DwShield .sys	324200	医師 Web
DwShield64	324200	医師 Web
IProtect. sys	324150	All Zone Inc.
TvFiltr	324140	All Zone INC.
TvDriver	324130	All Zone INC.
TvSPFiltr	324120	All Zone INC.
TvPtFile .sys	324110	All Zone INC.
TvMFiltr	324100	すべてのゾーン
SophosED	324050	Sophos
SAVOnAccess .sys	324010	Sophos
savonaccess .sys	324000	Sophos
sld	323990	Sophos
OADevice	323900	高さの高さ
pwipf6	323800	PWI, Inc.
EstRkmon. sys	323700	ESTsoft corp。
EstRkr-64 ビット	323700	ESTsoft corp。
dwprot	323610	医師 Web
Spiderg3	323600	医師の Web サイト。
STKrnI64	323500	Verdasys Inc.
UFDFilter	323400	Yoggie
SCFiltr	323300	SecurityCoverage, Inc.
fildds. sys	323200	Filseclab
fsfilter .sys	323100	MastedCode Ltd.

氏名	氏名	COMPANY
fpav_rtp.sys	323000	f 保護
cwdriver.sys	322900	Leith
AYFilter.sys	322810	ESTsoft
Rtw	322800	ESTsoft
RSRtw.sys	322790	ESTsecurity Corp
RSPCRtw	322780	ESTsecurity Corp
HookSys	322700	北京の情報技術 Corporation の制限
snscore.sys	322600	S. N 安全 & ソフトウェア
ssvhook.sys	322500	SecuLution GmbH
strapvista.sys	322400	AvSoft テクノロジ
strapvista64	322400	AvSoft テクノロジ
sascan	322300	SecureAge テクノロジ
savant.sys	322200	Savant Protection, Inc.
VrARnFlt	322161	HAURI
VrBBDFlt	322160	HAURI
vrSDfmx.sys	322153	HAURI
vrSDfmx.sys	322152	HAURI
vrSDam.sys	322151	HAURI
vrSDam.sys	322150	HAURI
VRAPTFLT	322140	HAURI Inc.
VrAptDef	322130	HAURI
VrSdCore.sys	322120	HAURI
VrFsFtM	322110	HAURI
VrFsFtMX.sys (AMD64)	322110	HAURI
vradfil2	322100	HAURI

II	II	COMPANY
fsgk	322000	f-セキュリティ保護
bouncer	321950	CoreTrace Corporation
PCTCore64	321910	PC ツールの Pty。 社.
PCTCore (Old name)	321910	PC ツールの Pty。 社.
ikfilesec	321900	PC ツールの Pty。 社.
ZxFsFilt	321800	オーストラリアのプロジェクト
antispyfilter	321700	C-NetMedia Inc.
PZDrvXP	321600	VisionPower Co., Ltd.
ggc	321510	クイック回復 TechnologiesPvt。 社.
catflt	321500	クイック回復 TechnologiesPvt。 社.
snsrflt	321495	迅速な回復テクノロジ Pvt。 社.
bdsflt	321490	迅速な回復テクノロジ Pvt。 社.
arwflt	321480	迅速な回復テクノロジ Pvt。 社.
csagent .sys	321410	CrowdStrike Ltd.
kmkufit	321400	Komoku Inc.
ntguard. sys	321337	IKARUS のセキュリティ
epdrv .sys	321320	McAfee Inc.
mfencoas	321310	McAfee Inc.
mfehdk. sys	321300	McAfee Inc.
swin .sys	321250	McAfee Inc.
CyvrFsf. sys	321234	Palo Alto Networks
cmdccav .sys	321210	Comodo グループ Inc.
cmdguard	321200	Comodo グループ Inc.
K7Sentry	321100	K7 Computing Private Ltd.
nsminflt	321050	NHN

社名	社名	COMPANY
nsminfl64	321050	NHN
nvcmlft	321000	Norman
dgsafe	320950	王国
issfltr	320900	ISS
hbflft	320840	BitDefender SRL
vlflft	320832	BitDefender SRL
bdsvm .sys	320830	Bitdefender
gzflft	320820	BitDefender SRL
bddevflft	320812	BitDefender SRL
ignis .sys です	320811	BitDefender SRL
AVCKF.SYS.DATABASES	320810	BitDefender SRL
bdfsfltr	320800	Softwin
bdfm sys	320790	Softwin
gemma	320782	BitDefender SRL
Atc	320781	BitDefender SRL
AVC3.SYS.DATABASES	320780	BitDefender SRL
TRUFOS.SYS.DATABASES	320770	BitDefender SRL
aswmonflft	320700	Alwil
kavnsi	320650	AVNOS
HookCentre	320602	G データ
Pk、.sys	320601	G データ
Minilcpt. sys	320600	G データ
acdrv. sys	320520	OnMoon 企業のお客様
tmfsdrv2	320510	Teramind
avgntflft	320500	AGmbH

氏名	氏名	COMPANY
klam	320450	Kaspersky ラボ
klbg	320440	Kaspersky
kldback.sys	320430	Kaspersky
kldlntf.sys	320420	Kaspersky
kldtool.sys	320410	Kaspersky
klif	320401	Kaspersky ラボ
klif	320400	Kaspersky
klam	320350	Kaspersky ラボ
hsm1twhl	320340	Hitachi ソリューション
hssfwhl.sys	320330	Hitachi ソリューション
DeepInsFS	320323	Deep 性質 Ltd。
DeepInsFS	320322	Deep 性質 Ltd。
DeepInsFS	320321	Deep 性質 Ltd。
DeepInsFS	320320	Deep 性質 Ltd。
avfsmn	320310	Anvisoft
lbd.sys	320300	Lavasoft AB
pavdrv	320210	Panzor サイバーセキュリティ
rvmson	320200	CJSC Returnil ソフトウェア
KawachFsMinifilter システム	320160	Sequaretek IT
secur0FSD_x64.sys	320150	knowwheresoft Ltd.
WRAEKernel	320112	Webroot Inc.
WRKrn	320111	Webroot Inc.
WRC0re.sys	320110	Webroot Inc.
ssfm0nm.sys	320100	Webroot Software, Inc.
ODFsFimFilter	320070	Odyssey サイバーセキュリティ

№	№	COMPANY
ODFsFilter	320060	Odyssey サイバーセキュリティ
vk_fsf.sys	320050	AxBx
VirtualAgent.sys	320005	Symantec

30万-309998: FSFilter レプリケーション

№	№	COMPANY
IntelCAS	309100	Intel Corporation
mvfs.sys	309000	IBM Corporation
frxccd	306000	FSLogix Inc.
fsrecord.sys	305000	マイクロソフト
InstMon	304201	Numecent Inc.
StreamingFSD	304200	Numecent Inc.
ubcminifilterdriver.sys	304100	Ullmore Ltd.
.sys	304000	Legato
stfsd	303900	試みテクノロジ
xomf	303800	CA (XOSOFT)
nfid.sys	303700	Neverfail Group Ltd.
sybfilter.sys	303600	Sybase, Inc.
rfsfilter.sys	303500	Evidian
cvmfsj.sys	303400	CommVault Systems, Inc.
iOraFilter	303300	Infonic plc
bkbmfd32 (x86)	303200	BakBone Software, Inc.
bkbmfd64 (x64)	303200	BakBone Software, Inc.
mblyn	303100	Packeteer
AV12NFNT	303000	AhnLab
mDP_win_mini.sys	302900	マクロの影響

II	II	COMPANY
ctxubs	302800	Citrix Systems Inc.
rrepfsf	302700	Rose-Ems Inc.
cbfsfilter2017	301900	スーパー柔軟なソフトウェア
AxFilter .sys	301800	Axcient Inc.
vxfsrep. sys	301700	Symantec
dellcapfd	301600	Dell Inc.
Sptres. sys	301500	Saf の終了
OfficeBackup .sys	301400	お客様のテクノロジー
pcvnfilt	301300	Blue Coat
repdac .sys	301200	NSI
repkap .sys	301100	NSI
repdrv	301000	NSI

28万-289998: FSFilter 連続バックアップ

II	II	COMPANY
File_monitor .sys	289000	Acronis
Klcdp	288900	Kaspersky ラボ
splitinfmon	288800	分割無限大
versamatic	288700	Acertant Tech
Yfilemon. sys	288690	Yarisoft
ibac .sys	288600	Ide alstor,、
fkdriver	288500	Filekeeper
AAFileFilter	288300	Dell Inc.
hyperoo. sys	288400	Hyperoo Ltd.
HyperBacCA	285000	Red Gate Software Ltd.
ZMSFsFltr	284400	Zenith の InfoTech

ID	ID	COMPANY
AlfaSC	284300	Alfa Corporation
hie_ifs.sys	284200	Hie エレクトロニクス, Inc.
AAFs.sys	284100	AppAssure ソフトウェア
defilter.sys (old)	284000	マイクロソフト
aFsvDrv.sys	283100	ITSTATION Inc.
tilana	283000	Tilana Sys
VmDPFilter.sys	282900	マクロの影響
LbFilter	281700	リンク文 S.r.l.
fbsfd.sys	281600	Ferro ソフトウェア
du、emf	281500	S.L.、SPI、
file_tracker.sys	281420	Acronis Inc.
exbackup.sys	281410	Acronis Inc.
afcdp.sys	281400	Acronis Inc.
dcefltr	281300	Cofio Software Ltd.
ipmrsync_mfilter.sys	281200	OpenMars 企業
cascade	281100	JP ソフトウェア
filearchive.sys	281000	コードの分析
syscdp	280900	システム OK AB
dpm ドライバー (x86)	280850	HP
dpnedriver64 (x64)	280850	HP
hpchgflt	280800	HP
VirtFile	280700	Veritas
DeqoCPS.sys	280600	Deqo
LV_Tracker.sys	280500	LiveVault
cpbak.sys	280410	チェックポイントソフトウェア

№	№	COMPANY
tdmonxp	280400	TimeData
nvfr_cpd	280310	Bakbone Software Inc.
nvfr_fdd	280300	Bakbone Software Inc.
Sptbkp	280290	Saf の終了
accessmonitor .sys	280280	Briljant Ekonomisystem

26万-269998: FSFilter コンテンツスクリーナー

№	№	COMPANY
taExeScanner sys	268350	ITSTATION Inc.
GuardFSFlt	268340	ProShield
usbguard. sys	268330	HangZhou 見る Tech
gibepdevflt	268320	グループ-IB LTD.
EffeDriver	268310	「VA」
Klshadow	268300	Kaspersky ラボ
TN28	268290	ID 認証技術
PGDriver .sys	268280	Avecto Ltd.
itseczvdb	268270	Innotium Inc.
isarsd .sys	268260	ISARS
zeoscanner	268255	PCKeeper
fileHiders	268250	PCKeeper
cbftfs4-ObservelT	268240	ObservelT
hipara	268230	Allsum の場合
このようにしてください。	268220	Alibaba
writeGuard. sys	268210	TCXA Ltd.
KKUDKProtectKer	268200	Goldmessage テクノロジ co。
HAWKFIMInt	268190	ホークのネットワーク防御

氏名	氏名	COMPANY
es、ctl	268180	EgoSecure GmbH
WSguard. sys	268170	ワイパー Software UAB
Atomizer	268160	DragonFlyCodeWorks
farwflt	268150	Malwarebytes
ADSpiderEx2	268140	Digitalonnet
sdfilter .sys	268130	Igor Zorkov
Safe	268120	rian@alum.mit.edu
mydlpdelete-scanner	268110	Medra Teknoloji
mydlpscanner. sys	268100	Medra Teknoloji
hnpro .sys	268040	この
DLDriverNetMini .sys	268030	DeviceLock Inc.
ENFFLTDRV	268020	Enforcive システム
imagentpg. sys	268012	インフォ最大
crocopg	268010	インフォ最大
sbapifs	268000	Sunbelt ソフトウェア
H6kernNT	267920	H6N テクノロジーの提供
SGKD32.SYS.DATABASES	267910	NetSection のセキュリティ
lccFilter. sys	267900	NEC システムテクノロジー
tflbc .sys	267800	Tani エレクトロニクス Corporation
ArmFlt	267000	防御力のあるウイルス対策
WBDrv	266700	Axiana の場合
DMSamFilter .sys	266600	Digimarc Corp.
mumbl	266540	ZenmuTech Inc.
5nine. cbt. .sys	266100	5nine ソフトウェア Inc.
bsfs. sys	266000	クイック回復 TechnologiesPvt. 社.

氏名	氏名	COMPANY
XXRegSFilter	265910	Zhe Jiang Xinxin Software Tech.
XXSFilter	265900	Zhe Jiang Xinxin Software Tech.
AloahaUSBBlocker	265800	Wrocklage Intermedia
frxdrv .sys	265700	FSLogix Inc.
FolderSecure .sys	265600	セキュリティで保護されたソフトウェアの最大数
XendowFLTC	265570	Credant テクノロジ
RepDac	265500	ビジョンソリューション
tbbdriver .sys	265400	Tedesì
spcgrd	265300	富士通の幅広いソリューション
fdtlock .sys	265250	富士通の幅広いソリューション & コンサルティング Inc.
ssfFSC .sys	265200	SECUWARE S.L.
GagSecurity	265120	北京 Shu Yan サイエンス
PrintDriver .sys	265110	北京 Shu Yan サイエンス
アクティブな	265100	Rapidware Pty Ltd.
avscan. sys	265010	マイクロソフト
スキャナー .sys	265000	マイクロソフト
DI_fs .sys	264910	ソフト SB
wgnpos. sys	264900	Orchestria
odfltr	264810	NetClean テクノロジ
ncpafldr	264800	NetClean テクノロジ
ct	264700	Haute Secure
fvefsmf	264600	Fortisphere, Inc.
ブロック .sys	264500	自律システム限定
csascr sys	264400	Cisco システム
SymAFR .sys	264300	Symantec Corporation

II	II	COMPANY
cwnep	264200	Websense Inc.
spywareremover	264150	C-Netmedia
malwarebot	264140	C-Netmedia
antispywarebot	264130	2Squared Inc.
adwarebot	264120	スパイウェア対策
スパイウェア対策	264110	スパイウェア対策
spywarebot	264100	C-Netmedia
nomp3	264000	Hamish Speirs (プライベート開発者)
dlfilter.sys	263900	Starfield.html ソフトウェア
sifsp.sys	263800	セキュリティ保護された諸島テクノロジー LTD.
DLFsFlt	263700	中央ツールソフトウェア GmbH
SamKeng	263600	Syvik Co, Ltd.
rml	263500	Logis IT Service GmbH
vfsmfed	263410	Vontu Inc.
vfsmfed	263400	Vontu Inc.
acfilter.sys	263300	Avalere, Inc.
psecfilter	263200	MDI 研究所、Inc.
システムのリダイレクト	263110	Soliton システム
solitkm	263100	Soliton システム
ipcfs	263000	NetVeda
netgateav_access.sys	262910	NETGATE 技術。s.r.o.
spyemrg_access.sys	262900	NETGATE 技術。s.r.o.
pxrmcet	262800	Proxure Inc.
EgisTecFF	262700	これはテクノロジーの Inc. です。
fgcpac	262600	Fortres の Corp Corp.

氏	氏	COMPANY
saappctl.sys	262510	SecureAge テクノロジ
sadlp	262500	SecureAge テクノロジ
CRExecPrev.sys	262410	Cybereason
PEG2	262400	PE ガード
AdminRunFlt	262300	Simon ジャービス
wvscr	262200	Chengdu Wei Tech Inc.
psepfiler	262100	絶対ソフトウェア
SAMDriver.sys	262000	サミット IT
emrcore.sys	261920	Ivanti Inc.
wire_fsfilter.sys	261910	ThreatSpike Labs
AMFileSystemFilter.sys	261900	AppSense Ltd.
mtflt	261880	mTalos Inc.
nxrmflt	261680	NextLabs, Inc.
oc_fsfilter.sys	261300	Raiffeisen Bank Aval
hdlpflt	261200	McAfee Inc.
CCFFilter.sys	261160	マイクロソフト
cbafilt	261150	マイクロソフト
SmbBandwidthLimitFilter	261110	マイクロソフト
DfsrRo	261100	マイクロソフト
DataScrn	261000	マイクロソフト
ldusbro.sys	260900	LANDesk Inc.
FileScreenFilter.sys	260800	Veritas
cpAcOnPnP	260720	conpal GmbH
cpsgfsmf	260710	conpal GmbH
psmmfilter	260700	PolyServe

氏名	氏名	COMPANY
pctefa	260650	PC ツールの Pty。 社.
pctefa64	260650	PC ツールの Pty。 社.
symefasi	260610	Symantec Corporation
symefa	260600	Symantec
symefa64	260600	Symantec
apdFSF sys	260550	Cyberbit Ltd.
aictracedrv_cs .sys	260500	AI コンサルティング
DWFlxxxx .sys	260410	SciencePark Corporation
DWFlxxxx .sys	260400	SciencePark Corporation
DSDriver .sys	260330	ManageEngine Zoho Corp
mcfllab. sys	260320	北京のマイクロ色
FDriver .sys	260310	Fox-IT
iqpk	260300	セキュリティ保護された諸島テクノロジ LTD.
VHDFlt	260240	Dell
VHDFlt	260230	Dell
VHDFlt	260220	Dell
VHDFlt	260210	Dell

24万-249999: FSFilter クォータの管理

氏名	氏名	COMPANY
ntps_qfs .sys	245100	NTP ソフトウェア
PSSFsFilter	245000	PSS システム
Sptqmg	245300	Saf の終了
storqosflt	244000	マイクロソフト

22万-229999: FSFilter システムの回復

II	II	COMPANY
file_protector .sys	227000	Acronis
fbwf .sys	226000	マイクロソフト
Klsysrec	221500	Kaspersky ラボ
SFDRV.SYS.DATABASES	221400	Utixo の場合
sp_prot .sys	221300	Xacti Corporation
nsfilep .sys	221200	Netsupport 制限付き
syscow	221100	システム OK AB
fsredir	221000	マイクロソフト

20万-209999: FSFilter クラスタファイルシステム

II	II	COMPANY
CVCBT .sys	203400	CommVault Systems, Inc.
ResumeKeyFilter	202000	マイクロソフト
VeeamFCT. sys	201900	Veeam ソフトウェア
ShadowVirtualStorage	201800	ブレードの SAS

18万-189999: FSFilter HSM

II	II	COMPANY
wcifs. sys	189900	マイクロソフト
prjflt	189800	マイクロソフト
gameflt	189750	マイクロソフト
nvmsqrd. sys	188900	NVIDIA Corporation
Ghost_file .sys	188800	Acronis
RsFlt	187000	Redstor 制限
mnefs .sys	186800	Nippon Techno Lab
Svfsf	186700	Spハーソフトテクノロジ
uVaultFlt	186650	DOR

氏	氏	COMPANY
syncmf	186620	酸素クラウド
gwmemory.sys	186600	Macrotec の場合
cteraflt.sys	186550	CTERA Networks Ltd.
dbx.sys	186500	Dropbox Inc.
iMDrvFlt	186450	iManage のオンライン
quaddrasi.sys	186400	Quaddra ソフトウェア
gdrive.sys	186300	同社
CoreSyncFilter	186250	Adobe Systems Inc.
EaseTag	186200	EaseVault テクノロジ Inc.
HSFilter	186150	HubStor Inc.
hcmnifilter	186100	すばらしい Cloud Inc.
PDFsFilter.sys	186000	Raxco Sftware Inc.
camino	185900	CaminoSoft Corp
C2C_AF1R。SYS.DATABASES	185810	C2C システム
DFdriver.sys	185800	DataFirst Corporation
amfadv.sys	185700	Quest ソフトウェア Inc.
HSMdriver	185600	Wim Vervoorn
kdfilter	185555	Komprise Inc.
htdafd.sys	185500	ブリッジヘッドソフト
odphflt	180455	マイクロソフト
cldflt	180451	マイクロソフト
SymHsm	185400	Symantec
evmf.sys	185100	Symantec
otfilter.sys	185000	過剰な雰囲気ソフト
ithsmdrv	184900	IBM

№	№	COMPANY
MfaFilter.sys	184800	Waterford テクノロジ
SonyHsmMinifilter	184700	Sony Corporation
acahsm	184600	自律企業
zlhsm.sys	184500	ZL テクノロジ
CFileProtect	184100	Zhejiang セキュリティ技術
stc_restore_filter.sys	184000	StorageCraft テクノロジ
dvfilter.sys	183003	マイクロソフト
システム化	183002	マイクロソフト
Changetracker.sys	183001	マイクロソフト
Fer.sys	183000	マイクロソフト
hsmcdpflt	182700	Metalogix
archivmgr	182690	Metalogix
ntps_oddm.sys	182600	NTP ソフトウェア
XDFileSys	182500	XenData 制限あり
upmjit.sys	182400	Citrix システム
AtmosFS	182310	EMC Corporation
DxSpy.sys	182300	EMC Software Inc.
car_hsmflt.sys	182200	Caringo, Inc.
BRDriver.sys	182100	BitRaider
BRDriver64	182100	BitRaider
autnhsm	182000	自律企業
cthsmflt	181970	ComTrade
NxMini	181900	NEXSAN 製
neuft	181818	NeuShield
npfdafit.sys	181800	Mimosa Systems Inc.

№	№	COMPANY
AppStream .sys	181700	AppStream, Inc.
HPEDpHsmX64	181620	ヒューレットパッカード, Co.
HPArcHsmX64	181610	ヒューレットパッカード, Co.
hphsmflt	181600	ヒューレットパッカード, Co.
cparchsm	181610	マイクロフォーカス
RepHsm .sys	181500	ソフトウェア, Inc. を再利用します。
RepSIS	181490	ソフトウェアの再利用
SquashCompressionFsFilter	181410	スカッシュ圧縮
GXHSM .sys	181400	Commvault Systems, Inc.
EdsiHsm .sys	181300	Enterprise Data Solutions, Inc.
BkfmMap	181200	データストレージグループ
hsmfilter	181100	GRAU Data Storage AG
mwilcflt	181020	Moonwalk Universal P/L
mwildflt	181015	Moonwalk
mwilsflt	181010	Moonwalk Universal P/L
mwidmflt	181000	Moonwalk Universal P/L
HcpAwfs	181960	Hitachi データシステム
sdrefltr	180950	Hitachi データシステム
fltasm	180900	グローバル360
cnet_hsm .sys	180850	Carroll-Net Inc.
pntvolflt	180800	ソフトウェア & システムのポイント
appxstrm	180710	マイクロソフト
wimmount	180700	マイクロソフト
hsmflt	180600	マイクロソフト
dfsrfilt	180500	マイクロソフト

氏名	ID	COMPANY
Storagesyncguard.sys	180465	マイクロソフト
Storagesync.sys	180460	マイクロソフト
重複除去. sys	180450	マイクロソフト
dfmflt	180410	マイクロソフト
sis .sys	180400	マイクロソフト
rbt_wfd .sys	180300	Riverbed テクノロジ、Inc.

17万-174999: * FSFilter Imaging (例:ZIP

氏名	ID	COMPANY
pfmfs_???.sys.databases	172100	Pismo Technic Inc.
virtual_file .sys	172000	Acronis
Wimflt.sys	170500	マイクロソフト

16万-169999: FSFilter 圧縮

氏名	ID	COMPANY
CmgFFC	166000	Credant テクノロジ
.sys を圧縮する	165000	マイクロソフト
cmpflt	162000	マイクロソフト
IridiumIO	161700	# O
logcompressor. sys	161600	VelociSQL Inc.
GcfFilter .sys	161500	GemacmbH
ssddoubler	161400	Sinan Karaca
Sptcmp	161300	Saf の終了
wimfsf	161000	マイクロソフト
GEFCMP .sys	160100	Symantec

14万-149999: FSFilter の暗号化

氏名	氏名	COMPANY
AAFS.sys	149110	ViGero
FJSeparettiFilterRamMon	149100	富士通限定
trsxefs	149060	このようになります。
psatfilter.sys	149050	ProYuga
RdFilter.sys	149040	CyberEye Research Labs
gisfile_decryption.sys	149030	中国の通信
TIFSFilter.sys	149020	SG Corporation
OsrDt2	149010	情報セキュリティ Corp
EasyKryptMF	149000	SoftKrypt の場合
南京錠 (.sys)	148910	IntSoft Inc.
ffecore	148900	Winmagic
bkfs	148880	Hangzhou JoyBlock Ltd.
fangcloud.sys	148860	Hangzhou Yifangyun
klvfs	148810	Kaspersky ラボ
Klfle	148800	Kaspersky ラボ
ISFP.sys	148701	アルプスのシステム統合
ISIRM	148700	アルプスのシステム統合 CO。
ASUSSecDrive.sys	148650	ASUS
ABFilterDriver.sys	148640	AlertBoot
QDocumentFSF	148630	BicDroid Inc.
bfusbenc	148620	bitFence Inc.
sztgbfsf	148610	安全です。
mwIPSDFilter	148600	これはテクノロジの Inc. です。
csccvdrv	148500	Computer サイエンス Corporation
aeefs.sys	148400	Angelltech Corporation Xi'an

№	№	COMPANY
VTEFsFlt	148374	EsComputer Corp
IWCSEFlt	148300	InfoWatch
GDDmk	148250	G データソフトウェア AG
clxcrc .sys	148210	AFORE ソリューション Inc.
OrisLPDrv .sys	148200	CGS 発行技術
nlemsys	148100	NETLIB
prvflr	148000	マイクロソフト
ssefs	147900	SecuLution GmbH
SePSed	147800	Humming Head, Inc.
dImfencx .sys	147700	Data Encryption Ltd.
SkyDEnc	147620	スカイ Co Ltd.
psgcrpt	147610	Yokogawa R & L Corp
bbfsft	147600	Bloombase
qx10efs	147500	Quixxant
MEefs .sys	147400	Eruces Inc.
medlpflt	147310	チェックポイントソフトウェアテクノロジ Ltd.
dsfa .sys	147308	チェックポイントソフトウェアテクノロジ Ltd.
Snicrpt	147300	Systemneeds 必要性, Inc.
iCrypt	147200	I-O DATA DEVICE, INC.
xdrmfilt	147100	bluefinsystems
dyFsFilter .sys	147000	Scrypto メディア
thinairwin	146960	シン・ Inc.
UcaDataMgr	146950	AppSense Ltd.
zesocc	146900	社
mfprom .sys	146800	McAfee Inc.

氏名	氏名	COMPANY
MfeEEFF	146790	McAfee Inc.
整合性 (.sys)	146700	TianYu ソフトウェア
leofs .sys	146600	Leotech
autocryptater sys	146500	Richard Hagen
WavxDMgr	146400	Scott Cochrane
eedmkxp32	146300	Entrust
SbCe	146200	SafeBoot
iSharedFsFilter	146100	Packeteer Inc.
dIrmenc	146010	DESlock
dImfenc	146000	DESlock
aksdf .sys	145900	Aladdin ナレッジシステム
DDSFilter .sys	145800	WuHan Forworld ソフトウェア
SecureShield .sys	145700	HMI
AifaFE	145600	Mg-alfa
GBFsMf .sys	145500	GreenBorder
jmeFs .sys	145400	上海 Elec
emugufs	145333	Emugu Secure FS
VFDriver .sys	145300	R システム
IntelDG	145250	Intel Corporation
システム	145240	Randtronics Pty
EVSDecrypt64	145230	Fortium Technologies Ltd.
skycryptorencfs	145220	Onecryptor CJSC。
AisLeg	145210	情報セキュリティの保証
windtalk	145200	Hyland ソフトウェア
TeamCryptor	145190	iTwin Pte. 社.

№	№	COMPANY
CVDLP. sys	145180	CommVault Systems, Inc.
5nine. 暗号化システム	145170	5nine ソフトウェア Inc.
ctpfile	145160	北京 Wondersoft テクノロジ Co。
システム	145150	IBM 日本
tsdlp. sys	145140	Foreware
KCDriver .sys	145130	Tallegra Ltd.
CmgFFE .sys	145120	Credant テクノロジ
fgcenc	145110	Fortres の Corp Corp.
sview	145100	Cinea
TalkeyFilterDriver	145040	myTALKEY s.r.o.
FedsFilterDriver .sys	145010	物理的な光の Corp
stocc	145000	Senforce テクノロジ
SnEfs	144900	の場合
ewSecureDox	144800	Echoworx Corporation
osrdmk	144700	OSR オープンシステムリソース, Inc.
uldc. sys	144600	NCR 財務ソリューション
Tkefsxp .sys-32 ビット	144500	インカインターネット Co, Ltd.
Tkefsxp64-64 ビット	144500	インカインターネット Co, Ltd.
Nml、.sys	144400	NEC システムテクノロジ、Ltd.
% Crypt	144300	Soliton Systems K.K.
IngDmk	144200	Ingrian Networks, Inc.
llenc	144100	SecureAxis ソフトウェア
SecureData. sys	144030	SecureAge テクノロジ
lockcube. sys	144020	SecureAge テクノロジの Pte Ltd.
sdmedia .sys	144010	SecureAge テクノロジ

№	№	COMPANY
mysdrive	144000	SecureAge テクノロジ
Filの Mor .sys	143900	モバイルアーマー
VSTXEncr	143800	VIA Technologies, Inc.
dgdmk	143700	Verdasys Inc.
shandy	143600	Saf end Ltd.
C2knet	143520	Secuware
C2kdef	143510	Secuware
ssfFS	143500	SECUWARE S.L.
PISRFE	143400	Jilin の IT 共同。
bapfecre	143300	BitArmor システム、Inc.
中 SD	143200	cihosoft
Fcfileio sys	143100	Brainzsquare, Ltd.
cpdrm. sys	143000	Pikewerks
vmfiltr .sys	142900	Vormetric Inc.
Sfntpfdd	142890	Thales CPL
VFSEnc. sys	142811	Symantec
pgpfs .sys	142810	Symantec
システム	142800	Symantec
TmFileEncDmk	142700	Trend Micro Inc.
cpefs .sys	142600	暗号化-Pro
dekfs .sys	142500	KasherLab co., ltd.
qlockfilter .sys	142400	Binqsoft Inc.
RRFilterDriverStack_d3 .sys	142300	合理的に保持
cve. sys	142200	絶対ソフトウェア Corp。
spcflt	142100	富士通 BSC Inc.

№	№	COMPANY
ldsecusb.sys	142000	LANDesk Inc.
fencr.sys	141900	SODATSW spol. s.r.o.
RubiFlt	141800	Hitachi
NCrypt	141700	Nimshi Corp
mfild.sys	141660	Penta セキュリティシステム
cbfsfilter2017	141635	オートマトン Inc.
cbfsfilter2017	141634	オートマトン Inc.
cbfsfilter2017	141633	オートマトン Inc.
cbfsfilter2017	141632	オートマトン Inc.
cbfsfilter2017	141631	オートマトン Inc.
cbfsfilter2017	141630	オートマトン Inc.
TypeSquare	141620	Morisawa inc.
xbdocfilter.sys	141610	Zrxb
EVSDDecrypt32	141600	Fortium Technologies Ltd.
EVSDDecrypt64	141600	Fortium Technologies Ltd.
EVSDDecryptia64	141600	Fortium Technologies Ltd.
SophosDt2	141510	Sophos Plc
afdriber.sys	141500	ATUS テクノロジの方法
TrivalentFSFltr	141430	サイバーに依存
CmdMnEfs.sys	141420	Comodo のセキュリティ
DWENxxxx.sys	141410	SciencePark Corporation
DWENxxxx.sys	141400	SciencePark Corporation
hdFileSentryDrv32	141300	He/g の防御
hdFileSentryDrv64	141300	He/g の防御
pnpps.sys	141250	PNP SECURE INC.

氏名	氏名	COMPANY
SmartCipherFilter	141240	マイクロフォーカス
cplcdt2	141230	conpal GmbH
asCryptoFilter .sys	141220	適用されたセキュリティ GmbH
NetCryptKR	141210	NetCrypt Pty Ltd.
BHFilter	141200	スパソリューション
Filecrypt	141100	マイクロソフト
http.sys の暗号化	141010	マイクロソフト
swapBuffers. sys	141000	マイクロソフト

13万-139999: FSFilter 仮想化

氏名	氏名	COMPANY
Klvirt	138100	Kaspersky ラボ
thsmmf. sys	138060	Talon ストレージソリューション
VMagic .sys	138050	AI コンサルティング
GetSAS .sys	138040	SAS 協会 Inc.
rqtNos	138030	Reアク Ta Ltd.
HIPS64	138020	Recrypt の場合
frxdrv .sys	138010	FSLogix Inc.
vzdrv .sys	138000	Altiris
sffsg .sys	137990	ヒトデ Storage Corp
AppStream .sys	137920	Symantec Corporation
Rasm	137915	OpDesk Inc.
boxifier	137910	Kenubi
xorw .sys	137900	CA (XOsoft)
ctlua	137800	B.V.
fgccow	137700	Fortres の Corp Corp.

氏名	氏名	COMPANY
aswSnx.sys	137600	ALWIL ソフトウェア
ApplsoFtr	137500	カーネルドライバー
ptcvfsd	137400	PTC
BDSandBox.sys	137300	BitDefender SRL
sxfpss-virt	137200	Skanix
DKRtWrt	137100	Diskeeper Corporation
ivm.sys	137000	RingCube テクノロジ
ivm.sys	136990	Citrix システム
dtiof.sys	136900	Instavia Software Inc.
NxTopCP	136800	仮想 Ccomputer Inc.
svdriver.sys	136700	VMware, Inc.
unifltr	136600	Unidesk
unidrive (名前の変更)	136600	Unidesk
unirsd.sys	136600	Unidesk
ive	136500	TrendMicro Inc.
odamf	136450	Sony Corporation
SrMxfMf	136440	Sony Corporation
pszmf.sys	136430	Sony Corporation
sxsudfmf	136410	Sony Corporation
vfammf	136400	Sony Corporation
lwfsflt	136300	Liquidware Labs
VHDFlt	136240	Dell
VHDFlt	136230	Dell
VHDFlt	136220	Dell
VHDFlt	136210	Dell

氏	氏	COMPANY
ncfsflt	136200	NComputing Inc.
cmdguard	136100	COMODO Security Solutions Inc.
hpfsredir	136000	HP
svhdxflt	135100	マイクロソフト
luafr .sys	135000	マイクロソフト
ivm .sys	134000	RingCube テクノロジ
ivm .sys	133990	Citrix システム
frxdrvvt	132700	FSLogix Inc.
pfmfs_???.sys.databases	132600	Pismo Technic Inc.
Stcvhdmf	132600	StorageCraft Tech Corp
appdrv01	132500	保護テクノロジ
virtual_file .sys	132400	Acronis
pdiFsFilter	132300	位 Data Inc.
avgvtx86	132200	平均テクノロジ CS-CZ
avgvtx64	132200	平均テクノロジ CS-CZ
DataNet_Driver .sys	132100	AppSense Ltd.
EgenPage .sys	132000	Eebcdic a, Inc.
unidrive-old	131900	Unidesk
ivm. sys. old	131800	RingCube テクノロジ
XiaobaiFsR	131710	SHENZHEN UNNOO LTD.
XiaobaiFs	131700	SHENZHEN UNNOO LTD.
iotfsflt	131600	IO タービン Inc.
mhpvfs. sys	131500	Wunix の制限
svdriver .sys	131400	SnapVolumes Inc.
Sptvrt	131300	Saf の終了

「	「	COMPANY
rswf のアンチ .sys	131210	Panzor サイバーセキュリティ
aicvirt	131200	AI コンサルティング
sfo .sys	130100	マイクロソフト
DeVolume	13万	マイクロソフト

12万-129999: FSFilter 物理クォータ管理

「	「	COMPANY
クォータ. sys	125000	マイクロソフト
qafilter	124000	Veritas
DroboFlt	123900	データロボット

10万-109999: FSFilter 開いているファイル

「	「	COMPANY
insyncmf	105000	InSync
SPILock8	100900	ソフトウェアのいいね。
Klbackupflt	100800	Kaspersky
repkap	100700	ビジョンソリューション
symrg	100600	Symantec
adsfilter	100500	PolyServ
FMonitor .sys	100490	Safetica

8万-89999: FSFilter セキュリティ言え

「	「	COMPANY
pfclft	88240	PNP SECURE INC.
pegasus	88230	情報セキュリティの保証
RSBDrv .sys	88220	SMTechnology Co。
psprotf	88210	Panzor サイバーセキュリティ

№	№	COMPANY
システム (I)	88100	Randtronics Pty
dsbwnck	88000	Easy Solution Inc.
cbfsfilter2017	87910	オートメーション
cbfsfilter2017	87909	オートマトン Inc.
cbfsfilter2017	87908	オートマトン Inc.
cbfsfilter2017	87907	オートマトン Inc.
cbfsfilter2017	87906	オートマトン Inc.
cbfsfilter2017	87905	オートマトン Inc.
cbfsfilter2017	87904	オートマトン Inc.
cbfsfilter2017	87903	オートマトン Inc.
cbfsfilter2017	87902	オートマトン Inc.
cbfsfilter2017	87901	オートマトン Inc.
cbfsfilter2017	87900	オートマトン Inc.
rsbfsfilter .sys	87800	Corel Corporation
hsm1tfit	87720	Hitachi ソリューション
hssfft. sys	87710	Hitachi ソリューション
acmnfft	87708	Hitachi ソリューション
ACSKFFD .sys	87700	Hitachi ソリューション
MyDLPMF	87600	Comodo グループ Inc.
ScuaRaw	87500	Scua Segurança daçã、o
HDSFilter .sys	87400	NeoAutus オートメーションシステム
ikfsmfft	87300	IronKey Inc.
Klsec	87200	Kaspersky ラボ
XtimUSBsFilterDrv .sys	87190	Dalian CP-SDT
RGFLT_FM .sys	87180	Hauri inc.

II	II	COMPANY
flockflt	87170	Ahranta Inc.
ZdCore .sys	87160	Zends の技術的ソリューション
dcrypt	87150	ReactOS Foundation
hpradeo	87140	Pradeo
SDFSAGDRV。SYS.DATABASES	87130	アルプスのシステム統合 CO。
SDFSDEVFDRV。SYS.DATABASES	87120	アルプスのシステム統合 CO。
SDIFSFDREV。SYS.DATABASES	87110	アルプスのシステム統合 CO。
SDFSFDREV。SYS.DATABASES	87100	アルプスのシステム統合 CO。
CModule .sys	87070	Zhejiang セキュリティ技術
HHRRPH	87010	H + H Software GmbH
HHVolFiltr	87000	H + H Software GmbH
SbieDrv .sys	86900	Sandboxie L. T. d.
assetpro	86800	pyaprotectlcom
dlp	86700	Tellus ソフトウェア
.eps	86600	Lumension のセキュリティ
RapportPG64	86500	Trusteer
amminifilter フィルター sys	86400	AppSense
Snifft	86300	Systemneeds 必要性, Inc.
SecFile .sys	86200	Design Inc. によるセキュリティ保護
philly	86110	triCerat Inc.
reggy	86100	triCerat Inc.
cygfilt	86000	Livegrid 組み込み
事前起動	85900	D3L
csareg .sys	85810	Cisco システム
csaenh	85800	Cisco システム

氏名	氏名	COMPANY
asEpsDrv .sys	85750	ASHINI のコロケーション。
CIDACL	85700	GE 航空 (Digital Systems Germantown)
CVDLP. sys	85610	CommVault Systems, Inc.
QGPEFlt	85600	Quest ソフトウェア
Dre	85500	CA
vracfil2	85400	HAURI
TFsDisk .sys	85300	Teruten
rcMiniDrv	85200	REDGATE CO., LTD.
SnMc5xx	85100	の場合
FSPFltd. sys	85010	Mg-alfa
Aifが p. .sys	85000	Mg-alfa
EsAccCtlFE	84901	EgoSecure GmbH
DpAccCtl. sys	84900	Softbroker GmbH
privman. sys	84800	BeyondTrust
eumntvol	84700	Eugrid Inc.
SoloEncFilter	84600	Soliton システム
sbfilter	84500	UC4 のソフトウェア
cposfw .sys	84450	チェックポイントソフトウェアテクノロジ Ltd.
vsdatant. sys	84400	Zone Labs のお金
SePnet	84350	Humming Head, Inc.
SeT d. .sys	84340	Humming Head, Inc.
SePpld	84330	Humming Head, Inc.
SePfsd	84320	Humming Head, Inc.
SePwld	84310	Humming Head, Inc.
SePprd	84300	Humming Head, Inc.

ID	NAME	COMPANY
InPFIter.sys	84200	Humming Head, Inc.
CProCtrl.sys	84100	暗号化-Pro
spyshelter	84000	Datpol
clpinspprot	83900	情報技術会社 Ltd.
uvmfsflt	83376	NEC Corporation
ProtectIt	82373	テラバイトの Inc.
dguard.sys	82300	Dmitry Varshavsky
NSUSBStorageFilter	82200	NetSupport Ltd.
RMSEFFMV。SYS.DATABASES	82100	CJSC Returnil ソフトウェア
BoksFLAC	82000	Fox テクノロジ
cpAcOnPnP	81910	conpal GmbH
cpsgfsmf	81900	conpal GmbH
ndevsec.sys	81800	Norman ASA
ViewIntus_RTDG.sys	81700	Pentego Technologies Ltd.
BKSandFS	81640	Binklac ワークステーション
エアロック.sys	81630	エアロックデジタル Pty Ltd.
zam.sys	81620	
ANXfsm	81610	Arcdo
CDrSDFlt	81600	Arcdo
crnselfdefence32	81500	Coranti Inc.
crnselfdefence64	81500	Coranti Inc.
zlock_drv.sys	81400	セキュリティ
f101fs	81300	Fortres の Corp Corp.
sysgar	81200	中核データ回復
EmbargoM	81100	ScriptLogic

氏名	氏名	COMPANY
ngssdef	81050	Wontok Inc.
ssb	81041	Wontok Inc.
regflt	81040	Wontok Inc.
fsds2a	81000	Splitstream Ltd.
csacentr. sys	80900	Cisco システム
ScvFLT50	80850	Secuve Ltd.
paritydriver .sys	80800	Bit9, Inc.
nkfsprot	80710	Konneka
nkprot	80700	KONNEKA 情報テクノロジ
acpadlock	80691	IntSoft Co
ksmf .sys	80690	IntSoft Co
http.sys	80680	CrowdStrike
SophosED	80670	Sophos
jazzfile .sys	80660	ジャズネットワーク
SMXFs. sys	80500	OSR

6万-69999: FSFilter のコピー保護

氏名	氏名	COMPANY
d3clock	67000	D3CRYPT3D の場合
cbfltf4	66500	I3D テクノロジ Inc.
CkProcess	66100	KASHU SYSTEM DESIGN INC.
dlnfprot	66000	データの暗号化 Sys
baprtsef	65700	BitArmor システム、Inc.
sxfpss	65600	Skanix
rgasdev	65500	Macrovision
SkyFPDrv	65410	空のコロケーション。

氏名	ID	COMPANY
SkyLWP	65400	空のコロケーション。
SkySDVRF.sys	65390	空のコロケーション。
SnEraser ゴム	65300	の場合
vfilter.sys	65200	RSJ Software GmbH
COGOFIt32	65100	Fortium Technologies Ltd.
COGOFIt64	65100	Fortium Technologies Ltd.
COGOFLTia64	65100	Fortium Technologies Ltd.
削除機能	65000	マイクロソフト
BRDriver.sys	64000	BitRaider の場合
BRDriver64	64000	BitRaider の場合
X7Ex	62500	Exent Technologies Ltd.
LibertyFSF	62300	Bayalink ソリューション共同
axfsdrv2	62100	Axence ソフトウェア Inc.
sds	62000	送信ソフトウェア
TotalSystemAuditor.sys	61600	ANRC のオンライン
MBAMApiary.sys	61500	Malwarebytes Corp.
WA_FSW.sys	61400	Programas Admin Acción y Mejoramiento
ViewIntus_RTAS	61300	Pentego テクノロジ
tffac.sys	61200	東芝 Corporation
tccp.sys	61100	TrusCont Ltd.
KomFS	61000	KOM ネットワーク

4万-49999: 下位の FSFilter

氏名	ID	COMPANY
RMPFileMounter	48000	ManageEngine Zoho
cbfsfilter2017	47400	12d Synergy

氏名	氏名	COMPANY
pfmfs_???sys.databases	47300	Pismo Technic Inc.
DLDriverMiniFlt	47200	DeviceLock Inc.
hsmfltlib	47110	Hitachi ソリューション
hskdlib.sys	47100	Hitachi ソリューション
acmnlb	47090	Hitachi ソリューション
aictracedrv_b.sys	47000	AI コンサルティング
hhdcflt	46900	Seagate テクノロジ
Npsvctrig	46000	マイクロソフト
klvfs	44900	Kaspersky ラボ
Klbackupflt	44890	Kaspersky ラボ
rsfxdrv.sys	41000	マイクロソフト
defilter.sys	40900	マイクロソフト
AppVVemgr	40800	マイクロソフト
wofadk.sys	40730	マイクロソフト
wof.sys	40700	マイクロソフト
fileinfo	40500	マイクロソフト
WinSetupBoot.sys	40400	マイクロソフト

2万-29999: FSFilter システム

[なし]。

ミニフィルター高度要求

2020/04/24 • [Edit Online](#)

フィルター マネージャー モデルを開発したファイル システム ミニフィルター ドライバーには、他のミニフィルターに対しては、ファイル システム スタックに存在する相対位置を定義する高度と呼ばれる一意の識別子が必要です。

ミニフィルターの高度の識別子を要求するには、ASCII テキスト電子メール メッセージに電子メールを送信 fsfcomm@microsoft.com 件名。「ミニフィルターの高度要求」します。電子メールの本文を含める必要があります、下のフィールドと対応する情報。

このフィルターの高度は、指定した連絡先の電子メール アドレスにメールで送信できます。

電子メールを送信することもできます。 fsfcomm@microsoft.com 既存の高度に関連付けられている情報を更新します。

項目	COMMENT
会社名:	
連絡先の電子メール。	提供、長期的な個別の電子メールと、会社の電子メール エイリアス。
製品名:	
製品の URL:	
製品/フィルターの説明:	Microsoft のフィルターの適切な高度の決定を支援する 1 つの段落 概要です。
ファイル名をフィルター処理します。	
フィルターの種類:	これらの値のいずれか:レジストリ、ファイル システムでは、両方
開始の種類をフィルター処理します。	これらの値のいずれか:ブート、システム、Auto、オンデマンド
ロード順序グループのフィルターを要求します。	参照してください、 ファイル システム ミニフィルター割り当てられた高度の 利用可能なロード順序グループ。
要求された高度:	Microsoft は、要求された高度で、高度の可用性に応じて、フィルター ドライバーの機能とは異なる高度を割り当てる権利を留保します。
追加情報:	このフィールドを使用して、Microsoft がこのフィルターを高度を割り当てるときに検討するか、情報があるかどうかを報告します。

割り当ての要求電子メールの本文の例を次に示します。

Hi,

Below is the request information to assign an altitude for our Contoso DataKleen file system minifilter.

Company name: Contoso Ltd.

Contact e-mail: filterdevgroup@contoso.com

Product name: Contoso DataKleen

Product URL: <http://fsfilters.contoso.com>

Product/Filter Description: The Contoso DataKleen filter removes all occurrences of any byte having a value between 128 and 255 during file reads. Our minifilter removes this value since it is not displayable on TTY devices.

Filter filename: ContosoDK.sys

Filter type: FileSystem

Filter start-type: Demand

Requested filter load order group: FSFilter Content Screener

Requested altitude: 268400

Additional information: None

Thanks,

FilterDev

注

- すべてのフィールドに入力する必要があります。
- 処理を高度を割り当てる 2 週間 Microsoft をかかる場合があります。不足している情報は、割り当てに遅れる可能性があります。
- 割り当てられた高度で表示されている高度に最終的に反映される[ファイル システム ミニフィルター割り当てられた高度](#)します。Microsoft のみを更新するこのリスト年 1 回に注意してください。

再解析ポイント タグ要求

2020/04/24 • [Edit Online](#)

これは、再解析ポイントタグを取得するためのメカニズムです。このタグは、必要なファイルシステムフィルタドライバ用です。

再解析ポイントタグ要求

再解析ポイントタグを取得するには、次の情報を Microsoft に送信します。

- 会社名
- 会社の電子メール
- 会社の URL
- 連絡先の電子メール
- 製品名です
- 製品の URL
- 製品の説明: (1 段落の概要)
- ドライバーのファイル名
- ドライバードバイス名
- ドライバー GUID
- 待機時間が長いビットを有効にする (はい/いいえ)
- 名前サロゲートビット有効 (はい/いいえ)

この情報を ASCII テキストの電子メールメッセージで rpids@microsoft.com に送信します。このドライバーの *ReparseID* 値は、指定された連絡先の電子メールアドレスに電子メールで送信されます。

次の一覧では、要求を送信するためのいくつかの要件について説明します。

- すべてのフィールドに入力する必要があります。
- この一般公開されているデータベースで独自の名前と電子メールアドレスを使用したくない場合は、この使用のための電子メールアドレスを作成してください (お客様の相互運用性の問題がある、ファイルシステム/フィルタードライバを含む製品がある他の企業にとって、相互に通信するために2社の企業のテスト/開発 推奨される名前は *"ntfskit@YourCompanyName.com"* です。
- "高待機時間" ビットが有効になっている場合は、ドライバーがファイルにタグを付け、待機時間が長いことが予想されます。たとえば、これは、再解析ポイントを使用して階層型ストレージソリューションを実装するドライバーによって設定されます。
- "Name サロゲート" ビットが有効になっている場合、ドライバーはシステム内の別の名前付きエンティティを表すことを意味します。たとえば、ボリュームマウントポイントの名前、またはディレクトリジャンクションの名前などです。
- 再解析ポイントは Windows の強力な機能ですが、開発者はファイルごとに1つの再解析ポイントしか存在できず、一部の Windows メカニズムでは再解析ポイント (HSM、ネイティブ構造化ストレージ) が使用されていることに注意する必要があります。開発者は、再解析ポイントタグがファイルに対して既に使用されている場合のフォールバックストラテジを持っている必要があります。

ミニフィルタードライバーの DriverEntry ルーチンを記述する

2020/04/24 • [Edit Online](#)

すべてのファイルシステムミニフィルタードライバーに **Driverentry** ルーチンが必要です。Driverentry ルーチンは、ミニフィルタードライバーが読み込まれるときに呼び出されます。

Driverentry ルーチンは、グローバルな初期化を実行し、ミニフィルタードライバーを登録し、フィルター処理を開始します。このルーチンは、IRQL 受動_レベルでシステムスレッドコンテキストで実行されます。

Driverentry ルーチンは次のように定義されています。

```
NTSTATUS
(*PDRIVER_INITIALIZE) (
    IN PDRIVER_OBJECT DriverObject,
    IN PUNICODE_STRING RegistryPath
);
```

Driverentry に2つの入力パラメーターがあります。最初の *Driverobject* は、ミニフィルタードライバーが読み込まれたときに作成されたドライバーオブジェクトです。2番目の *RegistryPath* は、ミニフィルタードライバーのレジストリキーへのパスを含む、カウントされた Unicode 文字列へのポインターです。

ミニフィルタードライバーの Driverentry ルーチンでは、次の手順を順番に実行する必要があります。

1. ミニフィルタードライバーに必要なグローバル初期化を実行します。
2. **Fltregisterfilter** を呼び出してミニフィルタードライバーを登録します。
3. **Fltstartfiltering** を呼び出してフィルター処理を開始します。
4. 適切な NTSTATUS 値を返します。

このセクションの内容は次のとおりです。

[ミニフィルタードライバーを登録しています](#)

[フィルター処理の開始](#)

[ミニフィルタードライバー DriverEntry ルーチンから状態を返す](#)

ミニフィルター ドライバーの登録

2020/04/24 • [Edit Online](#)

すべてのミニフィルタードライバーは、そのDriverentryルーチンからFltregisterfilterを呼び出して、登録されているミニフィルタードライバーのグローバルリストに追加し、フィルターマネージャーに、コールバックルーチンの一覧とその他の情報を提供する必要があります。ドライバー。

MiniSpy サンプルでは、次のコード例に示すようにミニフィルタードライバーが登録されています。

```
NTSTATUS status;  
status = FltRegisterFilter(  
    DriverObject,           //Driver  
    &FilterRegistration,    //Registration  
    &MiniSpyData.FilterHandle); //RetFilter
```

Fltregisterfilterには2つの入力パラメーターがあります。1つ目のドライバーは、ミニフィルタードライバーがdriverobjectルーチンにdriverobject入力パラメーターとして受け取ったドライバーオブジェクトポインターです。2番目の登録は、ミニフィルタードライバーのコールバックルーチンへのエントリポイントを含むFLT_登録構造体へのポインターです。

さらに、Fltregisterfilterには、ミニフィルタードライバーの非透過的フィルターポインターを受け取る出力パラメーター retfilterがあります。このフィルターポインターは、FltstartfilteringやFltstartfilteringなど、多くのFlt_Xxx_サポートルーチンに必要な入力パラメーターです。

フィルタリングの開始

2020/04/24 • [Edit Online](#)

Fltregisterfilterを呼び出した後、ミニフィルタードライバーの**driverentry**ルーチンは通常**fltregisterfilter**を呼び出して、i/o 操作のフィルター処理を開始します。

すべてのミニフィルタードライバーは、**Driverentry**ルーチンから**Fltstartfiltering**を呼び出して、フィルタマネージャーに対して、ミニフィルタードライバーがボリュームへのアタッチと i/o 要求のフィルター処理を開始する準備ができていることを通知する必要があります。ミニフィルタドライバが**Fltstartfiltering**を呼び出した後、フィルタマネージャはミニフィルタドライバを完全にアクティブなミニフィルタードライバとして扱い、i/o 要求およびアタッチするボリュームの通知を表示します。**Fltstartfiltering**が返される前に、ミニフィルタードライバーは、これらの i/o 要求と通知の受信を開始できるように準備する必要があります。

MiniSpy サンプルドライバーでは、次のコード例に示すように**Fltstartfiltering**が呼び出されます。

```
status = FltStartFiltering( MiniSpyData.FilterHandle );
if( !NT_SUCCESS( status )) {
    FltUnregisterFilter( MiniSpyData.FilterHandle );
}
```

Fltstartfilteringを呼び出すとステータス_SUCCESS が返されない場合、ミニフィルタードライバーは**Fltstartfiltering**を呼び出して自身の登録を解除する必要があります。

ミニフィルター DriverEntry ルーチンから返される状態

2020/04/24 • [Edit Online](#)

ミニフィルタードライバのDriverEntryルーチンは、正常に完了した状態_返します。ただし、ミニフィルターの初期化が失敗した場合は、DriverEntryルーチンから適切なエラー NTSTATUS 値が返されます。

DriverEntryルーチンが、成功した NTSTATUS 値ではない状態値を返した場合、システムはミニフィルタードライバをアンロードすることによって応答します。ミニフィルタードライバのFilterUnloadCallbackルーチンは呼び出されません。このため、DriverEntryルーチンは、システムリソースに割り当てられたメモリを解放してから、成功した NTSTATUS 値ではない状態値を返す必要があります。

FilterUnloadCallback ルーチンの登録

2020/04/24 • [Edit Online](#)

ファイルシステムミニフィルタードライバーでは、必要に応じて `PFLT_FILTER_UNLOAD_CALLBACK` 型のルーチンをミニフィルタードライバーの `FilterUnloadCallback` ルーチンとして登録することもできます。このコールバックルーチンは、ミニフィルタードライバーのアンロードルーチンとも呼ばれます。

ミニフィルタードライバーは、`FilterUnloadCallback` ルーチンを登録するためには必要ありません。ただし、ミニフィルタードライバーでは、`FilterUnloadCallback` ルーチンが登録されていない場合、ドライバーをアンロードできないため、このコールバックルーチンを登録することを強くお勧めします。

このコールバックルーチンを登録するために、ミニパスドライバーは、`PFLT_FILTER_UNLOAD_CALLBACK` で型指定されたルーチンのアドレスを、`FLT_REGISTRATION` 構造体の `FilterUnloadCallback` メンバーに格納します。このメンバーは、ミニフィルタードライバーが、その `Driverentry` ルーチンの `fltregisterfilter` にパラメーターとして渡します。

FilterUnloadCallback ルーチンの呼び出し時期

2020/04/24 • [Edit Online](#)

フィルタマネージャは、次のいずれかの方法でミニフィルタドライバをアンロードする前に、ミニフィルタドライバの **FilterUnloadCallback** ルーチンを呼び出します。

- **必須ではないアンロード**。この種類のアンロードは、ユーザーモードアプリケーションが **Filterunload** を呼び出した場合、またはカーネルモードドライバが **Fltunloadfilter** を呼び出した場合に発生します。また、コマンドプロンプトで「`fltmc unload`」と入力した場合にも発生します。
- **必須のアンロード**。この種類のアンロードは、コマンドプロンプトで「`sc stop`」または「`net stop`」と入力して、サービス停止要求を発行したときに発生します。(`Sc stop` コマンドと `net stop` コマンドの詳細については、スタートメニューの **ヘルプとサポート** をクリックしてください)。また、ユーザーモードアプリケーションが `Microsoft Win32 Controlservice` 関数を呼び出して、サービス 制御、`dwcontrol` パラメーターとしてコントロールコードを停止した場合にも発生します。(Win32 サービス関数の詳細については、Microsoft Windows SDK のドキュメントを参照してください)。

必須ではないアンロードの場合、ミニフィルタードライバの **FilterUnloadCallback** ルーチンによってエラーまたは警告の NTSTATUS 値が返された場合 (STATUS_FLT_DO_、DETACH など)、フィルタマネージャはミニフィルタをアンロードしません。driver.

必須のアンロードの場合、フィルタマネージャは、ミニフィルタードライバの **FilterUnloadCallback** ルーチンが呼び出された後にミニフィルタドライバをアンロードします。たとえば、**FilterUnloadCallback** ルーチンがエラーまたは警告の NTSTATUS 値を返した場合でも、ステータス_FLT_DO_ デタッチされません。

ミニフィルタードライバの必須のアンロードを無効にするには、ミニフィルタードライバによって、FLT の **Flags** メンバーで `_SERVICE_STOP` フラグがサポートされていない、FLTFL_登録_設定され、フィルタドライバが `driverentry` ルーチン内の **fltregisterfilter** にパラメーターとして渡す登録構造。このフラグが設定されている場合、フィルタマネージャは通常、必須ではないアンロード要求を処理します。ただし、必須のアンロード要求は失敗します。フィルタマネージャは、失敗したアンロード要求に対してミニフィルタードライバの **FilterUnloadCallback** ルーチンを呼び出しません。

ミニフィルタードライバの **Driverentry** ルーチンから警告またはエラーの NTSTATUS 値が返された場合、**FilterUnloadCallback** ルーチンは呼び出されないことに注意してください。フィルタマネージャは、ミニフィルタードライバを単にアンロードします。

FilterUnloadCallback ルーチンは、システムのシャットダウン時に呼び出されません。シャットダウン処理を実行する必要があるミニフィルタードライバは、IRP_MJ_のシャットダウン操作に対して `preoperation` コールバックルーチンを登録する必要があります。

FilterUnloadCallback ルーチンを記述する

2020/04/24 • [Edit Online](#)

FilterUnloadCallback ルーチンは次のように定義されています。

```
typedef NTSTATUS
(*PFLT_FILTER_UNLOAD_CALLBACK) (
    FLT_FILTER_UNLOAD_FLAGS Flags
);
```

FilterUnloadCallback ルーチンには1つの入力パラメーターがあります。*Flags*は**NULL**または FLTFL_フィルター_アンロード_必須です。フィルターマネージャーは、アンロード操作が必須であることを示すために、このパラメーターを FLTFL_FILTER_UNLOAD_必須に設定します。このパラメーターの詳細については、「[PFLT_FILTER_UNLOAD_CALLBACK](#)」を参照してください。

ミニフィルタードライバの *FilterUnloadCallback* ルーチンでは、次の手順を実行する必要があります。

- 開いているカーネルモード通信サーバーのポートハンドルを閉じます。
- [Fltunregisterfilter](#) を呼び出してミニフィルタードライバの登録を解除します。
- 必要なグローバルクリーンアップを実行します。
- 適切な NTSTATUS 値を返します。

このセクションの内容は次のとおりです。

[通信サーバーのポートを閉じています](#)

[ミニフィルターの登録解除](#)

[グローバルクリーンアップを実行しています](#)

[FilterUnloadCallback ルーチンから状態を返す](#)

通信サーバー ポートを閉じる

2020/04/24 • [Edit Online](#)

ミニフィルタードライバーで**FltCreateCommunicationPort**を呼び出してカーネルモード通信サーバーのポートを既に開いている場合は、**FltCloseCommunicationPort**を呼び出してポートを閉じる必要があります。アンロードプロセス中にシステムがハングするのを防ぐため、ミニフィルタードライバーの**FilterUnloadCallback**ルーチンは、**Fltunregisterfilter**を呼び出す前にこのポートを閉じる必要があります。

ユーザーモードアプリケーションが通信サーバーポートへの接続を開いている場合、**FltCloseCommunicationPort**が返すと、その接続のすべてのクライアントポートは開かれたままになります。ただし、フィルタマネージャは、ミニフィルタドライバがアンロードされたときに、すべてのクライアントポートを閉じます。

ミニフィルターの登録解除

2020/04/24 • [Edit Online](#)

ミニフィルタードライバーの`FilterUnloadCallback`ルーチンは、ミニフィルタードライバーの登録を解除するために`Fltunregisterfilter`を呼び出す必要があります。`Fltunregisterfilter`を呼び出すと、次の処理が行われます。

- ミニフィルタードライバーのコールバックルーチンが登録解除されます。
- ミニフィルタードライバーのインスタンスが破棄され、ミニフィルタードライバーの`InstanceTeardownStartCallback`ルーチンと`InstanceTeardownCompleteCallback`ルーチンが各ミニフィルタードライバーインスタンスに対して呼び出されます。
- ミニフィルタードライバーによって、ボリューム、インスタンス、ストリーム、またはストリームハンドルにコンテキストが設定されている場合、これらのコンテキストは削除されます。ミニフィルタードライバーによって特定のコンテキスト型に対して`cleanupcontext`コールバックルーチンが登録されている場合、フィルターマネージャーはコンテキストを削除する前に`cleanupcontext`ルーチンを呼び出します。

ミニフィルタードライバーの不透明なフィルターポインターに未処理のランダウン参照がある場合、`Fltunregisterfilter`は削除されるまで待機状態になります。未処理ランダウン参照は、通常、ミニフィルタードライバーが`Fltqueuegenericworkitem`を呼び出して、作業項目をシステム作業キューに挿入し、作業項目がまだデキューされて処理されていないことが原因で発生します。(フィルターマネージャーは、ミニフィルタードライバーが`Fltqueuegenericworkitem`を呼び出したときにランダウン参照を追加し、ミニフィルタードライバーの作業ルーチンから制御が戻ったときに、この参照を削除します)。

また、`Fltobjectreference`や`FltGetFilterFromInstance`などの、ミニフィルタードライバーの非透過フィルターポインターにランダウン参照を追加するルーチンをミニフィルタードライバーが呼び出した場合にも、未処理のランダウン参照が発生する可能性があります。後で`Fltobjectdereference` [参照](#)を呼び出すことはありません。

グローバルクリーンアップを実行しています

2020/04/24 • [Edit Online](#)

ミニフィルタードライバーの`FilterUnloadCallback`ルーチンでは、必要なグローバルクリーンアップを実行する必要があります。次の一覧に、ミニフィルタードライバーによって実行される可能性のあるグローバルクリーンアップタスクの例を示します。

- `ExDeleteResourceLite`を呼び出して、以前の`Ex Eresourcelite`の呼び出しで初期化されたグローバルリソース変数を削除します。
- `Exfreeatepoolwithtag`などのルーチンの前の呼び出しで割り当てられたグローバルメモリを解放するには、`exfreepool`または`exfreepoolwithtag`を呼び出します。
- `ExDeleteNPagedLookasideList`または`ExDeletePagedLookasideList`を呼び出して、`ExInitializeNPagedLookasideList`または`ExInitializePagedLookasideList`の前の呼び出しで割り当てられたルックアサイドリストを削除します。4.3.
- `PsRemoveCreateThreadNotifyRoutine`または`PsRemoveLoadImageNotifyRoutine`を呼び出して、`Pssetcreatethreadnotifyroutine`への以前の呼び出しで登録されたグローバルコールバックルーチンの登録を解除します。または、`PsSetLoadImageNotifyRoutine`。

FilterUnloadCallback ルーチンから返される状態

2020/04/24 • [Edit Online](#)

ミニフィルタードライバーのFilterUnloadCallbackルーチンは、通常、正常_状態を返します。

必須ではないアンロード操作を拒否するには、ミニフィルタードライバーは、状態_FLT などの適切な警告またはエラーの NTSTATUS 値を返す必要があります_DO_切断_します。必須のアンロード操作の詳細については、「[FilterUnloadCallback ルーチンを記述する](#)」および「[PFLT_FILTER_アンロード_コールバック](#)」を参照してください。

FilterUnloadCallbackルーチンが警告またはエラーの NTSTATUS 値を返し、アンロード操作が必須ではない場合、ミニフィルタードライバーはアンロードされません。

操作前と操作後のコールバック ルーチンの記述

2020/04/24 • [Edit Online](#)

Driverentryルーチンでは、ミニフィルタードライバーは、フィルター処理に必要な i/o 操作の種類ごとに1つの[preoperation コールバックルーチン](#)と、最大で1つの[postoperation コールバックルーチン](#)を登録できます。

従来のファイルシステムフィルタードライバーとは異なり、ミニフィルタードライバーでは、フィルター処理する i/o 操作の種類を選択できます。ミニフィルタードライバーは、postoperation コールバックを登録せずに、特定の種類の i/o 操作に対して preoperation コールバックルーチンを登録できます。逆の場合も同様です。ミニフィルタードライバーは、preoperation または postoperation コールバックルーチンが登録されている i/o 操作だけを受信します。

Preoperation コールバックルーチンは、レガシフィルタードライバーモデルのデイスパッチルーチンに似ています。フィルターマネージャーは、i/o 操作を処理するときに、この種類の i/o 操作のために登録されているミニフィルタードライバーインスタンススタック内の、各ミニフィルタードライバーの preoperation コールバックルーチンを呼び出します。スタック内の最上位のミニフィルタードライバー (つまり、インスタンスの上位レベルが最高の場合) は、最初に操作を受信します。ミニフィルタードライバーは、操作の処理を終了すると、操作をフィルターマネージャーに返します。これにより、操作が次に最も大きいミニフィルタードライバーに渡されます。ミニフィルタードライバーインスタンススタック内のすべてのミニフィルタードライバーで i/o 操作が処理された場合--フィルターマネージャーは、その操作をレガシフィルタードライバーとファイルシステムに送信します (ただし、フィルターマネージャーによって処理されます)。

Postoperation コールバックルーチンは、レガシフィルタードライバーモデルの完了ルーチンに似ています。i/o 操作の完了処理は、i/o マネージャーが操作をファイルシステムに渡したときと、その操作の完了ルーチンが登録されているレガシフィルタードライバーによって開始されます。これらの完了ルーチンが完了すると、フィルターマネージャーは操作の完了処理を実行します。フィルターマネージャーは、この種類の i/o 操作用に登録されているミニフィルタードライバーインスタンススタックで、各ミニフィルタードライバーの postoperation コールバックルーチンを呼び出します。スタック内の一番下のミニフィルタードライバー (つまり、インスタンスが最も低い標高を持つもの) は、最初に操作を受信します。このミニフィルタードライバーは、操作の処理を終了すると、それをフィルターマネージャーに返します。これにより、操作が次に最も低いミニフィルタードライバーに渡されます。

このセクションの内容:

[Preoperation および Postoperation コールバックルーチンを登録しています](#)

[ミニフィルタードライバーでの i/o 操作のフィルター処理](#)

[プリ操作コールバックルーチンの書き込み](#)

[Postoperation コールバックルーチンを書き込んでいます](#)

[I/o 操作のパラメーターの変更](#)

[I/o 操作のバッファリング方法の決定](#)

[I/o 操作のユーザーバッファへのアクセス](#)

操作前と操作後のコールバック ルーチンの登録

2020/04/24 • [Edit Online](#)

Preoperation コールバックルーチンと**postoperation コールバックルーチン**を登録するために、ミニフィルタードライバーは、**Driverentry**ルーチンで**fltregisterfilter**を1回呼び出します。Fltregisterfilterでの登録パラメーターの場合、ミニパスドライバーは、**FLT_登録**構造体へのポインターを渡します。この構造体のOperationregistrationメンバーには、**FLT__操作**の配列へのポインターが含まれています。これは、ミニフィルタードライバーがフィルター処理する必要がある i/o 操作の種類ごとに1つです。

配列の各 FLT_操作_登録構造には、最後の構造体を除き、次の情報が含まれています。

- 操作の主要な関数コード。I/O 操作の詳細については、「**FLT_PARAMETERS**」を、要求の種類に固有のパラメーターを参照してください。
- 読み取りと書き込みの操作 (IRP_MJ_READ および IRP_MJ_WRITE) の場合、キャッシュされた i/o またはページング i/o を無視するか、両方とも IRP ベースの i/o 操作を行うかを指定するフラグのセット
- 最大1つの preoperation コールバックルーチンと1つの postoperation コールバックルーチンのエントリポイント

配列の最後の要素は、{IRP_MJ_OPERATION_END} である必要があります。

スキャナーのサンプルミニフィルタードライバーから取得した次のコード例では、登録構造 FLT_操作の配列が示されています。スキャナーのサンプルミニフィルタードライバーは、irp_MJ の preoperation および postoperation コールバックルーチンを登録し、IRP_MJ_CLEANUP および IRP_MJ_書き込み操作のための_作成および事前操作コールバックルーチンを登録します。

```
const FLT_OPERATION_REGISTRATION Callbacks[] = {
    {IRP_MJ_CREATE,
      0,
      ScannerPreCreate,
      ScannerPostCreate},
    {IRP_MJ_CLEANUP,
      0,
      ScannerPreCleanup,
      NULL},
    {IRP_MJ_WRITE,
      0,
      ScannerPreWrite,
      NULL},
    {IRP_MJ_OPERATION_END}
};
```

ミニフィルター ドライバーでの I/O 操作のフィルタリング

2020/04/24 • [Edit Online](#)

次の一覧では、ファイルシステムミニフィルタードライバーで特定の種類の i/o 操作をフィルター処理するためのガイドラインについて説明します。

- IRP_MJ_CREATE の **preoperation コールバックルーチン** は、ファイル、ストリーム、またはストリームハンドルのコンテキストを照会または設定できません。これは、作成されるファイルまたはストリーム (存在する場合) がまだ決定されていないためです。
- IRP_MJ_CLOSE の **postoperation コールバックルーチン** は、ファイル、ストリーム、またはストリームハンドルのコンテキストを設定または照会できません。これらの項目に関連付けられているシステム内部構造体は、ポストクローズの前に解放されるためです。ルーチンが呼び出されます。
- ミニフィルタードライバーは、IRP_MJ_CLEANUP または IRP_MJ_CLOSE 操作に失敗しないようにする必要があります。これらの操作は、保留にするか、フィルターマネージャーに返すことができます。または、STATUS_SUCCESS で完了します。ただし、preoperation コールバックルーチンは、これらの操作を失敗させないようにする必要があります。
- ミニフィルタードライバーは、IRP_MJ_シャットダウンの postoperation コールバックルーチンを登録できません。

操作前コールバック ルーチンの記述

2020/04/24 • [Edit Online](#)

ファイルシステムミニフィルタードライバは、1つまたは複数の *preoperation* コールバックルーチンを使用して、i/o 操作をフィルター処理します。**Preoperation コールバックルーチン**は、従来のファイルシステムフィルタードライバで使用するディスパッチルーチンに似ています。

ミニフィルタードライバは、**FLT_REGISTRATION** 構造体の `operationregistration` メンバーにコールバックルーチンのエントリポイントを格納することによって、特定の種類の i/o 操作に対して *preoperation* コールバックルーチンを登録します。ミニパスドライバは、このメンバーを `Driverentry` ルーチンの **Fltregisterfilter** にパラメーターとして渡します。

ミニフィルタードライバは、*preoperation* または *postoperation* コールバックルーチンが登録されている種類の i/o 操作だけを受け取ります。ミニフィルタードライバは、**postoperation コールバックルーチン**を登録せずに、特定の種類の i/o 操作に対して *preoperation* コールバックルーチンを登録できます。また、その逆も可能です。

次の表は、特定の使用シナリオとその戻り値に対する *preoperation* コールバックルーチンの実装を示しています。

説明	動作	戻り値
このルーチンは操作に関連しないため、操作の最終ステータスを必要としないか、 <i>postoperation</i> コールバックがありません。	完了時にミニフィルターの <i>postoperation</i> コールバックを呼び出さずに、i/o 操作をパススルーします。	FLT_PREOP_SUCCESS_コールバックなし
ルーチンには、操作の最終状態が必要です。	操作を通過し、ミニパスコールバックルーチンを呼び出す必要があります。	_コールバックを使用した FLT_PREOP_成功_
ミニフィルターを完了するか、今後この操作の処理を続行する必要があります。	操作を保留中の状態にします。後で操作を完了するには、 Fltcompletependingpreoperation を使用します。FLT_PREOP_PENDING を返す事前操作ルーチンと Fltcompletependingoperation が呼び出される間に、許容される競合が発生する可能性があることに注意してください。フィルターマネージャーは、ドライバからの入力なしにこのシナリオを処理します。	FLT_PREOP_保留中
<i>Postoperation</i> の処理は、ディスパッチルーチンが呼び出されたのと同じスレッドのコンテキストで行われる必要があります。これにより、一貫性のある IRQL が確保され、ローカル変数の状態が維持されます。	操作を <i>postoperation</i> と同期します。	FLT_PREOP_SYNCHRONIZE
<i>Preoperation</i> コールバックルーチンは、操作を完了する必要があります。	操作の処理を停止し、最後の NTSTATUS 値を割り当てます。	FLT_PREOP_完了

すべての *preoperation* コールバックルーチンは、次のように定義されます。

```
typedef FLT_PREOP_CALLBACK_STATUS
(*PFLT_PRE_OPERATION_CALLBACK) (
    IN OUT PFLT_CALLBACK_DATA Data,
    IN PCFLT_RELATED_OBJECTS FltObjects,
    OUT PVOID *CompletionContext
);
```

ディスパッチルーチンと同様に、preoperation コールバックルーチンは、IRQL = パッシブ_レベル、または IRQL = APC_レベルで呼び出すことができます。通常、これは、i/o 要求を発信したスレッドのコンテキストで、IRQL = パッシブ_レベルで呼び出されます。高速 i/o およびファイルシステムフィルター (FsFilter) 操作の場合、preoperation コールバックルーチンは常に IRQL = パッシブ_レベルで呼び出されます。ただし、IRP ベースの操作では、より高いフィルターまたはミニフィルタードライバによってワーカースレッドによる処理が pend された場合、システムワーカースレッドのコンテキストでミニフィルタードライバの preoperation コールバックルーチンを呼び出すことができます。

IRQL > APC_レベルで、postoperation ルーチンでコンテキストオブジェクトを取得することはできません。代わりに、preoperation ルーチンでコンテキストオブジェクトを取得し、それを postoperation ルーチンに渡すか、または IRQL ≤ APC_LEVEL で postoperation 処理を実行します。コンテキストの詳細については、「[コンテキストの管理](#)」を参照してください。

フィルターマネージャーが、特定の i/o 操作に対してミニフィルタードライバの preoperation コールバックルーチンを呼び出すと、ミニフィルタードライバによって i/o 操作が一時的に制御されます。ミニフィルタードライバは、次のいずれかの操作を実行するまで、このコントロールを保持します。

- Preop コールバックルーチンから PENDING_PREOP_PENDING 以外の状態値を返します。
- Preoperation コールバックルーチンで保留されていた操作を処理した作業ルーチンから [FltCompletePendingPreoperation](#) を呼び出します。

このセクションの内容:

[I/O 操作をミニフィルターインスタンススタックに渡す](#)

[Preoperation コールバックルーチンでの i/o 操作の完了](#)

[Preoperation コールバックルーチンで高速 i/o 操作を禁止する](#)

[プリ操作コールバックルーチンで i/o 操作を保留しています](#)

I/O 操作をミニフィルタードライバインスタンススタックに渡す

2020/04/24 • [Edit Online](#)

ミニフィルタードライバの **preoperation コールバックルーチン** または作業ルーチンがフィルターマネージャーに i/o 操作を返すと、フィルターマネージャーは、ミニフィルタードライバインスタンススタック内の現在のミニフィルタードライバの下にあるミニフィルタードライバに操作を送信します。

ミニフィルタードライバの preoperation コールバックルーチンは、次のいずれかの状態値を返すことによって、後続の処理のために i/o 操作をフィルターマネージャーに返します。

- FLT_PREOP_SUCCESS__コールバックなし (すべての操作の種類)
- FLT_PREOP_成功_と_コールバック (すべての操作の種類)
- FLT_PREOP_SYNCHRONIZE (IRP ベースの i/o 操作のみ)

注 _preop_SYNCHRONIZE は、IRP ベースの i/o 操作に対してのみ返される必要がありますが、他の操作の種類については、この状態の値を返すことができます。IRP ベースの i/o 操作ではない i/o 操作に対して返された場合、フィルターマネージャーは、この戻り値を_コールバックと共に_PREOP_成功_として扱います。

別の方法として、preoperation コールバックルーチンで保留されていた操作の作業ルーチンが、**Fltcompletependingpreoperation** を呼び出して、保留中の i/o 操作の処理を再開するときに、その前の状態値の1つをコールバックステータスパラメーターに渡すことによって、フィルターマネージャーに i/o 操作を返します。

このセクションの内容:

[_コールバックで FLT_PREOP_SUCCESS_を返す](#)

[FLT_PREOP_SUCCESS__コールバックなしで返す](#)

[FLT_PREOP_SYNCHRONIZE を返す](#)

_コールバックで FLT_PREOP_SUCCESS_を返す

2020/04/24 • [Edit Online](#)

ミニフィルタードライバーの

```
preoperation コールバックルーチン
```

がコールバックで FLT_preoperation_SUCCESS_を返した場合、フィルターマネージャーは、i/o の完了時にミニフィルタードライバーの

```
postoperation コールバックルーチン
```

を呼び出します。

** ミニ**フィルタードライバーの preoperation コールバックルーチンがコールバックで FLT_preoperation_SUCCESS_を返したが、ミニフィルタードライバーが操作の postoperation コールバックルーチンを登録していない場合、システムはチェックされたビルドでアサートします。

ミニフィルタードライバーの preoperation コールバックルーチンがコールバックと共に FLT_PREOPERATION_SUCCESS_を返した場合、その完了後の出力パラメーターで NULL 以外の値を返すことができます。このパラメーターは、対応する postoperation コールバックルーチンに渡される省略可能なコンテキストポインターです。Postoperation コールバックルーチンは、このポインターを完了後の入力パラメーターに受け取ります。

_コールバック状態値を使用した FLT_PREOP_SUCCESS_は、すべての種類の i/o 操作に対して返すことができます。

FLT_PREOP_SUCCESS__コールバックなしで返す

2020/04/24 • [Edit Online](#)

ミニフィルタードライバーの

```
preoperation コールバックルーチン
```

が、FLT_preoperation_SUCCESS__コールバックを返さない場合、フィルタマネージャはミニフィルタドライバの

```
postoperation コールバックルーチン
```

を呼び出しません (存在する場合)。i/o の完了時。

ミニフィルタードライバーの preoperation コールバックルーチンが、FLT_PREOPERATION_SUCCESS__コールバックを返さない場合は、*その実行元*の出力パラメーターでNULLを返す必要があります。

FLT_PREOP_SUCCESS__コールバック状態値は、すべての種類の i/o 操作に対して返すことができます。

FLT_PREOP_SYNCHRONIZE を返す

2020/04/24 • [Edit Online](#)

ミニフィルタードライバの **preoperation コールバックルーチン** が、FLT_preoperation_SYNCHRONIZE を返すことによって i/o 操作を同期する場合、フィルターマネージャーは、i/o の完了時にミニフィルタードライバの **postoperation コールバックルーチン** を呼び出します。

フィルターマネージャーは、呼び出し元が preoperation コールバックと同じスレッドコンテキストでミニフィルタードライバの postoperation コールバックルーチンを呼び出します (IRQL ≤ APC_LEVEL)。 (このスレッドコンテキストは、必ずしも元のスレッドのコンテキストではないことに注意してください)。

ミニフィルタードライバの preoperation コールバックルーチンから FLT_PREOPERATION_SYNCHRONIZE が返されても、ミニフィルタードライバが操作の postoperation コールバックルーチンを登録していない場合は、チェックされたビルドに対してシステムがアサート ます。

ミニフィルタードライバの preoperation コールバックルーチンから FLT_PREOPERATION_SYNCHRONIZE が返された場合、その完了後の出力パラメーターに NULL 以外の値を返すことができます。このパラメーターは、対応する postoperation コールバックルーチンに渡される省略可能なコンテキストポインターです。Postoperation コールバックルーチンは、このポインターを完了後の入力パラメーターに受け取ります。

ミニフィルタードライバの preoperation コールバックルーチンは、IRP ベースの i/o 操作の場合にのみ、FLT_PREOPERATION_SYNCHRONIZE を返す必要があります。ただし、この状態値は、他の種類の操作でも返すことができます。IRP ベースの i/o 操作ではない i/o 操作に対して返された場合、フィルターマネージャーは、この戻り値を _コールバックと共に _PREOP_成功_として扱います。操作が IRP ベースの i/o 操作であるかどうかを判断するには、**FLT_is_irp_operation** マクロを使用します。

ミニフィルタードライバは、create 操作に対して FLT_PREOP_SYNCHRONIZE を返さないようにする必要があります。これらの操作は既にフィルターマネージャーによって同期されているためです。ミニフィルタードライバで、IRP_MJ の preoperation および postoperation コールバックルーチンが登録されている場合、作成後のコールバックルーチンは、作成前のコールバックルーチンと同じスレッドコンテキストで、IRQL = パッシブ_レベルで呼び出されます。

ミニフィルタードライバは、非同期の読み取り操作または書き込み操作のために、FLT_PREOP_SYNCHRONIZE を返さないようにする必要があります。このようにすると、ミニフィルタードライバとシステムのパフォーマンスが著しく低下し、変更されたページライタースレッドがブロックされた場合などにデッドロックが発生する可能性があります。IRP ベースの読み取り操作または書き込み操作に対して FLT_PREOP_SYNCHRONIZE を返す前に、ミニフィルタードライバは、**FltIsOperationSynchronous** を呼び出して、操作が同期されていることを確認する必要があります。

次の種類の i/o 操作は同期できません。

- Oplock ファイルシステム制御 (FSCTL) 操作 (MajorFunction は IRP_MJ_ファイル_システム_コントロール; Fscontrolcode は **FSCTL_request_FILTER_OPLOCK**、**FSCTL_request_BATCH_OPLOCK**、**FSCTL_request_oplock_level_1**、または **FSCTL_request_oplock_level_2**) です。
- ディレクトリの変更操作を通知する (MajorFunction is IRP_MJ_DIRECTORY_CONTROL; Minorfunction は、__ディレクトリの変更を通知_通知を_します。)
- バイト範囲ロック要求 (MajorFunction は IRP_MJ_LOCK_CONTROL; Minorfunction は、IRP__ロックされています。)

これらの操作のいずれに対しても、FLT_PREOP_SYNCHRONIZE を返すことはできません。

操作前コールバック ルーチン内で I/O 操作を完了する

2020/04/24 • [Edit Online](#)

I/O 操作を完了するには、操作の処理を停止し、最後の NTSTATUS 値を割り当てて、フィルターマネージャーに返します。

ミニフィルタードライバで I/O 操作が完了すると、フィルターマネージャーは次の処理を実行します。

- は、現在のミニフィルタードライバ、レガシフィルター、またはファイルシステムの下にあるミニフィルタードライバに操作を送信しません。
- ミニフィルタードライバインスタンススタック内の現在のミニフィルタードライバの前にあるミニ操作コールバックルーチンを呼び出します。
- は、現在のミニフィルタードライバの postoperation コールバックルーチンが存在する場合、その操作を呼び出しません。

ミニフィルタードライバの preoperation コールバックルーチンは、次の手順を実行して I/O 操作を完了します。

1. コールバックデータ構造の `IoStatus.Status` フィールドを、操作の最後の NTSTATUS 値に設定します。
2. `FLT_PREOP` を返します。

I/O 操作を完了する preoperation コールバックルーチンで、NULL 以外の完了コンテキストを設定することはできません (完了出力パラメーター)。

ミニフィルタードライバは、次の手順を実行して、以前に保留された I/O 操作の作業ルーチンで操作を完了することもできます。

1. コールバックデータ構造の `IoStatus.Status` フィールドを、操作の最後の NTSTATUS 値に設定します。
2. 作業ルーチンが `FltCompleteFLT Dedpreoperation` を呼び出すときに、コールアウトステータスパラメーターで `_preop_COMPLETE` を渡しました。

I/O 操作の完了時に、ミニフィルタードライバは、コールバックデータ構造の `IoStatus.Status` フィールドを操作の最終 NTSTATUS 値に設定する必要がありますが、この `ntstatus` 値は状態 `PENDING` または `STATUS_FLT` にはできません__高速 IO を許可しません。クリーンアップまたは閉じる操作の場合、フィールドは `[SUCCESS]_[SUCCESS]` になっている必要があります。これらの操作は、他の NTSTATUS 値では完了できません。

I/O 操作の完了は、次のように、NTSTATUS 値に応じて、"成功" または "操作の失敗" と呼ばれることがよくあります。

- I/O 操作を成功させるには、正常に完了したか、ステータス `_success` などの情報の NTSTATUS 値を使用して、I/O 操作を完了します。
- I/O 操作が失敗するようにするには、エラーまたは警告の NTSTATUS 値を使用して完了する必要があります。たとえば、状態 `_無効_デバイス_` の要求または状態 `_バッファ_オーバーフロー` します。

NTSTATUS 値は、`ntstatus` に定義されています。これらの値は、成功、情報、警告、エラーの4つのカテゴリに分類されます。これらの値の詳細については、「[NTSTATUS 値の使用](#)」を参照してください。

Preoperation コールバックルーチンで高速 i/o 操作を許可しない

2020/04/24 • [Edit Online](#)

場合によっては、ミニフィルタードライバーでは、実行するのではなく、高速の i/o 操作を許可しないことがあります。高速 i/o 操作を禁止すると、高速な i/o パスを操作に使用できなくなります。

I/o 操作の完了と同様に、高速な i/o 操作を禁止するということは、処理を停止し、フィルターマネージャーに返すことを意味します。ただし、高速の i/o 操作を許可しないことは、その処理を完了することとは異なります。ミニフィルタードライバーで i/o マネージャーによって発行された高速 i/o 操作が禁止されている場合、i/o マネージャーは、同等の IRP ベースの操作と同じ操作を再実行することがあります。

ミニフィルタードライバーの **preoperation コールバックルーチン** で高速 i/o 操作が禁止されている場合、フィルターマネージャーは次の処理を実行します。

- は、現在のミニフィルタードライバー、レガシフィルター、またはファイルシステムの下にあるミニフィルタードライバーに操作を送信しません。
- ミニフィルタードライバーインスタンススタック内の現在のミニフィルタードライバーの前にあるミニ**操作コールバックルーチン**を呼び出します。
- は、現在のミニフィルタードライバーの postoperation コールバックルーチンが存在する場合、その操作を呼び出しません。

ミニフィルタードライバーでは、FLT_PREOP_を返すことによって、操作の preop コールバックルーチンからの_FAO を許可しないようにすることで、高速な i/o 操作を行うことができません。

Preoperation コールバックルーチンは、コールバックデータ構造の FLT フィールドを設定しないでください。フィルターマネージャーによって、このフィールドが自動的に status__設定され、高速_IO_は許可されません。

FLT_PREOP__許可されていない高速 i/o 操作の場合にのみ返すことができます。操作が高速 i/o 操作であるかどうかを確認するには、「**FLT_is_faano操作**」を参照してください。

ミニフィルタードライバーは、FLT_PREOP_を返すことができません。IRP_MJ_SHUTDOWN、IRP_MJ_VOLUME_MOUNT、または IRP_MJ_VOLUME_のマウント解除操作で、_FAを許可しません。

プリ操作コールバックルーチンで i/o 操作を保留しています

2020/04/24 • [Edit Online](#)

ミニフィルタードライバーの **preoperation コールバックルーチン** は、操作をシステムの作業キューに送信し、FLT_preoperation_PENDING を返すことによって、i/o 操作を保留できます。この状態値を返すことは、ミニフィルタードライバーが、i/o 操作の処理を再開するために **Fltcompletependingpreoperation** を呼び出すまで i/o 操作の制御を維持していることを示します。

ミニフィルタードライバーの preoperation コールバックルーチンは、次の手順を実行して i/o 操作を pending します。

1. **FltQueueDeferredIoWorkItem** などのルーチンを呼び出して、i/o 操作をシステム作業キューに送信します。
2. FLT_PREOP_PENDING を返しています。

すべて (またはほとんど) の入出力操作を保留する必要があるミニフィルタードライバーでは、**FltQueueDeferredIoWorkItem** などのルーチンを使用して操作を保留することはできません。このルーチンを呼び出すと、システムの作業キューがあふれてしまう可能性があるからです。代わりに、このようなミニフィルタードライバーでは、キャンセルセーフキューを使用する必要があります。キャンセルセーフキューの使用の詳細については、「[FltCbqInitialize](#)」を参照してください。

次のいずれかの条件に該当する場合、**FltQueueDeferredIoWorkItem** の呼び出しは失敗することに注意してください。

- この操作は、IRP ベースの i/o 操作ではありません。
- 操作はページング i/o 操作です。
- 現在のスレッドの **TopLevelIrp** フィールドが **NULL** ではありません。(このフィールドの値を検索する方法の詳細については、「[IoGetTopLevelIrp](#)」を参照してください)。
- I/o 操作のターゲットインスタンスが破棄されています。

ミニフィルタードライバーの preoperation コールバックルーチンが、FLT_PREOPERATION_PENDING を返した場合、完了後の出力パラメーターで **NULL** を返す必要があります。

ミニフィルタードライバーは、IRP ベースの i/o 操作に対してのみ保留中の FLT_PREOP_PENDING を返すことができます。操作が IRP ベースの i/o 操作であるかどうかを判断するには、**FLT_is_irp_operation** マクロを使用します。

I/o 操作をデキューして処理する作業ルーチンでは、操作の処理を再開するために **Fltcompletependingpreoperation** を呼び出す必要があります。

操作後コールバック ルーチンの記述

2020/04/24 • [Edit Online](#)

ファイルシステムミニフィルタードライバーは、1つまたは複数の *postoperation* コールバックルーチンを使用して、i/o 操作をフィルター処理します。

Postoperation コールバックルーチンは、次のいずれかのアクションを実行できます。

- Postoperation ルーチンで、完了作業を直接実行します。すべての完了作業は、IRQL <= ディスパッチ_レベルで実現できます。
- 安全な IRQL で完了作業を実行します。__状態を返します。_処理_必要であり、ワーカースレッドをキューに置いて、安全な IRQL で処理を許可します。処理が完了すると、ワーカースレッドは [FtlCompleteDependedPostoperation](#) を呼び出して postoperation 処理を続行します。
- 正常な作成操作をキャンセルします。

[Postoperation コールバックルーチン](#)は、従来のファイルシステムフィルタードライバーで使用される *完了* ルーチンに似ています。

ミニフィルタードライバーは、特定の種類の i/o 操作に対して postoperation コールバックルーチンを登録します。これは、[preoperation コールバックルーチン](#)を登録するのと同じ方法で、つまり、コールバックルーチンのエントリポイントを operationregistration に格納します。[FLT_登録](#) 構造のメンバーで、ミニフィルタードライバーは、その [Driverentry](#) ルーチン [fltregisterfilter](#) にパラメーターとして渡します。

ミニフィルタードライバーは、preoperation または postoperation コールバックルーチンが登録されている種類の i/o 操作だけを受け取ります。ミニフィルタードライバーは、postoperation コールバックを登録せずに、特定の種類の i/o 操作に対して preoperation コールバックルーチンを登録できます。逆の場合も同様です。

すべての postoperation コールバックルーチンは、次のように定義されます。

```
typedef FLT_POSTOP_CALLBACK_STATUS
(*PFLT_POST_OPERATION_CALLBACK) (
    IN OUT PFLT_CALLBACK_DATA Data,
    IN PCFLT_RELATED_OBJECTS FltObjects,
    IN PVOID CompletionContext,
    IN FLT_POST_OPERATION_FLAGS Flags
);
```

完了ルーチンと同様に、postoperation コールバックルーチンは、任意のスレッドコンテキストで、IRQL <= ディスパッチ_レベルで呼び出されます。

IRQL = ディスパッチ_レベルで呼び出すことができるため、postoperation コールバックルーチンは、[FtlLockUserBuffer](#) や [RtlCompareUnicodeString](#) などの低い irql で呼び出す必要があるカーネルモードルーチンを呼び出すことはできません。同じ理由から、postoperation コールバックルーチンで使用するデータ構造は、非ページプールから割り当てられる必要があります。

次のような状況では、上記の規則に対していくつかの例外があります。

- ミニフィルタードライバーの preoperation コールバックルーチンが、IRP ベースの i/o 操作に対して FLT_PREOPERATION_SYNCHRONIZE を返した場合、対応する postoperation コールバックルーチンは、同じスレッド内の IRQL <= APC_レベルで呼び出されます。preoperation コールバックルーチンとしてのコンテキスト。
- 高速 i/o 操作の postoperation コールバックルーチンは、preoperation コールバックルーチンと同じスレッドコンテキストで、IRQL = パッシブ_レベルで呼び出されます。
- 作成後のコールバックルーチンは、IRP_MJ_CREATE 操作を開始したスレッドのコンテキストで、IRQL = パッシブ_レベルで

呼び出されます。

フィルターマネージャーが、特定の i/o 操作に対してミニフィルタードライバーの postoperation コールバックルーチンを呼び出すと、ミニフィルタードライバーによって i/o 操作が一時的に制御されます。ミニフィルタードライバーは、次のいずれかの操作を実行するまで、このコントロールを保持します。

- Postoperation コールバックルーチンからの処理_完了した FLT_POSTOP_を返します。
- Postoperation コールバックルーチンで保留されていた IRP ベース i/o 操作を処理した作業ルーチンから[Fltcompletenependedpostoperation](#)を呼び出します。

このセクションの内容:

[I/o 操作の完了処理を実行しています](#)

[Postoperation コールバックルーチンで i/o 操作を保留しています](#)

[Postoperation コールバックルーチンでの i/o 操作の失敗](#)

I/O 操作の完了処理を実行しています

2020/04/24 • [Edit Online](#)

ミニフィルタードライバーの **postoperation コールバックルーチン** は、基になるファイルシステム、レガシフィルタ、またはミニフィルタードライバーインスタンススタック内の高度が低い別のミニフィルタードライバーによって i/o 操作が完了したときに呼び出されます。

さらに、ミニフィルタードライバーインスタンスが破棄されている場合、フィルタマネージャーは、インスタンスが **preoperation コールバック** を受信し、**postoperation コールバック** を待機しているすべての i/o 操作を "ドレイン" します。このような状況では、フィルタマネージャーは、i/o 操作が完了していない場合でもミニフィルタードライバーの postoperation コールバックルーチンを呼び出し、フラグ入力に `_ドレインフラグ` を設定して `FLTFL_POST_` 操作を設定します。引き。

`FLTFL_POST_` 操作 `_ドレインフラグ` が設定されている場合、ミニフィルタードライバーは通常の完了処理を実行できません。代わりに、必要なクリーンアップのみを実行する必要があります。たとえば、その preoperation コールバックルーチンの完了時パラメーターに対してミニフィルタードライバーによって割り当てられたメモリを解放し、`FLT_postop_` 終了した場合は、を返し `_プロセス`。

ここでは、次のトピックについて説明します。

完了処理が安全な IRQL で実行されることの確認

完了処理が安全な IRQL で実行されるようにする

2020/04/24 • • [Edit Online](#)

[Postoperation コールバックルーチンの記述](#)に関する記事で説明したように、IRP ベースの i/o 操作の [postoperation コールバックルーチン](#)は、フィルター処理されたドライバーの preoperation コールバックルーチンでない限り、IRQL = ディスパッチ_レベルで呼び出すことができます。FLT_PREOP_SYNCHRONIZE を返すことによって操作が同期されました。または、操作が作成操作であり、本質的に同期がとられています。(この戻り値の詳細については、「[return FLT_PREOP_SYNCHRONIZE](#)」を参照してください)。

ただし、まだ同期されていない IRP ベースの i/o 操作の場合、ミニフィルタードライバーはを使用して2つの手法を使用して、完了処理が IRQL <= APC_レベルで実行されるようにすることができます。

最初の方法は、終了処理を IRQL <= APC_レベルで実行できるようになるまで i/o 操作を保留する postoperation コールバックルーチンです。この手法については、「[Postoperation コールバックルーチンで I/O 操作を保留中](#)」を参照してください。

2つ目の手法では、ミニフィルタードライバーの postoperation コールバックルーチンが [Fltdo completion processing when safe](#) を呼び出します。Fltdo depends Processing when safe は、現在の IRQL が >= ディスパッチ_レベルの場合にのみ i/o 操作を実行します。それ以外の場合、このルーチンはミニフィルタードライバーの [Safe post callback](#) ルーチンを直ちに実行します。この手法については、「[Fltdo の Processing when safe](#)」を参照してください。

操作後コールバック ルーチン内で I/O 操作を保留にする

2020/04/24 • [Edit Online](#)

ミニフィルタードライバの **postoperation コールバックルーチン** は、次の手順を実行して i/o 操作を保留できます。

1. I/O 操作のために作業項目を割り当てるために **FltAllocateDeferredIoWorkItem** を呼び出しています。
2. **FltQueueDeferredIoWorkItem** を呼び出して、i/o 操作をシステムの作業キューに送信します。
3. FLT_POSTOP を返すことで、_処理_必要な_を増やすことができます。

次のいずれかの条件に該当する場合、**FltQueueDeferredIoWorkItem** の呼び出しは失敗することに注意してください。

- この操作は、IRP ベースの i/o 操作ではありません。
- 操作はページング i/o 操作です。
- 現在のスレッドの **TopLevelIrp** フィールドが **NULL** ではありません。(このフィールドの値を検索する方法の詳細については、「[IoGetTopLevelIrp](#)」を参照してください)。
- I/O 操作のターゲットインスタンスが破棄されています。(フィルターマネージャーは、この状況を示しています。これは、FLTFL_POST_操作_、フラグ入カパラメーター内のドレインフラグを postoperation コールバックルーチンに設定することによって示されます。

このエラーを処理するには、ミニフィルタードライバを準備する必要があります。ミニフィルタードライバがこのようなエラーを処理できない場合は、「i/o 操作を保留するのではなく、[FLT_PREOP_同期を返す](#)」で説明されている手法を使用することを検討してください。

ミニフィルタードライバの postoperation コールバックルーチンによって FLT_POSTOP が返された後、より多くの_処理_必要な場合、フィルターマネージャーはミニフィルターを使用するまで i/o 操作の完了処理を実行しません。ドライバの作業ルーチンは **Fltcompletependingpostoperation** を呼び出して、操作の制御をフィルターマネージャーに返します。このような状況では、操作のコールバックデータ構造の **iostatus. Status** フィールドに失敗の NTSTATUS 値が設定されている場合でも、フィルターマネージャーはそれ以上の処理を実行しません。

I/O 操作の完了処理をデキューして実行する作業ルーチンは、**Fltcompletependingpostoperation** を呼び出して、操作の制御をフィルターマネージャーに返す必要があります。

操作後コールバック ルーチン内で I/O 操作を失敗とする

2020/04/24 • [Edit Online](#)

ミニフィルタードライバーの **postoperation コールバックルーチン** は、成功した i/o 操作を失敗させる可能性があります。i/o 操作の失敗によって操作の効果を元に戻すことはできません。ミニフィルタードライバーは、操作を元に戻すために必要な処理を実行します。

たとえば、ミニフィルタードライバーの作成後のコールバックルーチンは、次の手順を実行して、正常な IRP_MJ_作成操作に失敗する可能性があります。

1. 作成操作で作成または開かれたファイルを閉じるには、**Fltcancelfileopen** を呼び出します。Fltcancelfileopen では、ファイルに対する変更は元に戻されないことに注意してください。たとえば、Fltcancelfileopen は、新しく作成されたファイルを削除したり、切り捨てられたファイルを以前のサイズに復元したりすることはありません。
2. コールバックデータ構造の **lostatus. status** フィールドを、操作の最後の NTSTATUS 値に設定します。この値は、状態_アクセス_拒否されたなどの有効なエラーの値である必要があります。
3. コールバックデータ構造の **lostatus. 情報** フィールドをゼロに設定します。
4. FLT_POSTOP_返され_処理が終了しました。

コールバックデータ構造の **lostatus. status** フィールドを、操作の最終 NTSTATUS 値に設定するときは、ミニフィルタードライバーで有効なエラー ntstatus 値を指定する必要があります。ミニフィルタードライバーでは、_高速_IO を許可__FLT を指定できないことに注意してください。フィルターマネージャーのみがこの NTSTATUS 値を使用できます。

Fltcancelfileopen の呼び出し元は、IRQL <= APC_レベルで実行されている必要があります。ただし、ミニフィルタードライバーは、作成後のコールバックルーチンから安全にこのルーチンを呼び出すことができます。これは、IRP_MJ_CREATE 操作の場合、postoperation コールバックルーチンが、スレッドのコンテキストで IRQL = パッシュ_レベルで呼び出されるためです。これで作成操作が開始されました。

I/O 操作のパラメーターの修正

2020/04/24 • [Edit Online](#)

ミニフィルタードライバーは、i/o 操作のパラメーターを変更できます。たとえば、ミニフィルタードライバーの **preoperation コールバックルーチン** は、操作のターゲットインスタンスを変更することで、i/o 操作を別のボリュームにリダイレクトできます。新しいターゲットインスタンスは、別のボリュームの同じ高度な同じミニフィルタードライバーのインスタンスである必要があります。

i/o 操作のパラメーターは、操作のコールバックデータ (**FLT_callback_data**) 構造体と i/o パラメーターブロック (**FLT_IO_parameter_block**) 構造体にあります。ミニフィルタードライバーの **preoperation コールバックルーチン** と **postoperation コールバックルーチン** は、データ入力パラメーター内の操作のコールバックデータ構造へのポインターを受け取ります。コールバックデータ構造体の *lopb* メンバーは、操作のパラメーターを含む i/o パラメーターブロック構造体へのポインターです。

ミニフィルタードライバーの preoperation コールバックルーチンによって i/o 操作のパラメーターが変更された場合、ミニフィルタードライバーインスタンススタック内のミニフィルタードライバーの下にあるすべてのミニフィルタードライバーは、変更されたパラメーターを preoperation に受け取り、postoperation コールバックルーチン。

修正されたパラメーターは、現在のミニフィルタードライバーの postoperation コールバックルーチン、またはミニフィルタードライバーインスタンススタック内のミニフィルタードライバーであるミニフィルタードライバーによって受け取られません。どのような状況でも、ミニフィルタードライバーの preoperation および postoperation コールバックルーチンは、指定された i/o 操作に対して同じ入力パラメーター値を受け取ります。

i/o 操作のパラメーターを変更した後、preoperation または postoperation コールバックルーチンは、コールバックデータ構造の *lostatus* フィールドの内容が変更されていない限り、**FltSetCallbackDataDirty** を呼び出すことによってこれを実行したことを示す必要があります。それ以外の場合、フィルターマネージャーは、パラメーター値に対する変更をすべて無視します。**FltSetCallbackDataDirty** は、i/o 操作のコールバックデータ構造で **FLTFL_CALLBACK_DATA_DIRTY** フラグを設定します。ミニフィルタードライバーでは、**Fltiscallbackdatadirty** を呼び出すか、**Fltclearbackdatadirty** を呼び出してクリアすることで、このフラグをテストできます。

ミニフィルタードライバーの preoperation コールバックルーチンによって i/o 操作のパラメーターが変更された場合、ミニフィルタードライバーインスタンススタック内のミニフィルタードライバーの下にあるすべてのミニフィルタードライバーは、データ内の変更されたパラメーターを受け取ります。**FltObjects** および postoperation コールバックルーチンに入力パラメーターを入力します。(ミニフィルタードライバーは、**FltObjects** パラメーターによってポイントされる **FLT_関連_オブジェクト** 構造の内容を直接変更することはできません。ただし、ミニフィルタードライバーによって、i/o 操作の対象インスタンスまたはターゲットファイルオブジェクトが変更された場合は、フィルターマネージャーによって、対応するインスタンスの値または FLT_関連する_オブジェクトの **FileObject** メンバーの値が変更されます。より低いミニフィルタードライバーに渡される構造体。)

ミニフィルタードライバーの preoperation コールバックルーチンによって実行されるパラメーター変更は、ミニフィルタードライバーの独自の postoperation コールバックルーチンによって受信されませんが、preoperation コールバックルーチンは、変更されたパラメーターに関する情報を渡すことができます。ミニフィルタードライバーの独自の postoperation コールバックルーチン。Preoperation コールバックルーチンが、**FLT_PREOPERATION_SUCCESS** を返すことによってスタックに i/o 操作を渡した場合、**_CALLBACK** または **FLT_PREOPERATION_SYNCHRONIZE** では、変更されたパラメーターに関する情報を格納できます。値は、完了後の出力パラメーターによってポイントされる、ミニフィルタで定義された構造になります。フィルターマネージャーは、完了後の入力パラメーターにこの構造体のポインターを postoperation コールバックルーチンに渡します。

i/o 操作のパラメーターの詳細については、「**FLT_CALLBACK_DATA**」および「**FLT_IO_PARAMETER_BLOCK**」を参照してください。

I/O 操作のバッファ処理メソッドの確認

2020/04/24 • [Edit Online](#)

デバイスドライバと同様に、ファイルシステムはユーザーモードアプリケーションとシステムデバイス間でデータを転送します。オペレーティングシステムには、データバッファにアクセスするための次の3つの方法が用意されています。

- バッファ内 i/o では、i/o マネージャーは、非ページブールから操作のシステムバッファを割り当てます。i/o マネージャーは、i/o 操作を開始したスレッドのコンテキストで、このシステムバッファからアプリケーションのユーザーバッファにデータをコピーします。その逆も同様です。
- *ダイレクト i/o* では、i/o マネージャーによってユーザーバッファがブローおよびロックされます。次に、ロックされたバッファをマップするためのメモリ記述子リスト (MDL) を作成します。i/o マネージャーは、i/o 操作を開始したスレッドのコンテキストでバッファにアクセスします。
- バッファも *直接 i/o* でも、i/o マネージャーはシステムバッファを割り当てません。また、ユーザーバッファをロックまたはマップしません。代わりに、単にバッファの元のユーザー領域の仮想アドレスをファイルシステムスタックに渡します。ドライバは、それらが開始スレッドのコンテキストで実行されていること、およびバッファアドレスが有効であることを確認する役割を担います。

ミニフィルタードライバを使用する前に、ユーザー領域内のアドレスを検証する必要があります。i/o マネージャーとフィルターマネージャーでは、このようなアドレスは検証されず、ミニフィルタードライバに渡されたバッファに埋め込まれているポインターは検証されません。

すべての標準の Microsoft ファイルシステムでは、ほとんどの i/o 処理で、バッファも直接 i/o も使用されません。

バッファリングメソッドの詳細については、「[データバッファにアクセスするためのメソッド](#)」を参照してください。

IRP ベースの i/o 操作の場合、使用されるバッファリング方法は操作に固有であり、次の要因によって決定されます。

- 実行されている i/o 操作の種類
- ファイルシステムボリュームの [デバイス_オブジェクト](#) 構造の Flags メンバーの値
- i/o 制御 (IOCTL) 操作とファイルシステム制御 (FSCTL) 操作の場合、IOCTL または FSCTL が定義されたときに CTL_コードマクロに渡された *Transfertype* パラメーターの値

バッファを持つ高速 i/o 操作では、バッファも直接も直接使用されません。

ファイルシステムコールバック操作にはバッファがありません。

このセクションの内容:

[IRP ベースまたは高速 i/o の可能性がある操作](#)

[デバイスオブジェクトフラグに従う IRP ベースの i/o 操作](#)

[バッファリングされた i/o を常に使用する IRP ベースの i/o 操作](#)

[常にバッファも直接も直接使用する IRP ベース i/o 操作](#)

[IRP ベースの IOCTL および FSCTL 操作](#)

[バッファを持たない IRP ベースの i/o 操作](#)

IRP ベースまたは高速 I/O とすることが可能な操作

2020/04/24 • [Edit Online](#)

次の種類の操作では、IRP ベースまたは高速の i/o 操作を実行できます。

- IRP_MJ_デバイス_コントロールです。(IRP_MJ_内部_デバイス_コントロールは常に IRP ベースであることに注意してください)。
- IRP_MJ_クエリ_情報。Fileinformationclassパラメーターがfileinformationclass、Fileinformationclass、またはFileNetworkOpenInformationの場合、この操作は高速 i/o になることがあります。
- IRP_MJ_読み取ります。ミニフィルタードライバーは、MJ i/o IRP_が受信されないようにするために、[FLT_操作_登録](#)構造で、キャッシュされた_IO フラグの_を_スキップするように FLTFL_操作を設定でき_読み取り操作とキャッシュされた IRP ベースの読み取り。
- IRP_MJ_書き込みです。ミニフィルタードライバーでは、FLTFL_操作_登録_スキップ_キャッシュされた_IO フラグを FLT_操作_登録構造に設定することにより、高速 i/o IRP_MJ_書き込みが受信されないようにすることができます。操作とキャッシュされた IRP ベースの書き込み。

これらの操作のいずれかが高速 i/o 操作である場合、対応する IRP ベースの操作で別のバッファリング方法が使用されていても、バッファも直接 i/o も使用されません。

IRP_MJ_デバイス_制御が高速 i/o 操作である場合、IOCTL の転送の種類に関係なく、常にバッファと直接の両方の i/o が使用されます。

IRP_MJ_LOCK_CONTROL は、IRP ベースまたは高速の i/o 操作にすることができますが、バッファはありません。

デバイス オブジェクト フラグに従う IRP ベースの I/O 操作

2020/04/24 • [Edit Online](#)

次の IRP ベース i/o 操作のバッファリングメソッドは、ファイルシステムボリュームの**デバイス_オブジェクト**構造のFlagsメンバーの値によって決まります。

- IRP_MJ_DIRECTORY_コントロール
- IRP_MJ_クエリ_EA
- IRP_MJ_クエリ_クォータ
- IRP_MJ_READ
- IRP_MJ_設定_EA
- IRP_MJ_設定_クォータ
- IRP_MJ_WRITE

DO_バッファ_IO と、flagsメンバー内のダイレクト_io フラグを_するために、次のように使用されます。

- DO_バッファリング_IO フラグが設定されている場合、操作はバッファ i/o を使用します。
- DO_DIRECT_IO フラグが設定されていて、DO_バッファリング_IO フラグが設定されていない場合、操作は直接 i/o を使用します。
- どちらのフラグも設定されていない場合、操作ではバッファも直接 i/o も使用されません。

デバイスオブジェクトフラグの詳細については、「**デバイス_オブジェクト**」と「**デバイスオブジェクトの初期化**」を参照してください。

IRP_MJ_READ および IRP_MJ_ の書き込みは、IRP ベースまたは高速の i/o 操作であることに注意してください。IRP ベースの場合、バッファリングメソッドは、前述のデバイスオブジェクトフラグによって決定されます。これらの操作が高速 i/o の場合は、バッファも直接も使用されません。IRP ベースまたは高速 i/o 操作の i/o 操作の詳細については、「**irp ベースまたは高速**i/o を使用できる操作」を参照してください。

バッファ処理された I/O を常に使用する IRP ベースの I/O 操作

2020/04/24 • [Edit Online](#)

次の IRP ベースの i/o 操作では、ファイルシステムボリュームのオブジェクト構造のフラグメンバー [_オブジェクト](#) 構造の値に関係なく、常にバッファ内 i/o を使用します。

- IRP_MJ_CREATE ([EaBuffer parameter](#))
- IRP_MJ_QUERY_INFORMATION
- IRP_MJ_クエリ_ボリューム_情報
- IRP_MJ_SET_INFORMATION
- IRP_MJ__ボリューム_情報の設定

IRP_MJ_クエリ_情報も、高速 i/o 操作であることに注意してください。高速な i/o 操作では、バッファも直接 i/o も使用されません。IRP ベースまたは高速 i/o 操作の i/o 操作の詳細については、「[irp ベースまたは高速](#)i/o を使用できる操作」を参照してください。

バッファ処理された I/O またはダイレクト I/O を常に使用しない IRP ベースの I/O 操作

2020/04/24 • [Edit Online](#)

次の IRP ベースの i/o 操作では、ファイルシステムボリュームの**デバイス_オブジェクト**構造のFlagsメンバーの値に関係なく、バッファも直接 i/o も使用されません。

- IRP_MJ_PNP
- IRP_MJ_クエリ_セキュリティ
- IRP_MJ_設定_セキュリティ
- IRP_MJ_SYSTEM_CONTROL

IRP ベースの IOCTL と FSCTL 操作

2020/04/24 • [Edit Online](#)

次の IRP ベースの I/O 操作では、I/O 制御コード (IOCTL) またはファイル システムの制御コード (FSCTL) の定義で指定されている転送の種類に一致するバッファリング メソッドを使用します。

- IRP_MJ_DEVICE_CONTROL
- IRP_MJ_ファイル_システム_コントロール
- IRP_MJ_INTERNAL_DEVICE_CONTROL

転送の種類がで指定された、*TransferType*パラメーターは、CTL のコード マクロ。特定の IOCTL または FSCTL 転送の種類を取得するには、次のマクロを使用します。

```
#define METHOD_FROM_CTL_CODE(ctrlCode) ((ULONG)(ctrlCode & 3))
```

このマクロは、次の値のいずれかを返します。

```
#define METHOD_BUFFERED          0
#define METHOD_IN_DIRECT         1
#define METHOD_OUT_DIRECT        2
#define METHOD_NEITHER           3
```

詳細については、CTL マクロのコードは、「[I/O 制御コードを定義する](#)」を参照します。

注その IRP_MJ_デバイス_コントロールは高速な I/O 操作にもできます。高速な I/O 操作である場合は、IOCTL の転送の種類に関係なく、バッファも直接 I/O が常に使用します。タイミングの詳細については IRP_MJ_デバイス_コントロールが高速な I/O 操作を参照してください[操作することができますが IRP ベースまたは高速の I/O](#)します。

バッファが存在しない IRP ベースの I/O 操作

2020/04/24 • [Edit Online](#)

次の IRP ベースの I/O 操作があるないバッファ、バッファリング メソッドを含まないため。

- IRP_MJ_作成_ "メール スロット"
- IRP_MJ_作成_名前付き_パイプ
- IRP_MJ_ロック_コントロール

I/O 操作のユーザーバッファーへのアクセス

2020/04/24 • [Edit Online](#)

I/O 操作の `FLT_parameters` 構造体には、操作に固有の操作固有のパラメーターが含まれています。これには、操作で使われるバッファーのバッファーアドレスとメモリ記述子リスト (MDL) が含まれます。

IRP ベースの i/o 操作の場合、操作のバッファーは次を使用して指定できます。

- MDL のみ (通常はページング i/o 用)
- バッファーアドレスのみ
- バッファーアドレスと MDL

高速な i/o 操作の場合は、ユーザー領域のバッファーアドレスだけが指定されます。バッファーを持つ高速 i/o 操作では、バッファーも直接 i/o も使用されないため、MDL パラメーターを持つことはできません。

次のトピックでは、ミニフィルタードライバー `preoperation コールバックルーチン` と `postoperation コールバックルーチン` での IRP ベースおよび高速 i/o 操作に対するバッファーアドレスと mdls の処理に関するガイドラインを提供します。

[Preoperation コールバックルーチンでのユーザーバッファーへのアクセス](#)

[Postoperation コールバックルーチンでのユーザーバッファーへのアクセス](#)

操作前コールバック ルーチン内でユーザー バッファへアクセスする

2020/04/24 • [Edit Online](#)

ミニフィルタードライバーの

```
preoperation コールバックルーチン
```

は、次のように、IRP ベースの i/o 操作でバッファーを処理する必要があります。

- バッファーに対して MDL が存在するかどうかを確認します。MDL ポインターは、操作の

```
FLT_パラメーター
```

の

```
mdladdress
```

パラメーターまたは

```
outputmdladdress
```

パラメーターにあります。ミニフィルタードライバーは、

```
FltDecodeParameters
```

を呼び出して、MDL ポインターを照会できます。

有効な MDL を取得するための1つの方法として、コールバックデータ内の i/o パラメーターブロック

```
FLT_IO_parameter_block
```

の

```
minorfunction
```

メンバーで、IRP__MDL フラグを探します。次の例では、IRP__MDL フラグをチェックする方法を示します。

```
NTSTATUS status;
PMDL *ReadMdl = NULL;
PVOID ReadAddress = NULL;

if (FlagOn(CallbackData->Iopb->MinorFunction, IRP_MN_MDL))
{
    ReadMdl = &CallbackData->Iopb->Parameters.Read.MdlAddress;
}
```

ただし、IRP__MDL フラグは、読み取りおよび書き込み操作に対してのみ設定できます。

```
FltDecodeParameters
```

を使用して mdl を取得することをお勧めします。これは、ルーチンが任意の操作に対して有効な mdl をチェックするためです。次の例では、有効な場合にのみ、MDL パラメーターが返されます。

```
NTSTATUS status;
PMDL *ReadMdl = NULL;
PVOID ReadAddress = NULL;

status = FltDecodeParameters(CallbackData, &ReadMdl, NULL, NULL, NULL);
```

- バッファーに対して MDL が存在する場合は、

```
MmGetSystemAddressForMdlSafe
```

を呼び出してバッファーのシステムアドレスを取得し、このアドレスを使用してバッファーにアクセスします。

前の例から続けると、次のコードはシステムアドレスを取得します。

```
if (*ReadMdl != NULL)
{
    ReadAddress = MmGetSystemAddressForMdlSafe(*ReadMdl, NormalPagePriority);
    if (ReadAddress == NULL)
    {
        CallbackData->IoStatus.Status = STATUS_INSUFFICIENT_RESOURCES;
        CallbackData->IoStatus.Information = 0;
    }
}
```

- バッファーに対して MDL が存在しない場合は、バッファアドレスを使用してバッファーにアクセスします。ユーザー領域のバッファアドレスが有効であることを確認するには、ミニフィルタードライバーで

```
ProbeForRead
```

や

```
ProbeForWrite
```

などのルーチンを使用して、try/以外 のブロックですべてのバッファ参照を囲む必要があります。

Preoperation コールバックルーチンは、次のように、高速の i/o 操作でバッファーを処理する必要があります。

- バッファーアドレスを使用してバッファーにアクセスします (高速 i/o 操作には MDL を設定できないため)。
- ユーザー領域のバッファーアドレスが有効であることを確認するには、ミニフィルタードライバーでProbeForReadやProbeForWriteなどのルーチンを使用して、try/以外のブロックですべてのバッファー参照を囲む必要があります。

高速 i/o または IRP ベースの可能性のある操作では、すべてのバッファー参照をtry/以外のブロックで囲む必要があります。バッファリングされた i/o を使用する IRP ベースの操作では、これらの参照を囲む必要はありませんが、try/exceptブロックは安全な予防措置です。

操作後コールバック ルーチン内でユーザー バッファへアクセスする

2020/04/24 • [Edit Online](#)

ミニフィルタードライバー **postoperation コールバックルーチン**は、次のように、IRP ベースの i/o 操作でバッファーを処理する必要があります。

- バッファーに対して MDL が存在するかどうかを確認します。MDL ポインターは、操作の**FLT_パラメーター**の`mdladdress`パラメーターまたは`outputmdladdress`パラメーターにあります。ミニフィルタードライバーは、**FltDecodeParameters**を呼び出して、MDL ポインターを照会できます。

有効な MDL を取得するための1つの方法として、コールバックデータ内の i/o パラメーターブロック**FLT_IO_parameter_block**の**minorfunction**メンバーで、IRP__MDL フラグを探します。次の例では、IRP__MDL フラグをチェックする方法を示します。

```
NTSTATUS status;
PMDL *ReadMdl = NULL;
PVOID ReadAddress = NULL;

if (FlagOn(CallbackData->Iopb->MinorFunction, IRP_MN_MDL))
{
    ReadMdl = &CallbackData->Iopb->Parameters.Read.MdlAddress;
}
```

ただし、IRP__MDL フラグは、読み取りおよび書き込み操作に対してのみ設定できます。**FltDecodeParameters**を使用して mdl を取得することをお勧めします。これは、ルーチンが任意の操作に対して有効な mdl をチェックするためです。次の例では、有効な場合にのみ、MDL パラメーターが返されます。

```
NTSTATUS status;
PMDL *ReadMdl = NULL;
PVOID ReadAddress = NULL;

status = FltDecodeParameters(CallbackData, &ReadMdl, NULL, NULL, NULL);
```

- バッファーに対して MDL が存在する場合は、**MmGetSystemAddressForMdlSafe**を呼び出してバッファーのシステムアドレスを取得し、このアドレスを使用してバッファーにアクセスします。(MmGetSystemAddressForMdlSafeは IRQL <= DISPATCH_LEVEL で呼び出すことができます)。

前の例から続けると、次のコードはシステムアドレスを取得します。

```
if (*ReadMdl != NULL)
{
    ReadAddress = MmGetSystemAddressForMdlSafe(*ReadMdl, NormalPagePriority);
    if (ReadAddress == NULL)
    {
        CallbackData->IoStatus.Status = STATUS_INSUFFICIENT_RESOURCES;
        CallbackData->IoStatus.Information = 0;
    }
}
```

- バッファー用の MDL がいない場合は、**FLT_is_システム_buffer**マクロを使用して、システムバッファフラグが操作に対して設定されているかどうかを確認します。

- FLT_が_システム_バッファーマクロによってTRUEが返された場合、操作はバッファ i/o を使用し、IRQL = ディスパッチ_レベルでバッファに安全にアクセスできます。
- FLT_が_システム_バッファーマクロによってFALSEが返された場合、IRQL = ディスパッチ_レベルでバッファに安全にアクセスすることはできません。Postoperation コールバックルーチンをディスパッチ_レベルで呼び出すことができる場合、この呼び出しは、IRQL <= APC_レベルで処理できるようになるまで、操作を保留するために**FltDoCompletionProcessingWhenSafe**を呼び出す必要があります。FltDoFltLockUserBuffer ProcessingWhenSafeのsafepostcallbackパラメーターが指すコールバックルーチンは、最初にバッファをロックしてから**MmGetSystemAddressForMdlSafe**を呼び出すために、まず** should first call **FltLockUserBuffer**を呼び出します。バッファのシステムアドレスを取得します。

Postoperation コールバックルーチンは、次のように、高速の i/o 操作でバッファを処理する必要があります。

- バッファアドレスを使用してバッファにアクセスします (高速 i/o 操作には MDL を設定できないため)。
- ユーザー領域のバッファアドレスが有効であることを確認するには、ミニフィルタドライバで**ProbeForRead**や**ProbeForWrite**などのルーチンを使用して、try/以外のブロックですべてのバッファ参照を囲む必要があります。
- 高速 i/o 操作の postoperation コールバックルーチンは、正しいスレッドコンテキストで呼び出されることが保証されています。
- 高速 i/o 操作の postoperation コールバックルーチンは、IRQL <= APC_レベルで呼び出されることが保証されているので、**FltLockUserBuffer**などのルーチンを安全に呼び出すことができます。

次のコード例では、ディレクトリ制御操作のシステムバッファまたは高速 i/o フラグをチェックし、必要に応じて完了処理を延期します。

```
if (*DirectoryControlMdl == NULL)
{
    if (FLT_IS_SYSTEM_BUFFER(CallbackData) || FLT_IS_FASTIO_OPERATION(CallbackData))
    {
        dirBuffer = CallbackData->Iopb->Parameters.DirectoryControl.QueryDirectory.DirectoryBuffer;
    }
    else
    {
        // Defer processing until safe.
        if (!FltDoCompletionProcessingWhenSafe(CallbackData, FltObjects, CompletionContext, Flags,
        ProcessPostDirCtrlWhenSafe, &retValue))
        {
            CallbackData->IoStatus.Status = STATUS_UNSUCCESSFUL;
            CallbackData->IoStatus.Information = 0;
        }
    }
}
```

高速 i/o または IRP ベースのいずれかの操作を実行するには、すべてのバッファ参照をtry/以外のブロックで囲む必要があります。バッファリングされた i/o を使用する IRP ベースの操作では、これらの参照を囲む必要はありませんが、try/exceptブロックは安全な予防措置です。

ミニフィルター ドライバーでのコンテキストの管理

2020/04/24 • [Edit Online](#)

Aコンテキスト構造ミニフィルター ドライバーで定義されていると、フィルターのマネージャー オブジェクトに関連付けられていることができます。ミニフィルター ドライバーは、作成し、以下のオブジェクトのコンテキストを設定できます。

- ファイル (Windows Vista 以降のみ)。
- インスタンス
- Volumes
- ストリーム
- Stream ハンドル (オブジェクトのファイル)
- トランザクション (Windows Vista 以降のみ)。

非ページ プールから割り当てる必要がある、ボリュームのコンテキスト以外には、コンテキストをページまたは非ページ プールから割り当てることができます。

フィルター マネージャーに関連付けられている、オブジェクトが削除されるは、ミニフィルター ドライバーのインスタンスが、ボリュームからデタッチされている場合、またはミニフィルター ドライバーを読み込むときにコンテキストを自動的に削除します。

このセクションの内容:

[コンテキストの種類を登録します。](#)

[コンテキストを作成します。](#)

[コンテキストの設定](#)

[コンテキストを取得します。](#)

[参照元のコンテキスト](#)

[コンテキストを解放します。](#)

[コンテキストを削除します。](#)

[コンテキストを解放します。](#)

[コンテキストのファイル システムのサポート](#)

[ベスト プラクティス](#)

コンテキストの種類の登録

2020/04/24 • [Edit Online](#)

ミニフィルタードライバが[Driverentry](#)ルーチンから[Fltregisterfilter](#)を呼び出すと、使用する各種類のコンテキストを登録する必要があります。

コンテキスト型を登録するために、ミニバスドライバは、[FLT_コンテキスト_登録](#)構造体の可変長配列を作成し、その配列へのポインターを、ミニフィルタードライバが[Fltregisterfilter](#)の登録パラメータに渡す[FLT_登録](#)構造体の[contextregistration](#)メンバーに格納します。この配列内の要素の順序は関係ありません。ただし、配列の最後の要素は{FLT_CONTEXT_END}である必要があります。

ミニフィルタードライバが使用するコンテキストの種類ごとに、FLT_コンテキスト_登録構造の形式でコンテキスト定義を少なくとも1つ指定する必要があります。各 FLT_コンテキスト_登録構造は、コンテキストの型、サイズ、およびその他の情報を定義します。

ミニフィルタードライバで[FltAllocateContext](#)を呼び出すことによって新しいコンテキストを作成する場合、フィルターマネージャーは [FltAllocateContext](#)ルーチンの[size](#)パラメータと、FLT_Context_REGISTRATION 構造体の[size](#)および[Flags](#)メンバーを使用して、使用するコンテキスト定義を選択します。

固定サイズのコンテキストの場合、FLT_コンテキスト_登録構造の[size](#)メンバーは、ミニフィルタードライバによって定義されるコンテキスト構造の一部のサイズをバイト単位で指定します。コンテキストの最大サイズは MAXUSHORT (64 KB) です。0は有効なサイズ値です。フィルターマネージャーは、ルックアサイドリストを使用して固定サイズのコンテキストを実装します。フィルターマネージャーは、各サイズ値に対して2つのルックアサイドリストを作成します。1つはページングされていて、もう1つはページ

可変サイズのコンテキストの場合は、[size](#)メンバーを FLT に設定する必要があります。[_変数_サイズ_コンテキスト](#)に設定します。フィルターマネージャーは、ページングされたプールまたは非ページプールから、可変サイズのコンテキストを直接割り当てます。

FLT_CONTEXT_REGISTRATION 構造体の[Flags](#)メンバーでは、FLTFL_コンテキスト_登録_完全_サイズ_一致フラグを指定することはできません。ミニフィルタードライバで固定サイズのコンテキストが使用されていて、このフラグが指定されている場合、コンテキストのサイズが要求されたサイズ以上の場合、フィルターマネージャーはルックアサイドリストからコンテキストを割り当てます。それ以外の場合は、コンテキストのサイズが要求されたサイズと同じである必要があります。

指定されたコンテキストの種類では、ミニフィルタードライバは最大3つの固定サイズのコンテキスト定義を指定できます。それぞれ異なるサイズを持ち、1つの可変サイズの定義があります。詳細については、「[FLT_CONTEXT_REGISTRATION](#)」を参照してください。

ミニフィルタードライバでは、コンテキストが解放される前に呼び出されるコンテキストクリーンアップコールバックルーチンをオプションで指定できます。詳細については、「[PFLT_CONTEXT_CLEANUP_CALLBACK](#)」を参照してください。

ミニフィルタードライバでは、必要に応じて、フィルターマネージャーに依存してこれらのタスクを実行するのではなく、コンテキストを割り当てて解放するための独自のコールバックルーチンを定義できます。ただし、これはほとんど必要ありません。詳細については、「[PFLT_context_ALLOCATE_callback](#)」と「[PFLT_CONTEXT_FREE_callback](#)」を参照してください。

CTX サンプルミニフィルタードライバから取得された次のコード例では、インスタンス、ファイル、ストリーム、およびファイルオブジェクト (ストリームハンドル) のコンテキストを登録するために使用される、FLT_コンテキスト_登録構造の配列が示されています。

```

const FLT_CONTEXT_REGISTRATION contextRegistration[] =
{
    { FLT_INSTANCE_CONTEXT,          //ContextType
      0,                             //Flags
      CtxContextCleanup,             //ContextCleanupCallback
      CTX_INSTANCE_CONTEXT_SIZE,     //Size
      CTX_INSTANCE_CONTEXT_TAG       //PoolTag
    },
    { FLT_FILE_CONTEXT,              //ContextType
      0,                             //Flags
      CtxContextCleanup,             //ContextCleanupCallback
      CTX_FILE_CONTEXT_SIZE,         //Size
      CTX_FILE_CONTEXT_TAG           //PoolTag
    },
    { FLT_STREAM_CONTEXT,            //ContextType
      0,                             //Flags
      CtxContextCleanup,             //ContextCleanupCallback
      CTX_STREAM_CONTEXT_SIZE,       //Size
      CTX_STREAM_CONTEXT_TAG         //PoolTag
    },
    { FLT_STREAMHANDLE_CONTEXT,      //ContextType
      0,                             //Flags
      CtxContextCleanup,             //ContextCleanupCallback
      CTX_STREAMHANDLE_CONTEXT_SIZE, //Size
      CTX_STREAMHANDLE_CONTEXT_TAG   //PoolTag
    },
    { FLT_CONTEXT_END }
};

```

コンテキストの作成

2020/04/24 • [Edit Online](#)

ミニフィルタードライバーが使用するコンテキストの種類を登録すると、**FltAllocateContext**を呼び出すことによってコンテキストを作成できます。このルーチンは、「[コンテキスト型の登録](#)」で説明されている条件に従って、使用する適切なコンテキスト定義を選択します。

次のコード例では、CTX サンプルミニフィルタードライバーから取得した、Ctxinstancesetupルーチンは**FltAllocateContext**を呼び出してインスタンスコンテキストを作成します。

```
status = FltAllocateContext(  
    FltObjects->Filter,          //Filter  
    FLT_INSTANCE_CONTEXT,       //ContextType  
    CTX_INSTANCE_CONTEXT_SIZE,  //ContextSize  
    NonPagedPool,               //PoolType  
    &instanceContext);          //ReturnedContext
```

CTX サンプルでは、インスタンスコンテキストに対して次のコンテキスト定義が登録されています。

```
{ FLT_INSTANCE_CONTEXT,          //ContextType  
  0,                             //Flags  
  CtxContextCleanup,             //ContextCleanupCallback  
  CTX_INSTANCE_CONTEXT_SIZE,     //Size  
  CTX_INSTANCE_CONTEXT_TAG },    //PoolTag
```

サイズメンバーが定数であるため、これは固定サイズのコンテキスト定義です。(サイズのメンバーが_変数_サイズ_コンテキストの場合は、変数サイズのコンテキスト定義になります)。FLTFL_コンテキスト_登録_正確な_サイズ_一致フラグがフラグメンバーで設定されていないことに注意してください。この場合、**FltAllocateContext**のsize/パラメーターの値がコンテキスト定義のsizeメンバーの値と一致すると、FltAllocateContextは適切な非ページリックからインスタンスコンテキストを割り当てます。表. 値が一致しない場合、FltAllocateContextは STATUS_FLT_コンテキスト_割り当て_見つかりませ_んでした。

FltAllocateContextは、新しいコンテキストの参照カウントを1に初期化します。コンテキストが不要になった場合は、ミニフィルタードライバーがこの参照を解放する必要があります。そのため、**FltAllocateContext**のすべての呼び出しは、**FltReleaseContext**の後続の呼び出しで一致する必要があります。

コンテキストの設定

2020/04/24 • [Edit Online](#)

新しいコンテキストを作成した後、ミニフィルタードライバーはFltSetxxxコンテキストを呼び出すことによってオブジェクトにアタッチできます。ここで、xxxはコンテキストの種類です。

FltSetxxxコンテキストルーチンのOperationパラメーターがFLTに設定されている場合__コンテキスト_設定し、_が存在する場合は_を保持します。また、FltSetxxxコンテキストは、新しく割り当てられたコンテキストをにアタッチします。ミニフィルタードライバーがオブジェクトのコンテキストをまだ設定していない場合にのみオブジェクト。ミニフィルタードライバーによってコンテキストが既に設定されている場合、FltSetxxxコンテキストは、既に_定義されている状態_FLT_を返します。これは、既に NTSTATUS エラーコードであり、既存のコンテキストに置き換わるものではありません。FltSetxxxコンテキストルーチンのoldcontextパラメーターがNULL以外の場合は、既存のコンテキストへのポインターを受け取ります。このポインターが不要になった場合、ミニフィルタードライバーはFltReleaseContextを呼び出すことにより、このポインターを解放する必要があります。

OperationパラメーターがFLTに設定されている場合__コンテキスト_設定します。_が存在する場合、FltSetxxxコンテキストは常に新しいコンテキストをオブジェクトにアタッチします。ミニフィルタードライバーによってコンテキストが既に設定されている場合、FltSetxxxコンテキストによって既存のコンテキストが削除され、新しいコンテキストが設定され、新しいコンテキストの参照カウントがインクリメントされます。OldcontextパラメーターがNULL以外の場合は、削除されたコンテキストへのポインターを受け取ります。このポインターが不要になった場合、ミニフィルタードライバーはFltReleaseContextを呼び出すことにより、このポインターを解放する必要があります。

次のコード例では、CTX サンプルミニフィルタードライバーを使用して、インスタンスコンテキストを作成し、設定します。

```
status = FltAllocateContext(
    FltObjects->Filter,          //Filter
    FLT_INSTANCE_CONTEXT,       //ContextType
    CTX_INSTANCE_CONTEXT_SIZE,  //ContextSize
    NonPagedPool,               //PoolType
    &instanceContext);          //ReturnedContext
...
status = FltSetInstanceContext(
    FltObjects->Instance,        //Instance
    FLT_SET_CONTEXT_KEEP_IF_EXISTS, //Operation
    instanceContext,             //NewContext
    NULL);                      //OldContext

if (instanceContext != NULL) {
    FltReleaseContext(instanceContext);
}
return status;
```

Fltsetinstancecontextを呼び出した後は、FltReleaseContextを呼び出して、FltAllocateContextによって設定された参照カウント (fltsetinstancecontextではない) を解放することに注意してください。これについては、「[コンテキストの解放](#)」で説明しています。

コンテキストの取得

2020/04/24 • [Edit Online](#)

ミニフィルタードライバーによってオブジェクトのコンテキストが設定されると、`Fltgetxxx`コンテキストを呼び出すことによってコンテキストを取得できます。ここで、`xxx`はコンテキストの種類です。

次のコード例では、`SwapBuffers` サンプルミニフィルタードライバーを使用して、`Fltgetvolumecontext`を呼び出してボリュームコンテキストを取得します。

```
status = FltGetVolumeContext(  
    FltObjects->Filter,    //Filter  
    FltObjects->Volume,    //Volume  
    &volCtx);              //Context  
...  
if (volCtx != NULL) {  
    FltReleaseContext(volCtx);  
}
```

`Fltgetvolumecontext`の呼び出しが成功した場合、コンテキストパラメーターは呼び出し元のボリュームコンテキストのアドレスを受け取ります。`Fltgetvolumecontext`は、コンテキストポインターの参照カウントをインクリメントします。このため、このポインターが不要になった場合、ミニフィルタードライバーは`FltReleaseContext`を呼び出して解放する必要があります。

コンテキストの参照

2020/04/24 • [Edit Online](#)

フィルターマネージャーでは、参照カウントを使用してミニフィルタードライバーコンテキストの有効期間を管理します。参照カウントは、コンテキストの状態を示す数値です。コンテキストが作成されるたびに、コンテキストの参照カウントが1つに初期化されます (これはコンテキストへの最初の参照と呼ばれます)。システムコンポーネントによってコンテキストが参照されるたびに、コンテキストの参照カウントが1ずつインクリメントされます。コンテキストが不要になった場合は、その参照カウントがデクリメントされます。正の参照カウントは、コンテキストが使用可能であることを意味します。参照カウントがゼロになると、コンテキストは使用できなくなり、フィルターマネージャーによって最終的に解放されます。

通常、コンテキストへの最初の参照は、オブジェクトが破棄されるときに解放されます。ただし、ミニフィルタードライバーがオブジェクトからコンテキストを削除する必要がある場合、ミニフィルタードライバーは、コンテキストへの最初の参照を解放する必要があります。コンテキストへの最初の参照を安全に解放するために、ミニフィルタードライバーは `Fltdeletecontext` を呼び出します。

ミニフィルタードライバーは、 `Fltreferencecontext` を呼び出してコンテキストの参照カウントをインクリメントすることによって、コンテキストに独自の参照を追加できます。この追加された参照は、最終的に `FltReleaseContext` を呼び出すことによって削除する必要があります。

コンテキストのリリース

2020/04/24 • [Edit Online](#)

ミニフィルタードライバーは、[FltReleaseContext](#)を呼び出すことによってコンテキストを解放します。次のいずれかのルーチンの呼び出しが成功するたびに、[FltReleaseContext](#)の呼び出しによって最終的に一致する必要があります。

[FltAllocateContext](#)

[FltGetInstanceContext](#)

[FltGetFileContext](#)

[FltGetStreamContext](#)

[FltGetStreamHandleContext](#)

[FltGetTransactionContext](#)

[FltGetVolumeContext](#)

[FltReferenceContext](#)

[FltSetXxx](#)コンテキストによって返される*Oldcontext*ポインターと[Fltdeletecontext](#)によって返されるコンテキストポインターも、不要になったときに解放する必要があることに注意してください。

次のコード例では、CTX サンプルミニフィルタードライバーを使用して、インスタンスコンテキストを作成および設定してから、[FltReleaseContext](#)を呼び出します。

```
status = FltAllocateContext(
    FltObjects->Filter,           //Filter
    FLT_INSTANCE_CONTEXT,        //ContextType
    CTX_INSTANCE_CONTEXT_SIZE,   //ContextSize
    NonPagedPool,                //PoolType
    &instanceContext);           //ReturnedContext

...
status = FltSetInstanceContext(
    FltObjects->Instance,         //Instance
    FLT_SET_CONTEXT_KEEP_IF_EXISTS, //Operation
    instanceContext,              //NewContext
    NULL);                       //OldContext

if (instanceContext != NULL) {
    FltReleaseContext(instanceContext);
}
return status;
```

[Fltsetinstancecontext](#)の呼び出しが成功したかどうかに関係なく、[FltReleaseContext](#)が呼び出されることに注意してください。どちらの場合も、呼び出し元は[FltReleaseContext](#)を呼び出して、[FltAllocateContext](#)によって参照セットを解放する必要があります ([fltsetinstancecontext](#)ではありません)。

インスタンスに対してコンテキストが正常に設定されている場合は、[Fltsetinstancecontext](#)によってインスタンスコンテキストに独自の参照が追加されます。したがって、[FltAllocateContext](#)によって設定された参照は不要になり、[FltReleaseContext](#)を呼び出すと削除されます。

[Fltsetinstancecontext](#)の呼び出しが失敗した場合、インスタンスコンテキストには参照が1つだけあります。つまり、[FltAllocateContext](#)によって設定されています。[FltReleaseContext](#)がを返した場合、インスタンスコンテキストの参照カウンタはゼロであり、フィルターマネージャーによって解放されます。

コンテキストの削除

2020/04/24 • [Edit Online](#)

`FltSetXxx`コンテキストへの正常な呼び出しによって設定されたすべてのコンテキストは、最終的には削除する必要があります。ただし、フィルターマネージャーは、アタッチされたオブジェクトが削除されたとき、ミニフィルタードライバーインスタンスがボリュームからデタッチされたとき、またはミニフィルタードライバーがアンロードされたときに、コンテキストを自動的に削除します。そのため、ミニフィルタードライバーでコンテキストを明示的に削除する必要はほとんどありません。

ミニフィルタードライバーでは、`Fltdelete_xxx`コンテキストを呼び出すことによってコンテキストを削除できます。ここで、`xxx`はコンテキストの種類であり、`fltdeletecontext`を呼び出します。

コンテキストは、オブジェクトに対して現在設定されている場合にのみ削除できます。コンテキストは、まだ設定されていない場合、または`FltSetXxx`コンテキストへの正常な呼び出しによって既に置き換えられている場合は、削除できません。

`FltdeleteXxx`コンテキストへの呼び出しでは、古いコンテキストがNULL以外の場合は、`oldcontextl`パラメーターで返されます。`Oldcontextl`パラメーターがNULLの場合、フィルターマネージャーはコンテキストの参照カウントをデクリメントします。これは、ミニフィルタードライバーが未処理の参照を持っていない限り解放されます。

次のコード例は、ストリームコンテキストを削除する方法を示しています。

```
status = FltDeleteStreamContext(  
    FltObjects->Instance,      //Instance  
    FltObjects->FileObject,    //FileObject  
    &oldContext);              //OldContext  
...  
if (oldContext != NULL) {  
    FltReleaseContext(oldContext);  
}
```

この例では、`Fltdeletestreamcontext`はストリームからストリームコンテキストを削除しますが、`Oldcontextl`パラメーターがNULLではないため、コンテキストの参照カウントをデクリメントしません。`Fltdeletestreamcontext`は、`oldcontextl`パラメーターで削除されたコンテキストのアドレスを返します。必要な処理を実行した後、呼び出し元は`FltReleaseContext`を呼び出すことによって、削除されたコンテキストを解放する必要があります。

コンテキストの解放

2020/04/24 • [Edit Online](#)

コンテキストは削除された後に解放され、未処理のすべての参照が解放されます。

この規則には例外が1つあります。コンテキストが作成されているものの、`FltSetXxx`コンテキストを呼び出すことによって設定されていない場合は、削除する必要はありません。参照カウントが0になると解放されます。[\(コンテキストの解放に関するコード例を参照してください\)](#)。

ミニフィルタードライバーによってそのコンテキスト型が登録されると、コンテキストが解放される前に呼び出されるコンテキストクリーンアップコールバックルーチンを各コンテキスト定義に含めることができます。詳細については、「[PFLT_CONTEXT_CLEANUP_CALLBACK](#)」を参照してください。

ファイル システムによるコンテキストのサポート

2020/04/24 • [Edit Online](#)

ファイルコンテキスト (該当する場合)、ストリームコンテキスト、およびファイルオブジェクト (ストリームハンドル) のコンテキストをサポートするには、ファイルシステムで **FSRTL_ADVANCED_FCB_ヘッダー** 構造体を使用する必要があります。すべての Microsoft Windows ファイルシステムはこの構造体を使用します。また、すべてのサードパーティのファイルシステム開発者にもこのような構成を行うことを強くお勧めします。詳細については、「**Fsrtlsetupadvanced ヘッダー**」と「**FSRTL_ADVANCED_FCB_ヘッダー**」を参照してください。

NTFS および FAT ファイルシステムは、ページングファイル、作成前または終了パス、または **IRP_MJ_ネットワーク_クエリ_オープン** 操作のために、ファイル、ストリーム、またはファイルオブジェクトコンテキストをサポートしていません。

ミニフィルタードライバーは、**Fltsupportsstreamcontexts** と **Fltsupportsstreamhandl** テキストをそれぞれ呼び出して、ファイルシステムが特定のファイルオブジェクトのストリームコンテキストとファイルオブジェクトコンテキストをサポートするかどうかを判断できます。

ファイルコンテキストは、Windows Vista 以降で使用できます。

ファイルごとに1つのデータストリームのみをサポートするファイルシステム (FAT など) の場合、ファイルコンテキストはストリームコンテキストに相当します。通常、このようなファイルシステムではストリームコンテキストはサポートされていますが、ファイルコンテキストはサポートしていません。代わりに、フィルタマネージャーは、ファイルシステムのストリームコンテキストの既存のサポートを使用して、このサポートを提供します。これらのファイルシステムに接続されているミニフィルタードライバーのインスタンスの場合、

Fltsupportsfilecontexts は FALSE を返しますが、TRUE を返します (インスタンスパラメーターに有効な NULL 以外の値が渡された場合)。

ファイルでコンテキストの種類がサポートされていない場合、ミニフィルタでそのファイルにその種類のコンテキストをアタッチすることはできません。

ファイルのコンテキストをサポートするには、ファイルシステムで次のことを行う必要があります。

- ファイルコンテキスト構造 (通常はファイルコンテキストブロック (FCB)) に PVOID 型の **FileContextSupportPointer** メンバーを埋め込みます。ファイルシステムは、このメンバーを NULL に初期化する必要があります。
- **Fsrtlsetup** の **Headerex** (**fsrtlsetupadvanced ヘッダー** ではなく) を使用して、ストリームコンテキスト構造を初期化し、**FileContextSupportPointer** メンバーへの有効なポインターを渡します (対応するファイルコンテキストに埋め込まれています)。 **FileContextSupportPointer** パラメーターの構造体)。詳細については、「**FsrtlsetupFSRTL Headerex** と **_ADVANCED_FCB_ヘッダー**」を参照してください。
- **FsRtlTeardownPerFileContexts** を呼び出して、ファイルシステムによってファイルのファイルコンテキスト構造が削除されたときにフィルタ処理およびミニフィルタを適用するすべてのファイルコンテキスト構造を解放します。

ベスト プラクティス

2020/04/24 • [Edit Online](#)

ミニフィルター ドライバーでは、ボリュームごとに 1 つだけのミニフィルター ドライバーのインスタンスを作成する場合は、パフォーマンス向上のためのボリュームのコンテキストではなく、インスタンス コンテキストを使用してください。

ミニフィルター ドライバーはミニフィルター ドライバーのインスタンス コンテキスト、ストリーム内へのポインターを格納することでパフォーマンスを向上したり、ハンドルのコンテキストをストリームできます。

カーネルモードドライバーが開始する操作を作成するために ECPs を IRP_MJ にアタッチ_

2020/04/24 • [Edit Online](#)

次の手順に従って、ECPsを設定し、ファイルに対するIRP_MJ_CREATE操作に接続する必要があります。

1. `FltAllocateExtraCreateParameterList`または`FsRtlAllocateExtraCreateParameterList`ルーチン呼び出しで、`ECP_LIST`構造体にメモリを割り当てます。オペレーティングシステムは、ECP_リストの構造を自動的に解放しません。その代わりに、ECP_リスト構造が割り当てられた後、ミニフィルタードライバーは、`FltFreeExtraCreateParameterList`または`FsRtlFreeExtraCreateParameterList`ルーチン呼び出しで、ecp_リストを解放する必要があります。
2. `FltAllocateExtraCreateParameter`または`FsRtlAllocateExtraCreateParameter`ルーチン呼び出しで、ECP コンテキスト構造にページングされたメモリプールを割り当て、その構造体へのポインターを生成します。
3. `FltInsertExtraCreateParameter`または`FsRtlInsertExtraCreateParameter`ルーチン呼び出しで、ecp のコンテキスト構造を`ecp_LIST`構造体に挿入します。
4. `IoInitializeDriverCreateContext`ルーチン呼び出しで、`_コンテキスト構造を作成_IO_ドライバー`を初期化します。
5. `_コンテキスト構造_作成する IO_ドライバー`を定義します。この定義では、IO_ドライバーの`ExtraCreateParameter`メンバーをポイントし、_コンテキストを`ECP_リスト`構造に作成します。
6. `FltCreateFileEx2`または`IoCreateFileEx`ルーチン呼び出しで、ファイルのIRP_MJ_CREATE操作に ecps をアタッチします。この呼び出しで、`i/o_ドライバー`へのポインターを渡し、`drivercontext`パラメーターに_コンテキスト構造を作成_ます。
7. `FltFreeExtraCreateParameterList`または`FsRtlFreeExtraCreateParameterList`ルーチン呼び出しで、`ECP_リスト`の構造を解放します。

ユーザー定義の ECP

2020/04/24 • [Edit Online](#)

独自の ECP を定義するには、ECP をおよび、ECP を記述するコンテキスト構造体を識別する GUID を定義します。Guid とドライバーのコードでそれらを使用して定義する方法については、次を参照してください。[ドライバーを使用して Guid](#)します。

システム定義の ECP

2020/04/24 • [Edit Online](#)

オペレーティングシステムでは、*Ntifs*ヘッダーファイルに次の *ecps* が定義されています。これらのシステム定義の ECPs は、指定された追加情報を、ファイルに対する **IRP_MJ_CREATE** 操作にアタッチします。ファイルシステムスタックの要素は、追加情報を ECPs に照会できます。

通常、**irp_MJ** を処理してファイルに対して作成操作を実行し、その下のフィルターにファイルを渡すフィルターは、ファイルに対する **irp_MJ_CREATE** 操作に、次のシステム定義の *ecps* をアタッチしてスプーフィングする必要があります。同様に、IRP_MJ を処理および発行するカーネルモードドライバは、ファイルに対して作成操作を実行し、次のシステム定義のすべての ECPs を、IRP_MJ_ファイルに対する作成操作にアタッチして偽装することはできません。次のシステム定義の ECPs は読み取り専用です。情報を取得するためにのみ使用する必要があります。

フィルタードライバが次のシステム定義の ECPs のいずれかをアタッチしないように制限する例外の1つは、フィルタードライバがレイヤーファイルシステムを実装する場合です。これを行うには、ファイルオブジェクトを所有し、独自の **irp_MJ** を発行して、そのフィルターの下にあるファイルに対して、IRP_MJ_create 操作にตอบสนองして、そのファイルオブジェクトでフィルタードライバがサービスを実行することによって、ファイルオブジェクトを所有します。このようなフィルタードライバは、ファイルに対する元の IRP_MJ_作成操作から、フィルタードライバによって発行された IRP_MJ_CREATE 操作に、すべての ECP コンテキスト構造リスト (ECP_リスト) を伝達する必要があります。これらの ECP リストを伝達することにより、フィルタードライバは、IRP_MJ_CREATE 操作を発行するフィルターの下にあるフィルターが、元の IRP_MJ_CREATE 操作のコンテキストを認識していることを確認します。

GUID_ECP_OPLOCK_キー

_キー_ECP_コンテキスト 構造を識別する GUID。この GUID を使用して、oplock キーが open file 要求にアタッチされます。Oplock キーを使用すると、アプリケーション独自の oplock を損なうことなく、同じストリームに対して複数のハンドルを開くことができます。

Oplock キーと oplock キーの詳細については、「[Oplock セマンティクスの概要](#)」を参照してください。

GUID_ECP_ネットワーク_開いている_コンテキスト

ネットワークを識別する GUID **_ECP_コンテキスト** 構造を開き、ネットワークドライブの追加情報をアタッチするために使用されます。また、この GUID は、windows 7 以降のバージョンの Windows で実行され、Windows Vista 上に存在するファイルでネットワーク ECP コンテキストを解釈する必要があるドライバに対して、**_ECP_コンテキスト_V0 構造を開くネットワーク_** も識別します。

_プリフェッチ_の GUID_ECP

_ECP_コンテキスト構造を開く_のプリフェッチ を識別する GUID。

Prefetcher は、キャッシュマネージャーおよびメモリマネージャーと緊密に統合されているオペレーティングシステムのコンポーネントであり、ディスクへのアクセス効率を高め、パフォーマンスを向上させます。他のコンポーネントが prefetcher に干渉する場合、システムのパフォーマンスが低下し、デッドロックが発生する可能性があります。このため、prefetcher は **_ECP_コンテキスト構造を開いているプリフェッチ_** をファイルにアタッチして、ファイルに対して開いている要求を prefetcher が実行することを通知します。このオープンな要求は、**_ECP_コンテキスト_開いているプリフェッチのコンテキストメンバー** によって指定されます。ファイルシステムフィルタードライバなどの他のコンポーネントでは、プリフェッチ_開いている **_ECP_コンテキスト** がファイルに添付されているかどうかを判断し、適切なアクションを実行できます。

_NFS_の GUID_ECP

_ECP_コンテキスト構造を開く_NFS を識別する GUID。ネットワークファイルシステム (NFS) サーバーは、開いているファイル要求に対して、**_ECP_の_開いている NFS** をアタッチします。Nfs サーバーは、NFS サーバーがクライアント要求を満たすために使用するすべての開いているファイル要求で、この GUID を使用します。その後、ファイルシステムスタックは、**_ECP_コンテキスト** が開いているファイルの要求にアタッチされているかどうかを確認できます。NFS の情報に基づいて **_ECP_コンテキストを開く_**、ファイルシステムスタックは、ファイルのオープンを要求したクライアントとその理由を特定できます。

_ECP_SRV_開いている GUID

[_ECP_コンテキスト構造を開く_SRV](#)を識別する GUID。サーバーは、開いているファイル要求に対して_ECP_コンテキスト構造を開いている_SRV をアタッチします。サーバーは、条件付きクライアント要求を満たすためにサーバーが行うすべての開いているファイル要求で、この GUID を使用します。その後、ファイルシステムスタックは、SRV_開いている_ECP_コンテキストが open file 要求にアタッチされているかどうかを判断できます。SRV の情報に基づいて_ECP_コンテキストを開く、ファイルシステムスタックは、ファイルのオープンを要求したクライアントとその理由を特定できます。

GUID_ECP_2_OPLOCK_キー

[_ECP_コンテキスト構造の2つの_キー](#)を識別する GUID。Oplock_キー_ecp_コンテキスト構造と同様に、デュアル oplock_キー_ecp_contextを使用して、open file 要求に oplock キーをアタッチします。_キー_ECP_コンテキストでは、親キーを設定して、ターゲットファイルのディレクトリに対して oplock を指定することもできます。

GUID_ECP_IO_デバイス_ヒント

IO_デバイス_ヒント_ECP_コンテキスト構造を識別する GUID。デバイスヒントは、新しいデバイスへの再解析ターゲットを追跡するときに、名前プロバイダーのミニフィルタードライバーを支援するために使用されます。

_ネットワーク_アプリ_インスタンスの GUID_ECP

[ネットワーク_アプリ_インスタンス_ECP_コンテキスト](#)構造を識別する GUID。フェールオーバークラスター内のクライアントアプリケーションは、クラスター内のノードで開かれた一連のファイルを持つことができます。ファイルオブジェクトは、ネットワーク_アプリ_インスタンス_ECP_コンテキスト構造のインスタンス識別子によってアプリケーションにタグ付けされます。フェールオーバーでは、セカンダリノードは、以前にキャッシュされたアプリケーションインスタンス識別子を使用して、開いているファイルへのクライアントアプリケーションのアクセスを検証できます。

ファイル システム レガシ フィルター ドライバー について

2020/04/24 • [Edit Online](#)

このセクションの情報は、既存のレガシフィルタードライバーを管理する開発者を対象としています。レガシファイルシステムフィルターモデルは、[フィルターマネージャー](#)と[ミニフィルタードライバー](#)に置き換えられました。

NOTE

最適な信頼性とパフォーマンスを得るには、従来のファイルシステムフィルタードライバーではなく、フィルターマネージャーをサポートする[ファイルシステムミニフィルタードライバー](#)を使用します。レガシドライバーをミニフィルタードライバーに移植する方法については、「[レガシフィルタードライバーを移植するためのガイドライン](#)」を参照してください。

ドライバーのロード時期の決定方法

2020/04/24 • [Edit Online](#)

NOTE

最適な信頼性とパフォーマンスを得るには、従来のファイルシステムフィルタードライバーではなく、フィルターマネージャーをサポートする[ファイルシステムミニフィルタードライバー](#)を使用します。レガシドライバーをミニフィルタードライバーに移植する方法については、「[レガシフィルタードライバーを移植するためのガイドライン](#)」を参照してください。

システムのブートシーケンス中にファイルシステムドライバーが読み込まれるタイミングと方法を調べる前に、ドライバーの開始の種類と読み込み順序のグループについて理解しておく必要があります。

ドライバーの開始の種類

カーネルモードドライバーの開始の種類では、ドライバーがシステムの起動時または起動後に読み込まれるかどうかを指定します。次の5つの開始の種類があります。

- SERVICE_BOOT_START (0x00000000)
オペレーティングシステム (OS) ローダーによって開始されたドライバーを示します。ファイルシステムフィルタードライバーは、通常、この開始の種類または SERVICE_DEMAND_START を使用します。Microsoft Windows XP およびそれ以降のシステムでは、新しい[ファイルシステムフィルターの読み込み順序グループ](#)を利用するために、フィルターでこのスタートアップの種類を使用する必要があります。
- SERVICE_SYSTEM_START (0x00000001)
OS の初期化中にドライバーが開始されたことを示します。この開始の種類は、ファイルシステムレコグナイザーによって使用されます。"SERVICE_DISABLED" の下に一覧表示されているファイルシステムを除き、ファイルシステム (ネットワークファイルシステムコンポーネントを含む) は、通常、この種類の開始または SERVICE_DEMAND_START を使用します。このスタートアップの種類は、システムの初期化中に列挙され、システムの読み込みには必要ない PnP デバイスのデバイスドライバーによっても使用されます。
- SERVICE_AUTO_START (0x00000002)
システムの起動時にサービスコントロールマネージャーによって開始されたドライバーを示します。ほとんど使用されません。
- SERVICE_DEMAND_START (0x00000003)
PnP マネージャー (デバイスドライバーの場合) またはサービスコントロールマネージャー (ファイルシステムとファイルシステムフィルタードライバーの場合) のいずれかによって、要求時にドライバーが開始されたことを示します。
- SERVICE_DISABLED (0x00000004)
OS ローダー、サービスコントロールマネージャー、または PnP マネージャーによって開始されていないドライバーを示します。ファイルシステムレコグナイザー (ブートファイルシステムの場合を除く) または (EFS の場合) 別のファイルシステムによって読み込まれるファイルシステムによって使用されます。このようなファイルシステムには、CDFS、EFS、FastFat、NTFS、UDF などがあります。デバッグ中にドライバーを一時的に無効にするためにも使用されます。

指定 (開始の種類を)

ドライバーライターは、次のいずれかの方法で、インストール時にドライバーの開始の種類を指定できます。

- ドライバーの INF ファイルの[addservice](#)ディレクティブによって参照されている、`starttype`エントリの開始の種類を指定します。この方法については、「[ServiceInstall](#)」セクションを参照してください。

- ユーザーモードインストールプログラムからCreateServiceまたはChangeServiceConfigを呼び出すときに、`dwstarttype` パラメーターに必要な開始の種類を渡します。このメソッドについては、Microsoft Windows SDK のドキュメントのCreateServiceとChangeServiceConfigのリファレンスエントリを参照してください。

ドライバーの読み込み順序グループ

SERVICE_BOOT_START および SERVICE_SYSTEM_START の開始の種類では、ドライバーが読み込まれる相対的な順序は、各ドライバーの読み込み順序グループによって指定されます。

開始の種類が SERVICE_BOOT_START のドライバーは、ブート (ブート開始) ドライバーと呼ばれます。Microsoft Windows 2000 以前のシステムでは、ブートドライバーであるほとんどのフィルターは "フィルター" グループに属しています。Microsoft Windows XP 以降のシステムでは、通常、ブートドライバーであるフィルターは、新しい FSFilter の読み込み順序グループの1つに属しています。これらの負荷順序グループの詳細については、「[ファイルシステムフィルタードライバーの読み込み順序グループ](#)」を参照してください。

開始の種類が SERVICE_SYSTEM_START であるドライバーも、それらが属する読み込み順序グループの順序で読み込まれます。ただし、すべてのブートドライバーが読み込まれるまで、システム開始ドライバーは読み込まれません。

NOTE

開始の種類が SERVICE_AUTO_START、SERVICE_DEMAND_START、または SERVICE_DISABLED であるドライバーでは、読み込み順序グループは無視されます。

読み込み順序グループの完全な順序付きリストは、HKEY_LOCAL_MACHINE \system\currentcontrolset\controlレジストリキーのServiceGroupOrderサブキーの下にあります。

SERVICE_BOOT_START および SERVICE_SYSTEM_START のドライバーには、同じ負荷グループの順序が使用されます。ただし、すべての SERVICE_BOOT_START ドライバーは、SERVICE_SYSTEM_START ドライバーが読み込まれる前に読み込まれて起動されます。

読み込み順序グループの指定

ドライバーライターは、次のいずれかの方法で、インストール時にドライバーの読み込み順序グループを指定できます。

- ドライバーの INF ファイルのaddserviceディレクティブによって参照されている、loadordergroupエントリに対して目的の読み込み順序グループを指定する。この方法については、「ServiceInstall」セクションを参照してください。
- ユーザーモードのインストールプログラムからCreateServiceまたはChangeServiceConfigを呼び出すときに、`dwstarttype` パラメーターに必要な開始の種類を渡します。このメソッドについては、Microsoft Windows SDK のドキュメントのCreateServiceとChangeServiceConfigのリファレンスエントリを参照してください。

ドライバーの読み込み順序と読み込み順序のグループの一般的な情報については、「[ドライバーの読み込み順序の指定](#)」を参照してください。

システム起動時のファイル システムの状態

2020/04/24 • [Edit Online](#)

NOTE

最適な信頼性とパフォーマンスを得るには、従来のファイルシステムフィルタードライバーではなく、フィルターマネージャーをサポートする[ファイルシステムミニフィルタードライバー](#)を使用します。レガシドライバーをミニフィルタードライバーに移植する方法については、「[レガシフィルタードライバーを移植するためのガイドライン](#)」を参照してください。

ファイルシステムは、システムの起動プロセス中に初期化されます。具体的には、i/o システムの初期化中に発生します。I/o マネージャーは、グローバルファイルシステムキューを作成し、オペレーティングシステム (OS) ローダーと PnP マネージャーによって読み込まれたファイルシステムとレガシフィルタードライバーを初期化します。

システムブートプロセス

次に、ファイルシステムとレガシフィルタードライバーの開発者にとって重要な、システムブートプロセスの選択部分の概要を示します。

1. システムの起動時に、ローダーが制御をカーネルに転送する前に、OS ローダーによって、ブートファイルシステム、RAW ファイルシステム、および SERVICE_BOOT_START 型のすべてのドライバーが読み込まれます。これらのドライバーは、カーネルが制御を取得するときにメモリ内にあります。

ドライバーは、割り当てられている読み込み順序グループの順に読み込まれます。ファイルシステムフィルターでは、新しいファイルシステムフィルタードライバーの読み込み順序グループのいずれかに割り当てられているものは、他のすべてのフィルタードライバーの前に読み込まれます。これらの負荷順序グループの詳細については、「[ファイルシステムフィルタードライバーの読み込み順序グループ](#)」を参照してください。

次に、"フィルター" 読み込み順序グループ内のすべてのドライバーが読み込まれます。"フィルター" グループには、記憶域フィルタードライバーとファイルシステムフィルタードライバーが含まれており、サードパーティと組み込みのフィルタードライバーが含まれています。

2. I/o マネージャーは、4つのセグメントを持つグローバルファイルシステムキューを作成します。1つは、CD-ROM、ディスク、テープデバイス、およびネットワークファイルシステム用です。その後、各ファイルシステムが登録されると、そのコントロールデバイスオブジェクトが、このキューの適切なセグメントに追加されます。ただし、この時点では、まだ登録されていないファイルシステムは存在しないため、キューは空です。
3. PnP マネージャーは、RAW ファイルシステムとすべての SERVICE_BOOT_START ドライバーの [Driverentry](#) ルーチンを呼び出します。

SERVICE_BOOT_START ドライバーが他のドライバーに依存している場合は、それらのドライバーも読み込まれて起動されます。

PnP マネージャーは、ブートデバイスドライバーの [AddDevice](#) ルーチンを呼び出すことによって、ブートデバイスを起動します。ブートデバイスに子デバイスがある場合は、それらのデバイスが列挙されます。子デバイスは、ドライバーがブート開始ドライバーである場合にも構成され、開始されます。デバイスのドライバーがすべてのブート開始ドライバーでない場合は、PnP マネージャーによってデバイスの devnode が作成されますが、デバイスは起動されません。

この時点で、すべてのブートドライバーが読み込まれ、ブートデバイスが起動されます。

4. PnP マネージャーは、[pnp デバイスツリー](#)を走査し、各 devnode に関連付けられているがまだ実行されていないドライバーを探して読み込みます。

各 PnP デバイスが起動すると、PnP マネージャーはデバイスの子を列挙します (存在する場合)。PnP マネージャーは、子

デバイスを構成し、デバイスドライバを読み込んで、デバイスを起動します。

PnP マネージャーは、ドライバのStarttype、loadordergroup、またはDependenciesの値に関係なく、各デバイスのドライバを読み込みます。

このステップでは、pnp 管理者は、PnP で列挙可能なデバイスのみを構成して起動します。デバイスが PnP で列挙可能でない場合、PnP マネージャーはデバイスを無視し、子デバイスが PnP で列挙可能であっても、そのデバイスの子を列挙しません。

5. PnP マネージャーは、まだ読み込まれていない SERVICE_SYSTEM_START 種類のドライバを読み込んで初期化します。

ファイルシステムレコグナイザー (FsRec) は、現時点で読み込まれています。これは、"ブートファイルシステム" の読み込み順序グループに含まれていますが、FsRec はブートファイルシステムではないことに注意してください。実際のブートファイルシステム (ブートボリュームをマウントしたファイルシステム) は、ブートプロセスの開始時に読み込まれます。

後の SERVICE_SYSTEM_START フェーズでは、"ファイルシステム" の [読み込み順序] グループのファイルシステムが読み込まれます。これには、名前付きパイプファイルシステム (NPFS) とメールスロットファイルシステム (MSFS) が含まれます。これには、NTFS、FAT、CDFS、UDF などのメディアベースのファイルシステムは含まれません。

"ネットワーク" 読み込み順序グループに含まれるネットワークファイルシステムも、このフェーズで読み込まれます。

6. ブート時に読み込まれるすべてのドライバが初期化されると、i/o マネージャーは、ドライバを含むすべてのドライバの再初期化ルーチンを呼び出します。再初期化ルーチンは、ブートプロセスのこの時点で追加の処理時間を指定する必要があるブートドライバによって登録されるコールバックルーチンです。再初期化ルーチンは、[IoRegisterBootDriverReinitialization](#)または[IoRegisterDriverReinitialization](#)を呼び出すことによって登録されます。
7. サービスコントロールマネージャーは、まだ読み込まれていない種類の SERVICE_AUTO_START のドライバを読み込みます。

ファイルシステムレコグナイザー

システムの起動後、システムに接続されているすべてのボリュームの記憶装置ドライバが読み込まれ、起動されます。ただし、すべての組み込みファイルシステムが読み込まれるわけではありません。すべてのファイルシステムボリュームがマウントされるわけではありません。ファイルシステムレコグナイザー (FsRec) は、必要に応じてこれらのタスクを実行してIRP_MJ_CREATE要求を処理します。

FsRec は、システム起動時の SERVICE_SYSTEM_START フェーズで読み込まれます。これは、"ブートファイルシステム" の読み込み順序グループに含まれていますが、FsRec はブートファイルシステムではないことに注意してください。実際のブートファイルシステム (ブートボリュームをマウントしたファイルシステム) は、ブートプロセスの開始時に読み込まれます。

ストレージ デバイス スタック、ストレージ ボリューム、ファイル システム スタック

2020/04/24 • [Edit Online](#)

NOTE

最適な信頼性とパフォーマンスを得るには、従来のファイルシステムフィルタードライバーではなく、フィルターマネージャーをサポートする[ファイルシステムミニフィルタードライバー](#)を使用します。レガシドライバーをミニフィルタードライバーに移植する方法については、「[レガシフィルタードライバーを移植するためのガイドライン](#)」を参照してください。

ファイルシステムのレガシフィルタードライバーがファイルシステムとボリュームにどのようにアタッチされるかを調べる前に、記憶域デバイススタック、記憶域ボリューム、およびファイルシステムスタック間の関係を理解する必要があります。

ストレージ デバイス スタック

ほとんどの記憶装置ドライバーは pnp デバイスドライバーであり、PnP マネージャーによって読み込まれて管理されます。記憶装置は、コンピューター上の物理デバイスまたは論理デバイスごとに、デバイスノード (*devnode*) を含む PnP デバイスツリーで表されます。ファイルシステムとファイルシステムフィルタードライバーは PnP デバイスドライバーではないことに注意してください。そのため、PnP [デバイスツリー](#)にはそれらの devnodes が含まれていません。

特定の記憶装置の devnode には、デバイスの *記憶装置スタック*が含まれます。これは、デバイスの記憶装置ドライバーを表す、アタッチされたデバイスオブジェクトのチェーンです。記憶域デバイス (ディスクなど) には論理ボリューム (パーティションまたはダイナミックボリューム) が1つ以上含まれている場合があるため、多くの場合、記憶域デバイススタック自体はスタックよりもツリーのように見えます。このツリーのルートは、記憶域アダプターまたは記憶域スタックと統合されている別のデバイススタックの機能デバイスオブジェクト (FDO) です。このツリーのリーフは、ファイルシステムボリュームをマウントできる論理ボリューム (*記憶域ボリューム*とも呼ばれます) の物理デバイスオブジェクト (pdos) です。

一般的な記憶装置スタックの図と説明については、「[記憶装置設計ガイド](#)」の次のセクションを参照してください。

- [SCSI HBA のデバイスオブジェクトの例](#)
- [IEEE 1394 コントローラーのデバイスオブジェクトの例](#)

ストレージ ボリューム

*ボリューム*は、記憶装置 (固定ディスク、フロッピーディスク、cd-rom など) で、ディレクトリとファイルを格納するようにフォーマットされています。大きなボリュームは、パーティションとも呼ばれる複数の *論理ボリューム*に分けることができます。各論理ボリュームは、NTFS、FAT、CDFS など、特定のメディアベースのファイルシステムで使用するようフォーマットされています。

記憶域ボリューム(*記憶装置オブジェクト*) は、システムの論理ボリュームを表す、デバイスオブジェクト (通常は物理デバイスオブジェクト (PDO)) です。ストレージデバイスオブジェクトは記憶装置スタックに存在しますが、必ずしもスタック内の最上位のデバイスオブジェクトであるとは限りません。

ファイルシステムが記憶域ボリュームにマウントされると、ファイルシステムのボリュームを表すファイルシステムボリュームデバイスオブジェクト (VDO) が作成されます。ファイルシステム VDO は、*ボリュームパラメーターブロック*(vpb) と呼ばれる共有オブジェクトを使って、ストレージデバイスオブジェクトにマウントされます。

マウントマネージャー

マウントマネージャーは、ボリューム名、ドライブ文字、ボリュームマウントポイントなどの記憶域ボリューム情報の管理を行う i/o システムの一部です。新しい記憶域ボリュームがシステムに追加されると、次のいずれかの方法でマウントマネージャーに到着が通知さ

れます。

- ストレージボリュームを作成したクラスドライバーは、MOUNTDEV_MOUNTED_DEVICE_GUID インターフェイスクラスに新しいインターフェイスを登録するために [IoRegisterDeviceInterface](#) を呼び出します。これが発生すると、プラグアンドプレイデバイスインターフェイスの通知メカニズムによって、マウントマネージャーに対してシステムでのボリュームの到着が通知されます。
- 記憶域ボリュームのドライバーは、マウントマネージャーに IRP_MJ_DEVICE_CONTROL 要求を送信して、i/o 制御コードの [IOCTL_MOUNTMGR_VOLUME_ARRIVAL_NOTIFICATION](#) を指定します。この要求は、[IoBuildDeviceIoControlRequest](#) を呼び出すことによって作成できます。

一意のボリューム名

マウントマネージャーは、ボリュームドライバーに次の情報を照会することによって、新しい記憶域ボリュームの到着に応答します。

- システムオブジェクトツリーのデバイスディレクトリにある、ボリュームの非永続的なデバイスオブジェクト名 (またはターゲット名) (例: "\Device\HarddiskVolume1")
- ボリュームのグローバル意識別子 (GUID)。一意の *ボリューム名* とも呼ばれます。
- ドライブ文字など、ボリュームに対して推奨される永続的なシンボリックリンク名 (たとえば、"\ dosdev%")

ストレージドライバーとマウントマネージャーの相互作用の詳細については、「[記憶域クラスドライバーでのマウントマネージャー要求のサポート](#)」を参照してください。

ファイル システム スタック

ファイルシステムドライバーは、コントロールデバイスオブジェクト (CDO) とボリュームデバイスオブジェクト (VDO) という2種類のデバイスオブジェクトを作成します。ファイルシステムスタックは、これらのデバイスオブジェクトの1つと、それにアタッチされているファイルシステムフィルタードライバーのすべてのフィルタードバイスオブジェクトで構成されます。ファイルシステムのデバイスオブジェクトは、常にスタックの一番下を形成します。

ファイルシステム CDOs

ファイルシステムの CDOs は、個々のボリュームではなく、ファイルシステム全体を表し、グローバルファイルシステムキューに格納されます。ファイルシステムによって、Driverentryルーチンに1つ以上の名前付き cdos が作成されます。たとえば、FastFat は2つの CDOs を作成します。1つは固定メディア用で、もう1つはリムーバブルメディア用です。CDFFS は、リムーバブルメディアしかないため、CDO を1つだけ作成します。

ファイルシステムの CDOs には名前を付ける必要があります。これは、ファイルシステムフィルタードライバーと多くのカーネルモードサポートルーチンが、それらを区別する手段として、VDOs と CDOs の違いに依存しているためです。

ファイルシステム VDOs

ファイルシステム VDOs は、ファイルシステムによってマウントされたボリュームを表します。ファイルシステムはボリュームをマウントするときに VDO を作成し、通常はボリュームマウント要求に応答します。CDO とは異なり、VDO は常に特定の論理または物理記憶装置に関連付けられています。

NOTE

CDOs とは異なり、VDOs に名前を付けることはできません。これは、ボリュームデバイスオブジェクトに名前を付けるとセキュリティホールが作成されるためです。

ボリュームの認識方法

2020/04/24 • [Edit Online](#)

NOTE

最適な信頼性とパフォーマンスを得るには、従来のファイルシステムフィルタードライバーではなく、フィルターマネージャーをサポートする[ファイルシステムミニフィルタードライバー](#)を使用します。レガシドライバーをミニフィルタードライバーに移植する方法については、「[レガシフィルタードライバーを移植するためのガイドライン](#)」を参照してください。

ボリュームマウントプロセスは、通常、次のように論理ボリューム（つまりパーティションまたはダイナミックボリューム）上のファイルを開く要求によってトリガーされます。

1. ユーザーアプリケーションは、`CreateFile`を呼び出してファイルを開きます。またはカーネルモードドライバーは、`Zwcreatefile`または`IoCreateFileHint`を呼び出します。
2. I/O マネージャーは、どの論理ボリュームが要求のターゲットであるかを判断し、そのデバイスオブジェクトがマウントされているかどうかを確認します。VPB_MOUNTED フラグが設定されている場合は、ファイルシステムによってボリュームがマウントされています。
3. システムブート以降にファイルシステムによってボリュームがマウントされていない場合（つまり、VPB_MOUNTED フラグが設定されていない場合）、I/O マネージャーは、ボリュームを要求する可能性のある各ファイルシステムにボリュームマウント（`IRP_MJ_FILE_SYSTEM_CONTROL`、`IRP_MN_MOUNT_VOLUME`）要求を送信します。

システムを起動した後でも、すべての組み込みファイルシステムが読み込まれるとは限りません。（「[システムブート中のファイルシステムの動作](#)」を参照してください）。まだ読み込まれていない組み込みのファイルシステムの場合、I/O マネージャーはボリュームマウント要求をファイルシステムレコグナイザー（FsRec）に送信します。これにより、これらのファイルシステムに代わってボリュームブートセクターがチェックされます。

まだ読み込まれていないファイルシステムによってボリュームがフォーマットされたことが FsRec によって判断された場合、I/O マネージャーは、ファイルシステムを読み込む読み込みファイルシステム（`IRP_MJ_FILE_SYSTEM_CONTROL`、`IRP_MN_LOAD_FILE_SYSTEM`）要求を fsrec に送信することによって応答します。その後、I/O マネージャーが元のボリュームマウント要求をファイルシステムに送信します。

4. ボリュームのマウント要求を受信する各ファイルシステムは、ボリュームのブートセクターを調べて、ボリュームの形式とその他の情報がその特定のファイルシステムでフォーマットされたことを示しているかどうかを判断します。形式が一致すると、ファイルシステムによってボリュームがマウントされます。

次のセクションでは、ファイルシステムが認識した後にボリュームをマウントする方法について説明します。

ボリュームのマウント方法

ボリュームマウントの例

ボリュームのマウント方法

2020/04/24 • [Edit Online](#)

NOTE

最適な信頼性とパフォーマンスを得るには、従来のファイルシステムフィルタードライバーではなく、フィルターマネージャーをサポートする[ファイルシステムミニフィルタードライバー](#)を使用します。レガシドライバーをミニフィルタードライバーに移植する方法については、「[レガシフィルタードライバーを移植するためのガイドライン](#)」を参照してください。

ボリュームが[認識された](#)後、ボリュームがマウントされる方法は、ファイルシステムと、以前にボリュームがマウントされているかどうかによって異なります。

ファイルシステムは、新しいボリュームのボリュームマウント要求を受信すると、ボリュームのボリュームデバイスオブジェクト (VDO) を作成します。VDO は、DEVICE_OBJECT と、オプションのファイルシステムで定義されたデバイス拡張機能で構成されます。新しく作成された VDO は、新しい (または再マウントされた) ボリュームのファイルシステムボリュームスタックのベースを形成します。

ファイルシステムは、対応する記憶装置オブジェクトの VDO をボリュームパラメーターブロック (VPB) に関連付けて、ボリュームをマウントし、VPB に VPB_MOUNTED フラグを設定します。

ファイルシステムによってボリュームがマウントされると、ファイルシステムフィルタードライバーを新しいファイルシステムボリュームスタックの一番上にアタッチできます。ファイルシステムに送信された i/o 要求は、最初にボリュームスタックの一番上のファイルシステムフィルターデバイスオブジェクトに自動的に送信されます。ただし、ファイルシステムフィルターはボリュームスタックからデタッチするだけで済みます。i/o マネージャーが高速 i/o デタッチ要求を送信すると、ボリュームが削除されようとしていることをボリュームスタック上に通知します。

例については、[ボリュームマウントの例](#)を参照してください。

NOTE

ボリュームの記憶装置オブジェクトは記憶装置スタックに存在しますが、必ずしもスタック内の最上位のデバイスオブジェクトであるとは限りません。さらに、ボリュームがマウントされた後でも、記憶域フィルタードライバーは記憶域スタックの一番上に接続できます。ドライバーの作成者は、ファイルシステムが VDO からストレージデバイススタックに IRP を送信するときに、スタック内の最上位のデバイスオブジェクトではなく、ボリュームのストレージデバイスオブジェクトにその IRP を送信することに注意してください。(ただし、i/o マネージャーが IRP を直接ストレージスタックに送信し、ファイルシステムをバイパスした場合、その IRP はスタック内の最上位のデバイスオブジェクトに送信されます)。

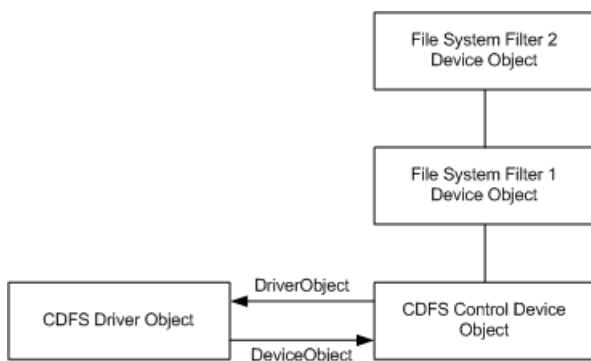
ボリュームのマウントの例

2020/04/24 • [Edit Online](#)

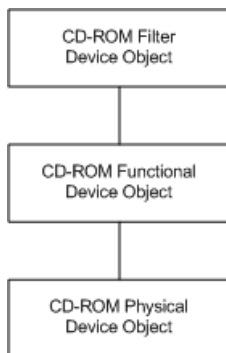
NOTE

最適な信頼性とパフォーマンスを得るには、従来のファイルシステムフィルタードライバーではなく、フィルタマネージャーをサポートする [ファイルシステムミニフィルタードライバー](#) を使用します。レガシドライバをミニフィルタードライバーに移植する方法については、「[レガシフィルタードライバーを移植するためのガイドライン](#)」を参照してください。

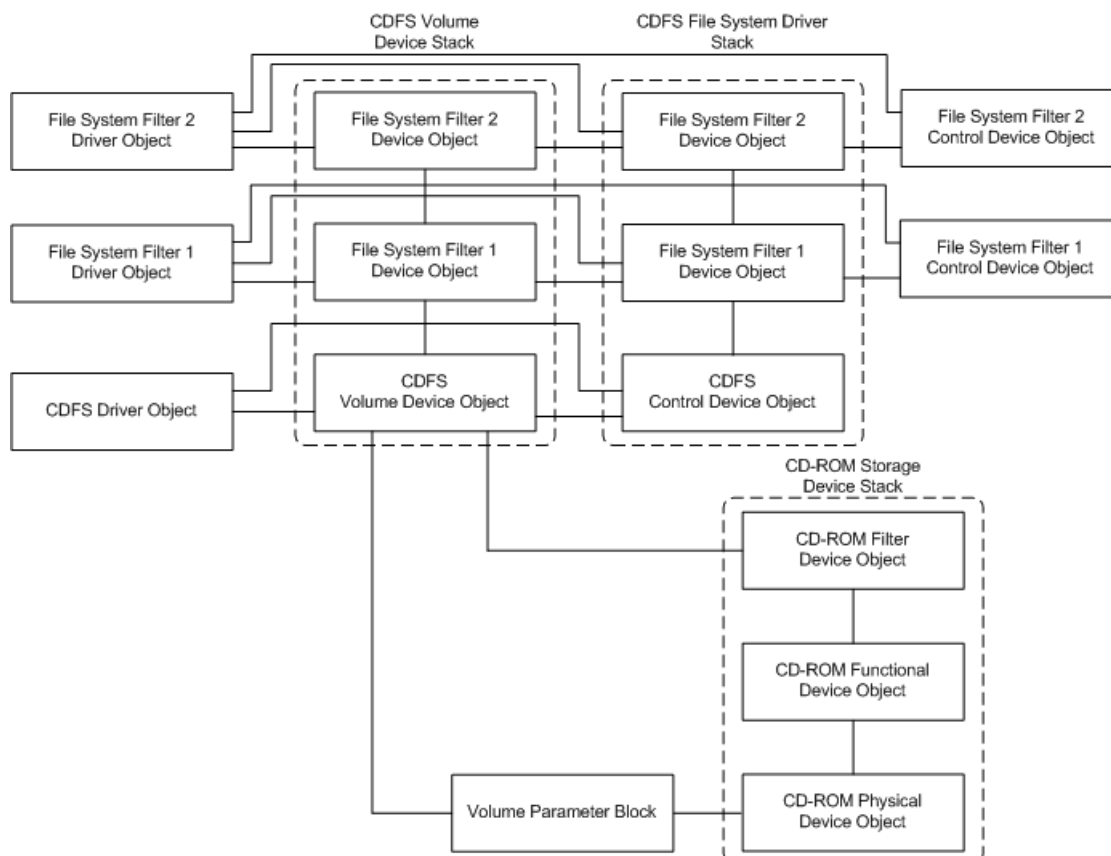
次の図は、ボリュームをマウントする前に、CDFS がどのように見えるかを示しています。この例では、2つのフィルタが CDF コントロールデバイスオブジェクトに関連付けられています。(注: CDFS コントロールデバイスオブジェクトを含むグローバルファイルシステムキューは表示されません)。



次の図は、まだ CDFS ボリュームとしてマウントされていない CD-ROM 記憶装置の一般的なドライバースタックを示しています。



次の図は、CDFS ファイルシステムによって CD-ROM デバイスにボリュームがマウントされた後のファイルシステムドライバースタック、ボリュームスタック、および CD-ROM 記憶装置スタックの外観を示しています。



前の図に関する注意事項を次に示します。

- CDFS コントロールデバイスオブジェクトは、ファイルシステムドライバースタックのベースを形成します。ストレージデバイスにマウントされていないこのスタックは、Irp を直接受信できます。また、ファイルシステムフィルターデバイスオブジェクトを含めることもできます。フィルターは、ボリュームのマウント ([IRP_MJ_FILE_SYSTEM_CONTROL](#)、[IRP_MN_MOUNT_VOLUME](#)) 要求を監視するために、ファイルシステムコントロールのデバイスオブジェクトにアタッチします。ファイルシステムコントロールデバイスオブジェクトには名前を付ける必要があります。これにより、という名前ではないファイルシステムボリュームデバイスオブジェクトと区別されます。
- 図に示すように、CDFS ボリュームがマウントされた後で、CD-ROM 記憶装置スタックの一番上に2番目の記憶域フィルターをアタッチすることもできますが、このフィルターはファイルシステムスタックからストレージ d に渡される Irp を受信しません。e スタック。ただし、ストレージデバイススタックに直接送信される任意の Irp を受信します。
- ファイルシステムによってボリュームがマウントされた後も、記憶装置スタックが Irp を直接受信できることに注意してください。具体的には、電源 Irp ([IRP_MJ_POWER](#)) は、常にストレージデバイススタックに直接送信され、ファイルシステムスタックには送信されません。(したがって、たとえば、ファイルシステムフィルタードライバーは、Driverentryルーチンで [IRP_MJ_POWER](#) のディスパッチルーチンを登録してはなりません)。

ただし、PnP Irp ([IRP_MJ_PNP](#)) はどちらのスタックにも送信できます。ファイルシステムボリュームの上にチェーンされているフィルタードライバーは、既定では、これらの Irp を次の下位のドライバーに渡す必要があります。これは、ファイルシステムのボリュームデバイスが Irp を記憶装置スタックに渡すことができるようにするためです。

ファイル システム フィルター ドライバーのインストール

2020/04/24 • [Edit Online](#)

Microsoft Windows XP と以降のオペレーティング システムでは、INF ファイルと、インストール アプリケーションを使用して、ファイル システム フィルター ドライバーをインストールする必要があります。(Windows 2000 および以前のオペレーティング システムでは、フィルター ドライバーがよくインストール サービス コントロール マネージャーによって)。

今後、INF ベースのインストールは、ファイル システム フィルター ドライバーの Windows ハードウェア認定キットの要件を満たしていると想定されます。「INF ベースのインストール」の意味をのみファイルをコピーして、レジストリの情報を格納する INF ファイルを使用する必要がありますに注意してください。INF ファイルのみを使用して、全体の製品をインストールする必要はありませんし、提供する必要はありません、「[インストールを右クリックして](#)」ドライバーのオプション。

このセクションの内容:

[ファイル システム フィルター ドライバーの INF ファイルを作成します。](#)

[ファイル システム フィルター ドライバーの読み込み順序グループ](#)

[ファイル システム フィルター ドライバーのクラスおよびクラスの Guid](#)

[INF ファイルを使用して、ファイル システム フィルター ドライバーをインストールするには](#)

[ファイル システム フィルター ドライバーをアンインストールする INF ファイルを使用します。](#)

ファイル システム フィルター ドライバー用の INF ファイルの作成

2020/05/05 • [Edit Online](#)

Windows セットアップおよびデバイスインストーラーサービス (総称して「[setupapi.log](#)」といいます) は、Windows セットアップとドライバーのインストールを制御する機能を提供します。インストールプロセスは INF ファイルによって制御されます。

ファイルシステムフィルタードライバーの INF ファイルは、Setupapi.log がドライバーのインストールに使用する指示を提供します。INF ファイルは、ドライバーを実行するために必要なファイル、およびドライバーファイルのソースディレクトリと宛先ディレクトリを指定するテキストファイルです。INF ファイルには、Setupapi.log によってレジストリに格納されるドライバー構成情報 (ドライバーの開始の種類や読み込み順序グループなど) も含まれています。

INF ファイルとその作成方法の詳細については、「[Inf ファイルおよび inf ファイルの作成](#)」セクションと「[ディレクティブ](#)」を参照してください。ドライバーの署名に関する一般的な情報については、「[ドライバーの署名](#)」を参照してください。

1つの INF ファイルを作成して、複数のバージョンの Windows オペレーティングシステムにドライバーをインストールすることができます。このような INF ファイルの作成の詳細については、「[複数のプラットフォームおよびオペレーティングシステム用の Inf ファイルの作成](#)」および「[国際対応の Inf ファイルの作成](#)」を参照してください。

64ビットバージョンの Windows Vista 以降では、ファイルシステムドライバー (ファイルシステム、レガシフィルター、ミニフィルタードライバー) などの非 PnP (プラグアンドプレイ) ドライバーを含むすべてのカーネルモードコンポーネントが、読み込んで実行するために署名されている必要があります。これらのバージョンの Windows オペレーティングシステムでは、次の一覧には、ファイルシステムフィルタードライバーに関連する情報が含まれています。

- ファイルシステムドライバーを含む、PnP 以外のドライバーの INF ファイルに[は、製造元]または[モデル]のセクションを含める必要はありません。
- [SignTool](#) コマンドラインツールは、WDK インストールディレクトリの\bin\selfsign ディレクトリにあります。このツールを使用すると、ドライバー SYS 実行可能ファイルに直接 "sign" を挿入できます。パフォーマンス上の理由から、ブート開始ドライバーには、埋め込み署名が含まれている必要があります。
- INF ファイルを指定すると、[Inf2Cat](#) コマンドラインツールを使用して、ドライバーパッケージのカタログ (.cat) ファイルを作成できます。[WHQL](#) ロゴ署名を受け取ることができるのは、カタログファイルだけです。
- 管理者特権では、署名されていないドライバーを Windows Vista 以降の x64 ベースのシステムにインストールすることができます。ただし、ドライバーは署名されていないため (そのため実行) に失敗します。
- 64ビットバージョンの Windows Vista の運転署名プロセスを含む、運転署名プロセスの詳細については、「[カーネルモードコード署名チュートリアル](#)」を参照してください。
- カスタムカーネルモードの開発ツールを含む、すべてのカーネルモードコンポーネントに署名する必要があります。詳細については、「[開発およびテスト中のドライバーへの署名 \(Windows Vista 以降\)](#)」を参照してください。

INF ファイルを使用してレジストリから情報を読み取ったり、ユーザーモードアプリケーションを起動したりすることはできません。

INF ファイルを作成した後、通常はセットアップアプリケーションのソースコードを記述します。セットアップアプリケーションは、ユーザーモードのセットアップ関数を呼び出して、INF ファイル内の情報にアクセスし、インストール操作を実行します。

独自のフィルタードライバーの INF ファイルを作成するには、サンプルファイルシステムフィルタードライバーの INF ファイルをテンプレートとして使用します。[InfVerif](#) ツールを使用して、INF ファイルの構文を確認できます。

ファイルシステムフィルタードライバーの INF ファイルには、通常、次のセクションが含まれています。

- バージョン (必須)

- DestinationDirs (省略可能ですが推奨)
- SourceDisksNames (必須)
- SourceDisksFiles (必須)
- DefaultInstall (必須)
- DefaultInstall (必須)
- ServiceInstall (必須)
- DefaultUninstall (省略可能)
- DefaultUninstall.Services (省略可能)
- 文字列 (必須)

Version セクション (必須)

Version セクションでは、次のコード例に示すように、フィルターの種類によって決定されるクラスと GUID を指定します。

```
[Version]
Signature    = "$WINDOWS NT$"
Class        = "ActivityMonitor"
ClassGuid     = {b86dff51-a31e-4bac-b3cf-e8cfe75c9fc2}
Provider     = %Msft%
DriverVer    = 08/28/2000,1.0.0.1
CatalogFile =
```

次の表に、**[バージョン]** セクションでファイルシステムフィルタードライバによって指定される値を示します。

項目	値
■	"\$WINDOWS NT \$"
■	「 ファイルシステムフィルタードライバークラスとクラス guid 」を参照してください。
ClassGuid	「 ファイルシステムフィルタードライバークラスとクラス guid 」を参照してください。
■	独自の INF ファイルでは、Microsoft 以外のプロバイダーを指定する必要があります。
DriverVer	「 INF DriverVer ■」を参照してください。
CatalogFile	このエントリを空白のままにします。将来、署名されたドライバの WHQL が提供したカタログファイルの名前が含まれます。

DestinationDirs セクション (省略可能ですが推奨)

Destinationdirs セクションでは、フィルタードライバとアプリケーションファイルのコピー先となるディレクトリを指定します。

このセクションとServiceinstallセクションでは、システム定義の数値を使用して、既知のシステムディレクトリを指定できます。これらの値の一覧については、「**[INF DestinationDirs](#)**」セクションを参照してください。次のコード例では、値 "12" は Drivers ディ

レクトリ (% windir%\system32\Drivers) を参照し、値 "10" は Windows ディレクトリ (% windir%) を参照します。

```
[DestinationDirs]
DefaultDestDir      = 12
MyLegacyFilter.DriverFiles = 12
MyLegacyFilter.UserFiles  = 10,MyLegacyFilter
```

SourceDisksNames セクション (必須)

Sourcedisksnames セクションでは、使用する配布メディアを指定します。

次のコード例では、**Sourcedisksnames** セクションに1つのディストリビューションメディアが一覧表示されます。メディアの一意の識別子は1です。メディアの名前は、INF ファイルの**Strings** セクションで定義されている % Disk1% トークンによって指定されます。

```
[SourceDisksNames]
1 = %Disk1%
```

SourceDisksFiles セクション (必須)

Sourcedisksfiles セクションでは、コピーするファイルの場所と名前を指定します。

次のコード例では、[**Sourcedisksfiles**] セクションに、ドライバー用にコピーされるファイルが一覧表示され、一意の識別子が 1 (この識別子は INF ファイルの**Sourcedisksfiles** セクションで定義されています) であるメディアでファイルが見つかることを指定しています。

```
[SourceDisksFiles]
myLegacyFilter.exe = 1
myLegacyFilter.sys = 1
```

DefaultInstall セクション (必須)

DefaultInstall セクションでは、**CopyFiles** ディレクティブにより、ファイルシステムフィルタードライバーのドライバーファイルとユーザアプリケーションファイルが、**destinationdirs** セクションで指定されている宛先にコピーされます。

Note **CopyFiles** ディレクティブは、カタログファイルまたは INF ファイル自体を参照してはならないことに注意してください。これらのファイルは、Setupapi.log によって自動的にコピーされます。

1つの INF ファイルを作成して、複数のバージョンの Windows オペレーティングシステムにドライバーをインストールすることができます。この種類の INF ファイルは、オペレーティングシステムのバージョンごとに、追加の**DefaultInstall**、**DefaultInstall**、**Defaultuninstall**、および**defaultuninstall. services** セクションを作成することによって作成されます。各セクションには、適用されるオペレーティングシステムのバージョンを指定する**装飾**(例、. ntx86、. ntia64、または nt) が付いています。この種類の INF ファイルの作成の詳細については、「**複数のプラットフォームおよびオペレーティングシステム用の Inf ファイルの作成**」を参照してください。

次のコード例では、**CopyFiles** ディレクティブを使用して、INF ファイルの MyLegacyFilter ファイルおよび MyLegacyFilter セクションに一覧表示されているファイルをコピーします。

```
[DefaultInstall]
OptionDesc = %MyLegacyFilterServiceDesc%
CopyFiles = MyLegacyFilter.DriverFiles, MyLegacyFilter.UserFiles
```

DefaultInstall セクション (必須)

DefaultInstall セクションには、特定のドライバーのサービスが読み込まれる方法とタイミングを制御する**Addservice** ディレクティブが含まれています。

次のコード例では、**addservice** ディレクティブによって、MyLegacyFilter サービスがオペレーティングシステムに追加されます。%

MyLegacyFilterServiceName% トークンには、INF ファイルの**Strings**セクションで定義されているサービス名の文字列が含まれています。MyLegacyFilter は、ドライバの**Serviceinstall**セクションの例の名前です。

```
[DefaultInstall.Services]
AddService = %MyLegacyFilterServiceName%,MyLegacyFilter.Service
```

ServiceInstall セクション (必須)

Serviceinstallセクションでは、レジストリにサブキーまたは値の名前を追加し、値を設定します。**Serviceinstall**セクションの名前は、**DefaultInstall**セクションの**addservice**ディレクティブに含まれている必要があります。

次のコード例は、MyLegacyFilter サンプルドライバの**Serviceinstall**セクションを示しています。

```
[MyLegacyFilter.Service]
DisplayName = %MyLegacyFilterServiceName%
Description = %MyLegacyFilterServiceDesc%
ServiceBinary = %12%\myLegacyFilter.sys
ServiceType = 2 ; SERVICE_FILE_SYSTEM_DRIVER
StartType = 3 ; SERVICE_DEMAND_START
ErrorControl = 1 ; SERVICE_ERROR_NORMAL
LoadOrderGroup = "FSFilter Activity Monitor"
AddReg = MyLegacyFilter.AddRegistry
```

DisplayNameエントリは、サービスの名前を指定します。前の例では、サービス名の文字列は、INF ファイルの**Strings**セクションで定義されている% MyLegacyFilterServiceName% トークンによって指定されています。

Descriptionエントリは、サービスを説明する文字列を指定します。前の例では、この文字列は、INF ファイルの**Strings**セクションで定義されている% MyLegacyFilterServiceDesc% トークンによって指定されています。

Servicebinaryエントリは、サービスの実行可能ファイルへのパスを指定します。前の例では、値12は Drivers ディレクトリ (%windir%\system32\drivers) を示しています。

ServiceTypeエントリは、サービスの種類を指定します。次の表に、**ServiceType**に使用できる値と、対応するサービスの種類を示します。

I	II
0x00000001	SERVICE_KERNEL_DRIVER (デバイスドライバサービス)
0x00000002	SERVICE_FILE_SYSTEM_DRIVER (ファイルシステムまたはファイルシステムフィルタードライバサービス)
0x00000010	SERVICE_WIN32_OWN_PROCESS (独自のプロセスで実行される Microsoft Win32 サービス)
0x00000020	SERVICE_WIN32_SHARE_PROCESS (プロセスを共有する Win32 サービス)

ファイルシステムフィルタードライバの**ServiceType**エントリは、_常_に_サービスファイルシステムドライバに設定する必要があります。

Startttypeエントリは、サービスをいつ開始するかを指定します。次の表に、**Startttype**とそれに対応する開始の種類に使用できる値を示します。

【	【【
0x00000000	SERVICE_BOOT_START
0x00000001	SERVICE_SYSTEM_START
0x00000002	SERVICE_AUTO_START
0x00000003	SERVICE_DEMAND_START
0x00000004	SERVICE_DISABLED

これらの開始の種類の詳細については、「[ドライバーが読み込まれるタイミングを決定する](#)」を参照してください。

ドライバーの開始の種類が [サービス_ブート_開始] (つまり、ドライバーがブート開始ドライバーである) の場合は、**loadordergroup** エントリが、開発しているフィルターの種類に適していることを確認する必要があります。読み込み順序グループを選択するには、「[ファイルシステムフィルタードライバーの読み込み順序グループ](#)」を参照してください。また、x64 ベースの Windows Vista システム以降では、ブート開始ドライバーのバイナリイメージファイルには、埋め込み署名が含まれている必要があります。この要件により、システムブートのパフォーマンスが最適になります。詳細については、「[カーネルモードコード署名のチュートリアル](#)」を参照してください。

ドライバーが読み込まれるタイミングを、**Starttype** および **loadordergroup** のエントリがどのように決定するかについては、「[ドライバーが読み込まれるタイミングを確認する方法](#)」を参照してください。

Errorcontrol エントリは、システムの起動時にサービスを開始できなかった場合に実行するアクションを指定します。次の表に、**errorcontrol** に使用できる値と、それに対応するエラー制御値の一覧を示します。

【	【【【【
0x00000000	SERVICE_ERROR_IGNORE (エラーをログに記録し、システムの起動を続行します。)
0x00000001	SERVICE_ERROR_NORMAL (エラーをログに記録し、ユーザーにメッセージを表示して、システムの起動を続行します。)
0x00000002	SERVICE_ERROR_SEVERE (レジストリの [前回の制御セット] に切り替えて、システムの起動を続行します。)
0x00000003	SERVICE_ERROR_CRITICAL (システムスタートアップがレジストリの正常でないコントロールセットを使用していない場合は、[正常起動時] に切り替えて、もう一度やり直してください。まだ起動できない場合は、バグチェックルーチンを実行します。システムを起動するために必要なドライバーだけが、INF ファイルにこの値を指定する必要があります。)

Loadordergroup エントリは、開発中のファイルシステムフィルタードライバーの種類に適した読み込み順序グループに設定する必要があります。読み込み順序グループを選択するには、「[ファイルシステムフィルタードライバーの読み込み順序グループ](#)」を参照してください。

AddReg **ディレクティブ** は、新しくインストールされたサービスのレジストリに格納される情報を含む、1 つまたは複数の INF ライター

で定義されたaddregistryセクションを参照します。

注 初期インストール後に INF ファイルを使用してドライバーをアップグレードする場合は、addregistryセクションに含まれているエントリで、0x00000002 (FLG_ADDREG_NOCLONBER) フラグを指定する必要があります。このフラグを指定すると、後続の \ファイル\がインストールされるときに HKLM CurrentControlSet Services のレジストリエントリが保持されます。次に例を示します。

```
[ExampleFileSystem.AddRegistry]
HKR,Parameters,ExampleParameter,0x00010003,1
```

DefaultUninstall セクション (省略可能)

Defaultuninstallセクションは省略可能ですが、ドライバーをアンインストールできる場合は推奨されます。このファイルには、ファイルとレジストリエントリを削除するためのDelfilesディレクティブとdelfilesディレクティブが含まれています。

次のコード例では、delfilesディレクティブによって、ドライバーの INF ファイルの MyLegacyFilter ファイルおよび MyLegacyFilter セクションに記載されているファイルが削除されます。

```
[DefaultUninstall]
DelFiles = MyLegacyFilter.DriverFiles, MyLegacyFilter.UserFiles
DelReg = MyLegacyFilter.DelRegistry
```

Delregディレクティブは、アンインストールするサービスのレジストリから削除する情報を含む、1つまたは複数の INF ライターで定義されたdelregセクションを参照します。

DefaultUninstall. Services セクション (省略可能)

Defaultuninstall. Servicesセクションは省略可能ですが、ドライバーをアンインストールできる場合は推奨されます。ファイルシステムフィルタードライバーのサービスを削除するためのDelserviceディレクティブが含まれています。

次のコード例では、delserviceディレクティブによって、オペレーティングシステムから MyLegacyFilter サービスが削除されます。

```
[DefaultUninstall.Services]
DelService = MyLegacyFilter,0x200
```

注 delserviceディレクティブでは、削除前にサービスを_停止するように、常に 0x200 (SPSVCINST stopservice) フラグを指定する必要があります。

Strings セクション (必須)

Stringsセクションでは、次の例に示すように、INF ファイルで使用される% strkey% トークンを定義します。

```
[Strings]
Msft = "Microsoft Corporation"
MyLegacyFilterServiceDesc = "MyLegacyFilterFilter Driver"
MyLegacyFilterServiceName = "MyLegacyFilter"
MyLegacyFilterRegistry = "system\currentcontrolset\services\MyLegacyFilter"
MyLegacyFilterMaxRecords = "MaxRecords"
MyLegacyFilterMaxNames = "MaxNames"
MyLegacyFilterDebugFlags = "DebugFlags"
Disk1 = "MyLegacyFilter Source Media"
```

ロケール固有の文字列を追加することで、単一の国際 INF ファイルを作成できます。INF ファイルのLanguageIDセクション。国際対応の INF ファイルの詳細については、「[国際対応の Inf ファイルの作成](#)」を参照してください。

ファイル システム フィルター ドライバー用の ロード順序グループ

2020/04/24 • [Edit Online](#)

Microsoft Windows XP およびそれ以降のオペレーティング システム専用のセットを提供ロード順序グループのシステムの起動時に読み込まれているファイル システム フィルター ドライバー。Windows XP より前に、のオペレーティング システムでフィルター ドライバーは、「フィルター」のみを使用でき、"file system"の読み込み順序グループ。

フィルターは、既存のファイル システム ドライバー スタックの先頭にのみアタッチでき、スタックの途中でアタッチできません。その結果、ロード順序グループはファイル システム フィルター ドライバー、重要な前フィルター ドライバーが読み込まれるため、小さいにアタッチできますファイル システム ドライバー スタック上。

ロード順序グループの詳細については、次の規則は、ファイル システム フィルター ドライバーが読み込まれるときに決定します。

- 特定のロード順序グループを指定するファイル システム フィルター ドライバーは、そのグループ内の他のフィルター ドライバーと同時に読み込まれます。
- 各ロード順序グループ内では、フィルター ドライバーはランダムな順序で読み込まれます。
- ファイル システム フィルター ドライバーがロード順序グループを指定しない場合は、ロード順序グループを指定して型をすべて同じの他のドライバーの開始後に読み込まれます。

次の表は、ファイル システム フィルター ドライバーのシステム定義のロード順序グループを一覧表示します。ロード順序グループごとに、ロード順序グループ列には、そのグループを指定する値が含まれています、LoadOrderGroup内のエントリ、[バージョン セクション](#)フィルターの INF ファイルです。

読み込まれた順の逆であると、スタックに表示されると、ロード順序グループが表示されることに注意してください。

LoadOrderGroup	Description
Filter	このグループでは、Windows 2000 で使用可能な以前 [フィルター] のロード順序グループと同じです。このグループは、最後に読み込まれ、したがって、ファイル システム遠いアタッチします。
FSFilter Upper	このグループは、何より FSFilter 他の種類をアタッチする必要がありますフィルター ドライバーに対して提供されます。
FSFilter Usage Monitor	このグループには、フィルター ドライバー確認し、ファイル I/O レポートにはが含まれています。
FSFilter Deletion Recovery	このグループには、削除されたファイルを回復するフィルター ドライバーが含まれています。
FSFilter Virus Protection	このグループには、検出して、ファイル I/O 中にウイルスを駆除するフィルターが含まれています。
FSFilter Replication	このグループには、リモート サーバーにファイル データをレプリケートするフィルター ドライバーが含まれています。

機能	説明
FSFilter 継続的なバックアップ	このグループには、バックアップ メディアにファイル データをレプリケートするフィルター ドライバーが含まれています。
FSFilter コンテンツごだわりの検索	このグループには、特定のファイルまたはファイルの内容の作成を禁止するフィルター ドライバーが含まれています。
FSFilter クォータの管理	このグループには、拡張ファイル システムのクォータを提供するフィルター ドライバーが含まれています。
FSFilter システムの回復	このグループには、システムの復元 (SR) フィルターなどのオペレーティング システムの整合性を維持するために操作を実行するフィルター ドライバーが含まれています。
FSFilter クラスタ ファイル システム	このグループには、ネットワーク経由でファイル サーバーのメタデータを提供する製品で使用するフィルター ドライバーが含まれます。
FSFilter HSM	このグループには、階層型記憶域の管理を実行するフィルター ドライバーが含まれています。
FSFilter イメージング	このグループには、仮想名前空間を提供する ZIP に似たフィルター ドライバーが含まれています。 このグループの読み込みは、Windows Vista およびそれ以降のバージョンのオペレーティング システムで利用できます。
FSFilter 圧縮	このグループには、ファイルのデータ圧縮を実行するフィルター ドライバーが含まれています。
FSFilter 暗号化	このグループには、暗号化、およびファイル I/O 中にデータを復号化するフィルター ドライバーが含まれています。
FSFilter 仮想化	このグループには、Windows Vista で追加された、少なくとも承認されたユーザー (LUA) フィルター ドライバーなどのファイルのパスを仮想化フィルター ドライバーが含まれています。 このグループの読み込みは、Windows Vista およびそれ以降のバージョンのオペレーティング システムで利用できます。
FSFilter 物理クォータの管理	このグループには、物理ブロックの数を使用してクォータを管理するフィルター ドライバーが含まれています。
開いている FSFilter ファイル	このグループには、既に関いているファイルのスナップショットを提供するフィルター ドライバーが含まれています。

機能	説明
FSFilter セキュリティ強化	このグループには、ロックダウンを適用するフィルター ドライバーが含まれていて、強化されたアクセス制御リスト (Acl)。
FSFilter コピー防止	このグループには、メディア上の帯域外のデータをチェックするフィルター ドライバーが含まれています。
FSFilter 下部	このグループは、他のすべての FSFilter 型の下にアタッチする必要がありますフィルター ドライバーに対して提供されます。
FSFilter システム	内部使用のために予約済みです。このグループには、HSM と SIS フィルター ドライバーが含まれています。
FSFilter インフラストラクチャ	内部使用のために予約済みです。このグループは、ファイル システムに最も近い接続および最初を読み込みます。

ファイル システム フィルター ドライバー クラスとクラス GUID

2020/04/24 • [Edit Online](#)

Microsoft Windows XP およびそれ以降のオペレーティング システムは、ファイル システム フィルター ドライバーのセットアップ クラスを提供します。これらのクラスは、ハードウェア デバイスのシステム提供のデバイス セットアップ クラスを提供する機能のサブセットを提供します。(ハードウェア デバイスのセットアップ クラスの詳細については、次を参照してください[デバイス セットアップ クラス](#)。)。

各セットアップ クラスは、クラスの GUID に関連付けられています。システム定義のクラス Guid は devguid.h で定義されます。

このトピックでは、ファイル システム フィルター ドライバーのセットアップ クラスを使用します。各クラスの定義で、クラスとClassGuidエントリで指定する必要があります値を含める、[INF パージョン セクション](#)のフィルターの INF ファイル。フィルター ドライバーには、クラスとドライバーの INF ファイルで指定されているロード順序グループに一致する GUID を使用する必要があります。

デバイスの場合、またはに加え、INF ファイル内の適切なクラス GUID 値を指定して、クラス = クラス名エントリのパフォーマンスを大幅に向上します。システム INF ファイルを検索します。

次の一覧には、システム定義のクラスおよびファイル システム フィルター ドライバーのクラスの Guid が含まれています。このリストのエントリは、Windows XP 以降のオペレーティング システムでファイル システム フィルター ドライバーに対して作成されたロード順序グループに対応します。

注 セットアップ クラスとは見なされませんいて、したがってクラスに割り当てられた Guid がないため、ボックスの一覧で、次の 3 つのロード順序グループは表示されません。フィルター、FSFilter、上下 FSFilter します。

FSFilter 利用状況モニター

クラス ActivityMonitor を =

ClassGuid = {b86dff51-a31e-4bac-b3cf-e8cfe75c9fc2}

FSFilter 削除の取り消し

クラス = 削除の取り消し

ClassGuid = {fe8f1572-c67a-48c0-bbac-0b5c6d66cafb}

FSFilter ウイルス対策

クラスのウイルス対策を =

ClassGuid = {b1d1a169-c54f-4379-81db-bee7d88d7454}

FSFilter レプリケーション

クラスのレプリケーションを =

ClassGuid = {48d3ebc4-4cf8-48ff-b869-9c68ad42eb9f}

FSFilter 継続的なバックアップ

クラス ContinuousBackup を =

ClassGuid = {71aa14f8-6fad-4622-ad77-92bb9d7e6947}

FSFilter コンテンツこだわりの検索

クラス ContentScreener を =

ClassGuid = {3e3f0674-c83c-4558-bb26-9820e1eba5c5}

FSFilter クォータの管理

クラス QuotaManagement を =

ClassGuid = {8503c911-a6c7-4919-8f79-5028f5866b0c}

FSFilter クラスター ファイル システム

クラス CFSMetaDataServer を =

ClassGuid = {cdcf0939-b75b-4630-bf76-80f7ba655884}

FSFilter HSM

クラスの HSM を =

ClassGuid = {d546500a-2aeb-45f6-9482-f4b1799c3177}

FSFilter 圧縮

クラスの圧縮を =

ClassGuid = {f3586baf-b5aa-49b5-8d6c-0569284c639f}

FSFilter 暗号化

クラスの暗号化を =

ClassGuid = {a0a701c0-a511-42ff-aa6c-06dc0395576f}

FSFilter 物理クォータの管理

クラス PhysicalQuotaManagement を =

ClassGuid = {6a0a8e78-bba6-4fc4-a709-1e33cd09d67e}

開いている FSFilter ファイル

クラス OpenFileBackup を =

ClassGuid = {f8ecafa6-66d1-41a5-899b-66585d7216b7}

FSFilter セキュリティ強化

クラス SecurityEnhancer を =

ClassGuid = {d02bc3da-0c8e-4945-9bd5-f1883c226c8c}

FSFilter コピー防止

クラス CopyProtection を =

ClassGuid = {89786ff1-9c12-402f-9c9e-17753c7f4375}

FSFilter システム

クラス FSFilterSystem を =

ClassGuid = {5d1b9aaa-01e2-46af-849f-272b3f324c46}

FSFilter インフラストラクチャ

クラスのインフラストラクチャを =

ClassGuid = {e55fa6f9-128c-4d04-abab-630c74b1453a}

INF ファイルを使用したファイル システム フィルター ドライバーのインストール

2020/04/24 • [Edit Online](#)

INF ファイルを作成した後は、インストール、アップグレード、およびファイル システム フィルター ドライバーをアンインストールに使用できます。単独またはバッチ ファイルまたはと共にユーザー モードのセットアップ アプリケーション INF ファイルを使用することができます。

インストールを右クリックします。

実行する、[DefaultInstall](#) と [DefaultInstall.Services](#) のセクションでは、INF ファイルの次を行う必要があります。

1. Windows エクスプローラーでは、INF ファイル名を右クリックします。ショートカット メニューが表示されます。
2. [インストール] をクリックします。

注 INF ファイルが含まれている場合にのみ、ショートカット メニューが表示されます、[DefaultInstall](#)セクション。

コマンドラインまたはバッチ ファイルのインストール

実行する、[DefaultInstall](#)と[DefaultInstall.Services](#)コマンドラインまたはバッチ ファイルのインストールを使用して、INF ファイルのセクションは、コマンド プロンプトで次のコマンドを入力または作成し、このコマンドが含まれたバッチ ファイルを実行します。

```
RUNDLL32.EXE SETUPAPI.DLL,InstallHinfSection DefaultInstall 132 path-to-inf\infname.inf
```

"Rundll32"と"InstallHinfSection"が、ツール、セットアップとシステム管理で説明されているセクションをそれぞれの Microsoft Windows SDK のドキュメント。

アプリケーションをセットアップします。

[InstallHinfSection](#) 次のコード例に示すように、セットアップ アプリケーションから呼び出すもできます。

```
InstallHinfSection(NULL,NULL,TEXT("DefaultInstall 132 path-to-inf\infname.inf"),0);
```

ドライバーをインストールするセットアップ アプリケーションを使用する場合は、次のガイドラインに従います。

- 最終的なアンインストールの準備として、セットアップ アプリケーションは、ディレクトリのアンインストールにドライバーの INF ファイルをコピーする必要があります。
- セットアップ アプリケーションでは、ドライバーを使用した、ユーザー モード アプリケーションをインストールする場合、ユーザーでアンインストールに必要な場合はようにこのアプリケーションは追加またはコントロール パネルの [プログラムの削除] に表示する必要があります。アプリケーションとドライバーの両方を表す 1 つの項目を表示する必要があります。

追加と削除 で、アプリケーションの一覧を表示する方法の詳細については、Windows sdk のセットアップとシステム管理のセクションでは"を削除する an Application"を参照してください。

- アプリケーションをセットアップする必要があります Windows INF ファイルのディレクトリにドライバーの INF ファイルをコピーしない (%windir%\INF)。SetupAPI がファイルを自動的にコピーの一部として、[InstallHinfSection](#) 呼び出します。

セットアップ アプリケーションの詳細については、次を参照してください。[デバイス インストール アプリケーションを記述して](#)します。

INF ファイルを使用したファイル システム フィルター ドライバーのアンインストール

2020/04/24 • [Edit Online](#)

ドライバーをアンインストールするには、コマンドライン、PowerShell、INF ファイルをバッチファイルと共に使用するか、ユーザーモードのアンインストールアプリケーションを使用します。

"右クリックアンインストール" オプションはありません。

コマンドラインまたはバッチファイルのアンインストール

INF ファイルのdefaultuninstallおよびdefaultuninstall. Servicesセクションをコマンドラインで実行するには、コマンドプロンプトで次のコマンドを入力するか、このコマンドを含むバッチファイルを作成して実行します。

```
RUNDLL32.EXE SETUPAPI.DLL,InstallHinfSection DefaultUninstall 132 path-to-uninstall-dir\infname.inf
```

Rundll32とInstallhinfsectionの詳細については、Microsoft Windows SDK のドキュメントの「ツールとセットアップ」と「システム管理」セクションを参照してください。

Powershell のアンインストール

Powershell コマンドプロンプトで、次のコマンドを入力します。

```
Get-CimInstance Win32_SystemDriver -Filter "name='your_driver_name'" | Invoke-CimMethod -MethodName Delete
```

アプリケーションのアンインストール

次のコード例に示すように、アンインストールアプリケーションから INF ファイルのdefaultuninstallおよびdefaultuninstall. Servicesセクションを実行することもできます。

```
InstallHinfSection(NULL,NULL,TEXT("DefaultUninstall 132 path-to-uninstall-dir\infname.inf"),0);
```

アプリケーションを使用してドライバーをアンインストールする場合は、次のガイドラインに従ってください。

- 最終的なアンインストールを準備するには、セットアップアプリケーションで、ドライバーの INF ファイルをアンインストールディレクトリにコピーする必要があります。
- INF ファイルのdefaultuninstall. Servicesセクションでは、delserviceディレクティブで、削除前にサービスを停止する_ように、常に 0x200 (SPSVCINST stopservice) フラグを指定する必要があります。
- ユーザーモードアプリケーションがドライバーと共にインストールされている場合は、ユーザーが必要に応じてアンインストールできるように、コントロールパネルの [プログラムの追加と削除] にこのアプリケーションが表示されます。アプリケーションとドライバーの両方を表す項目が1つだけ表示されます。[プログラムの追加と削除] でアプリケーションを一覧表示する方法の詳細については、Microsoft Windows SDK ドキュメントの「セットアップとシステム管理」セクションの「アプリケーションの削除」を参照してください。
- アンインストールアプリケーションでは、Windows inf ファイルディレクトリ (%windir%\INF) から inf ファイル (またはそれに関連付けられている PNF ファイル) を削除しないでください。
- 一部のフィルタードライバーファイルは、アプリケーションのアンインストール時に安全に削除できません。これらのファイルは、INF ファイルのDefaultuninstall. Servicesセクションには記載されていません。

アプリケーションのアンインストールの詳細については、「[デバイスインストールアプリケーションの作成](#)」を参照してください。

ファイル システム フィルター ドライバーの初期化

2020/04/24 • [Edit Online](#)

ファイルシステムフィルタードライバーを初期化する**Driverentry**ルーチンは、デバイスドライバーを初期化するための**driverentry**ルーチンとよく似ています。ドライバーが読み込まれた後、ドライバーを読み込んだ同じコンポーネントも、ドライバーの**Driverentry**ルーチンを呼び出すことによってドライバーを初期化します。ファイルシステムフィルタードライバーの場合、ドライバーを読み込むコンポーネントは、i/o マネージャー (start type が SERVICE_BOOT_START) またはサービスコントロールマネージャー (他の開始の種類の場合) のいずれかです。

Driverentryルーチンは、IRQL = パッシブ レベルでシステムスレッドコンテキストで実行されます。このルーチンは、ページングが可能であり、破棄されるように INIT セグメントに存在する必要があります。ドライバーコードをページング可能にする方法の詳細については、**MmLockPagableCodeSection**の「解説」を参照してください。

Driverentryルーチンは次のように定義されています。

```
NTSTATUS
(*PDRIVER_INITIALIZE) (
    IN PDRIVER_OBJECT DriverObject,
    IN PUNICODE_STRING RegistryPath
);
```

このルーチンには2つの入力パラメーターがあります。最初の *DriverObject* は、ファイルシステムフィルタードライバーが読み込まれたときに作成されたドライバーオブジェクトです。2番目の *RegistryPath* は、ドライバーのレジストリキーへのパスを含む、カウントされた Unicode 文字列へのポインターです。

ファイルシステムフィルタードライバーの**Driverentry**ルーチンは、次の手順を実行します。

コントロールデバイスオブジェクトの作成

IRP ディスパッチルーチンを登録しています

高速 i/o ディスパッチルーチンを登録しています

FsFilter コールバックルーチンを登録しています

その他の必要な初期化を実行しています

[オプション] コールバックルーチンの登録

[オプション] レジストリパス文字列のコピーを保存します。

ステータスを返す

コントロール デバイス オブジェクトの作成

2020/04/24 • [Edit Online](#)

通常、ファイルシステムフィルタードライバーの**Driverentry**ルーチンは、コントロールデバイスオブジェクトを作成することから始まります。コントロールデバイスオブジェクトの目的は、フィルターがファイルシステムまたはボリュームデバイスオブジェクトにアタッチされる前であっても、アプリケーションがフィルタードライバーと直接通信できるようにすることです。

ファイルシステムでは、コントロールデバイスオブジェクトも作成されることに注意してください。ファイルシステムフィルタードライバーは、個々のファイルシステムボリュームではなく、ファイルシステムにそれ自体をアタッチするときに、ファイルシステムのコントロールデバイスオブジェクトにそれ自体をアタッチします。

次の例では、コントロールデバイスオブジェクトを作成します。

```
RtlInitUnicodeString(&nameString, MYLEGACYFILTER_FULLDEVICE_NAME);
status = IoCreateDevice(
    DriverObject,           //DriverObject
    0,                     //DeviceExtensionSize
    &nameString,            //DeviceName
    FILE_DEVICE_DISK_FILE_SYSTEM, //DeviceType
    FILE_DEVICE_SECURE_OPEN, //DeviceCharacteristics
    FALSE,                 //Exclusive
    &gControlDeviceObject); //DeviceObject

RtlInitUnicodeString(&linkString, MYLEGACYFILTER_DOSDEVICE_NAME);
status = IoCreateSymbolicLink(&linkString, &nameString);
```

ファイルシステムとは異なり、ファイルシステムフィルタードライバーは、コントロールデバイスオブジェクトに名前を指定する必要はありません。**IoRegisterDeviceInterface**への呼び出しはコントロールデバイスオブジェクトに対して無効であるため、ユーザーモードアプリケーションは、デバイス名を使用してフィルタードライバーにアクセスすることはできません。*DeviceName*パラメーターにNULL以外の値が渡された場合、この値はコントロールデバイスオブジェクトの名前になります。**Driverentry**ルーチンは、前のコード例に示されているように、**IoStream Ymの iclink**ルーチン呼び出しで、オブジェクトのカーネルモード名をアプリケーションから参照できるユーザーモード名にリンクさせることができます。

注 コントロールデバイスオブジェクトは、ドライバースタックにアタッチされていない唯一のデバイスオブジェクトであるため、安全に名前を付けることができるデバイスオブジェクトの唯一の種類です。このため、ファイルシステムフィルタードライバーのデバイスオブジェクトの制御には、必要に応じて名前を付けることができます。ファイルシステムのコントロールデバイスオブジェクトには名前を付ける必要があることに注意してください。フィルターデバイスオブジェクトに名前を付けることはできません。

(*Devicetype*パラメーターに割り当てられる値は、ntifs で定義されているデバイスの種類の1つである必要があります。たとえば、*ファイル_デバイス_ディスク_ファイル_システム*などです。

*DeviceName*パラメーターにNULL以外の値が渡された場合、*DEVICECHARACTERISTICS*フラグには、*ファイル_デバイス_セキュリティで保護された_を開く*必要があります。このフラグは、コントロールデバイスオブジェクトに送信されるすべての開いている要求に対してセキュリティチェックを実行するように i/o マネージャーに指示します。これらのセキュリティチェックは、名前付きデバイスオブジェクトの ACL に対して行われます。

ファイルシステムフィルタードライバーがディスパッチルーチンで独自のコントロールデバイスオブジェクトを識別する効果的な方法は、デバイスポインターと、以前に格納されていたグローバルポインターをコントロールデバイスオブジェクトに比較することです。したがって、前のサンプルでは、**IoCreateDevice**によって返された*DeviceObject*ポインターを、グローバルに定義されたポインター変数 `gControlDeviceObject` に格納しています。

IRP ディスパッチ ルーチンの登録

2020/04/24 • [Edit Online](#)

フィルタードライバーの **DriverObject** ルーチンの *driverobject* パラメーターは、フィルタードライバーの **ドライバーオブジェクト** へのポインターを提供します。I/O 要求パケット (IRP) ディスパッチルーチンを登録するには、これらのルーチンのエントリポイントを driver オブジェクトの **MajorFunction** メンバーに格納する必要があります。たとえば、"MyLegacyFilter" という架空のドライバーでは、ディスパッチルーチンのエントリポイントを次のように設定できます。

```
for (i = 0; i <= IRP_MJ_MAXIMUM_FUNCTION; i++) {
    DriverObject->MajorFunction[i] = MyLegacyFilterDispatch;
}
DriverObject->MajorFunction[IRP_MJ_CREATE] = MyLegacyFilterCreate;
DriverObject->MajorFunction[IRP_MJ_CLOSE] = MyLegacyFilterClose;
DriverObject->MajorFunction[IRP_MJ_FILE_SYSTEM_CONTROL] = MyLegacyFilterFsControl;
```

上記の **for** ループでは、すべての IRP 主要な関数コードに既定のディスパッチルーチンが割り当てられていることに注意してください。この割り当ては適切な方法です。これを行わないと、i/o マネージャーは不明な状態の IRP を完了し、既定ではデバイス_要求_無効_ます。このような要求は通常、ドライバースタックの下位にある別のドライバーを対象としているため、ファイルシステムフィルタードライバーは未知の Irp をこのように拒否することはできません。このため、既定のディスパッチルーチンは、通常、次の下位レベルのドライバーに IRP を渡します。

高速 I/O ディスパッチ ルーチンの登録

2020/04/24 • [Edit Online](#)

フィルタードライバーの **DriverObject** ルーチンの *driverobject* パラメーターは、フィルタードライバーの **ドライバーオブジェクト** へのポインターを提供します。

ファイルシステムフィルタードライバーの高速 i/o ディスパッチルーチンを登録するには、高速 i/o ディスパッチテーブルを割り当てて初期化し、高速 i/o ディスパッチルーチンのエントリポイントをテーブルに格納して、テーブルのアドレスを `Fasystem.servicemodel Odispatch` に格納する必要があります。driver オブジェクトのメンバー。

たとえば、仮想的な "MyLegacyFilter" ドライバーでは、次のように、高速 i/o ディスパッチルーチンのエントリポイントを設定できます。

```
RtlZeroMemory(fastIoDispatch, sizeof(FAST_IO_DISPATCH));
fastIoDispatch->SizeOfFastIoDispatch = sizeof(FAST_IO_DISPATCH);
fastIoDispatch->FastIoCheckIfPossible = MyLegacyFilterIoCheckIfPossible;
fastIoDispatch->FastIoRead = MyLegacyFilterIoRead;
fastIoDispatch->FastIoWrite = MyLegacyFilterIoWrite;
fastIoDispatch->FastIoQueryBasicInfo = MyLegacyFilterIoQueryBasicInfo;
fastIoDispatch->FastIoQueryStandardInfo = MyLegacyFilterIoQueryStandardInfo;
fastIoDispatch->FastIoLock = MyLegacyFilterIoLock;
fastIoDispatch->FastIoUnlockSingle = MyLegacyFilterIoUnlockSingle;
fastIoDispatch->FastIoUnlockAll = MyLegacyFilterIoUnlockAll;
fastIoDispatch->FastIoUnlockAllByKey = MyLegacyFilterIoUnlockAllByKey;
fastIoDispatch->FastIoDeviceControl = MyLegacyFilterIoDeviceControl;
fastIoDispatch->FastIoDetachDevice = MyLegacyFilterIoDetachDevice;
fastIoDispatch->FastIoQueryNetworkOpenInfo = MyLegacyFilterIoQueryNetworkOpenInfo;
fastIoDispatch->MdlRead = MyLegacyFilterIoMdlRead;
fastIoDispatch->MdlReadComplete = MyLegacyFilterIoMdlReadComplete;
fastIoDispatch->PrepareMdlWrite = MyLegacyFilterIoPrepareMdlWrite;
fastIoDispatch->MdlWriteComplete = MyLegacyFilterIoMdlWriteComplete;
fastIoDispatch->FastIoReadCompressed = MyLegacyFilterIoReadCompressed;
fastIoDispatch->FastIoWriteCompressed = MyLegacyFilterIoWriteCompressed;
fastIoDispatch->MdlReadCompleteCompressed = MyLegacyFilterIoMdlReadCompleteCompressed;
fastIoDispatch->MdlWriteCompleteCompressed = MyLegacyFilterIoMdlWriteCompleteCompressed;
fastIoDispatch->FastIoQueryOpen = MyLegacyFilterIoQueryOpen;

DriverObject->FastIoDispatch = fastIoDispatch;
```

FsFilter コールバック ルーチンの登録

2020/04/24 • [Edit Online](#)

FsFilter 通知コールバックルーチンは、基になるファイルシステムが特定の操作を実行する前と後に呼び出されます。FsFilter のコールバックルーチンの詳細については、「[Fsrtlregisterfilesystemfiltercallbacks](#)」を参照してください。

FsFilter 通知コールバックルーチンを登録するには、FS_フィルター_コールバック構造体を割り当てて初期化し、FsFilter コールバックルーチンのエントリポイントを構造体に格納して、構造体のアドレスをFsrtlregisterfilesystemfiltercallbacks バックへのコールバックパラメーター。

たとえば、架空の "MyLegacyFilter" ドライバーは、次のように、その FsFilter コールバックルーチンを登録できます。

```
fsFilterCallbacks.SizeOfFsFilterCallbacks = sizeof(FS_FILTER_CALLBACKS);
fsFilterCallbacks.PreAcquireForSectionSynchronization = MyLegacyFilterPreFsFilterOperation;
fsFilterCallbacks.PostAcquireForSectionSynchronization = MyLegacyFilterPostFsFilterOperation;
fsFilterCallbacks.PreReleaseForSectionSynchronization = MyLegacyFilterPreFsFilterOperation;
fsFilterCallbacks.PostReleaseForSectionSynchronization = MyLegacyFilterPostFsFilterOperation;
fsFilterCallbacks.PreAcquireForCcFlush = MyLegacyFilterPreFsFilterOperation;
fsFilterCallbacks.PostAcquireForCcFlush = MyLegacyFilterPostFsFilterOperation;
fsFilterCallbacks.PreReleaseForCcFlush = MyLegacyFilterPreFsFilterOperation;
fsFilterCallbacks.PostReleaseForCcFlush = MyLegacyFilterPostFsFilterOperation;
fsFilterCallbacks.PreAcquireForModifiedPageWriter = MyLegacyFilterPreFsFilterOperation;
fsFilterCallbacks.PostAcquireForModifiedPageWriter = MyLegacyFilterPostFsFilterOperation;
fsFilterCallbacks.PreReleaseForModifiedPageWriter = MyLegacyFilterPreFsFilterOperation;
fsFilterCallbacks.PostReleaseForModifiedPageWriter = MyLegacyFilterPostFsFilterOperation;

status = FsRtlRegisterFileSystemFilterCallbacks(DriverObject, &fsFilterCallbacks);
```

その他必要な初期化の実行

2020/04/24 • [Edit Online](#)

IRPと高速のI/Oの登録後にディスパッチルーチンの場合、ファイルシステムフィルタードライバーのDriverEntryルーチンが必要に応じて、グローバルの追加のドライバーの変数とデータの構造体を初期化できます。

[オプション] コールバックルーチンの登録

2020/04/24 • [Edit Online](#)

フィルタードライバーは、[IoRegisterFsRegistrationChange](#)を呼び出して、ファイルシステムドライバーが[IoRegisterFileSystem](#)または[iounregisterfilesystem](#)を呼び出して自身を登録または登録解除するたびに呼び出されるコールバックルーチンを登録できます。フィルタードライバーこのコールバックルーチンを登録すると、新しいファイルシステムにシステムを入力し、アタッチするかどうかを選択できます。

注 ファイルシステムフィルタードライバーは、[IoRegisterFileSystem](#)または[iounregisterfilesystem](#)を呼び出すことはできません。これらのルーチンは、ファイルシステム専用です。

明示的に指示された場合 (ユーザーモードアプリケーションなど) にのみボリュームにアタッチするフィルタードライバーは、[IoRegisterFsRegistrationChange](#)を呼び出すことはできません。ただし、このルーチンを使用するフィルターは、そのボリュームがマウントされた直後に任意のボリュームにアタッチすることができます。このルーチンを使用しても、フィルターが直接ボリュームデバイスオブジェクトにアタッチされるとは限りません。ただし、このようなフィルターは、現在のファイルシステムボリュームのデバイススタックの最上位にしか接続できないため、ユーザーモードアプリケーションからのコマンドを待機するフィルターの前 (およびその下) にアタッチされることを保証します。

[オプション] レジストリパス文字列のコピーを保存します。

2020/04/24 • [Edit Online](#)

注 この手順が必要なのは、**driverentry**ルーチンから制御が戻った後にフィルタードライバーがレジストリパスを使用する必要がある場合のみです。

入力として**Driverentry**に渡された**RegistryPath**文字列のコピーを保存します。このパラメーターは、ドライバーのレジストリキーへのパスを指定する、カウントされた Unicode 文字列を指しています。

\registry\Machine\System\CurrentControlSet\Servicesドライバー名です。ここで、ドライバー名はドライバーの名前です。後で**RegistryPath**文字列が必要になった場合、**driverentry**は **driverentry** ルーチンが返された後にポインターが無効になるため、**driverentry**はポインターだけでなく、そのコピーを保存する必要があります。

RtlCopyUnicodeStringルーチンを使用して、**RegistryPath**ソース文字列をコピー先の文字列にコピーできます。

返される状態

2020/04/24 • [Edit Online](#)

ファイル システム フィルター ドライバーのDriverEntryルーチンが通常の状態を返します_成功します。ただし、ドライバーの初期化に失敗した場合、DriverEntryルーチンは、該当するエラー状態の値を返す必要があります。

場合、DriverEntryルーチンが成功状態の値ではない状態の値を返す、システムのドライバーをアンロードして応答します。このため、DriverEntry常に、ルーチンは成功の状態の値ではない状態の値を返す前に、デバイス オブジェクトなどのシステムリソースの割り当てられたメモリを解放する可能性があります。

ファイル システムまたはボリュームへのフィルターのアタッチ

2020/04/24 • [Edit Online](#)

ファイル システム フィルター ドライバーは自体を 1 つまたは複数のマウントされたボリュームにアタッチし、それらのすべての I/O 操作をフィルターします。しかし、それ自体をアタッチするには、どのボリュームを決定方法はどうでしょうか。サンプルのフィルター ドライバー、Windows Driver Kit (WDK) では、実行は、この 2 つの最も一般的な方法を示しています。

- エンド ユーザーでは、フィルター処理にボリュームをたとえば、指定するため、ボリュームのドライブ文字を入力します。エンド ユーザーのコマンドは、プライベートとしてフィルター ドライバーに中継 [IRP_MJ_デバイス_コントロール](#) 要求。
- ファイル システム フィルター ドライバーが 1 つまたは複数のファイル システム ドライバーをアタッチ、リッスン [IRP_MJ_ファイル_システム_コントロール](#)、IRP_MN_マウント_ボリュームの要求とは、マウントされたボリュームに接続します。

注 ドライブ文字をボリュームのマッピングは、1 対多、一対一しないことを想定してください一般にします。これは、ダイナミック ボリュームとボリュームのマウント ポイントなど、高度なストレージ機能のためです。

注 想定しないでくださいその IRP_MN_マウント_ボリューム要求がファイル システムで同期的に処理が常にします。たとえば、フロッピー ドライブは、ドライブにフロッピー ディスクがない場合、非同期的にマウントすることがあります。したがって、フィルター ドライバーを準備を伝達する必要があります、PendingReturnedそのマウント完了ルーチンのフラグ。詳細については、次を参照してください"[PendingReturned フラグ](#)"。

ファイル システム フィルター ドライバーにアタッチし、任意のファイル システム ボリュームの I/O をフィルター処理できます。これらは、ディスク ドライブやパーティションなどの記憶装置に直接アタッチできません。また、個々のディレクトリまたはファイルに、アタッチできません。

詳しくは、次のトピックをご覧ください。

[フィルターのデバイス オブジェクトを作成します。](#)

[ターゲット デバイスのオブジェクトへのフィルターのデバイス オブジェクトのアタッチ](#)

[メッセージを伝達する_バッファーに格納された_IO、および DO_直接_IO フラグ](#)

[ファイルを伝達する_デバイス_SECURE_オープン フラグ](#)

[オフ_デバイス_初期化フラグ](#)

フィルター デバイス オブジェクトの作成

2020/04/24 • [Edit Online](#)

IoCreateDeviceを呼び出して、ボリュームまたはファイルシステムスタックにアタッチするフィルターデバイスオブジェクトを作成します。次に例を示します。

```
status = IoCreateDevice(  
    gFileSpyDriverObject,           //DriverObject  
    sizeof(MYLEGACYFILTER_DEVICE_EXTENSION), //DeviceExtensionSize  
    NULL,                           //DeviceName  
    DeviceObject->DeviceType,       //DeviceType  
    0,                              //DeviceCharacteristics  
    FALSE,                          //Exclusive  
    &newDeviceObject);              //DeviceObject
```

上記のコードスニペットでは、*DeviceObject*は、フィルターデバイスオブジェクトがアタッチされるターゲットデバイスオブジェクトへのポインターです。*newDeviceObject*は、フィルターデバイスオブジェクト自体へのポインターです。

*Deviceextensionsize*パラメーターを`sizeof(MYLEGACYFILTER_DEVICE_EXTENSION)`に設定すると、フィルターデバイスオブジェクトに対して `MYLEGACYFILTER_DEVICE_EXTENSION` 拡張構造が割り当てられます。新しく作成されたフィルターデバイスオブジェクトの *Deviceextension* メンバーは、この構造体を指すように設定されます。通常、ファイルシステムフィルタードライバは、各フィルターデバイスオブジェクトのデバイス拡張機能のメモリを定義し、割り当てます。デバイス拡張機能の構造と内容は、ドライバによって異なります。ただし、Microsoft Windows XP 以降では、フィルタードライバによって、少なくとも次のメンバーを含むフィルタードライバオブジェクトのデバイス_拡張構造が定義されている必要があります。

```
PDEVICE_OBJECT AttachedToDeviceObject;
```

上記の**IoCreateDevice**の呼び出しで、*DeviceName*パラメーターをNULLに設定すると、フィルターデバイスオブジェクトの名前は指定されません。フィルターデバイスオブジェクトには名前が付けられません。フィルターデバイスオブジェクトはファイルシステムまたはボリュームドライバースタックにアタッチされるので、フィルターデバイスオブジェクトに名前を割り当てると、システムセキュリティホールが作成されます。

(*Devicetype*パラメーターは、フィルターデバイスオブジェクトをアタッチするターゲット (ファイルシステムまたはフィルター) デバイスオブジェクトと同じデバイスの種類に常に設定する必要があります。この方法でデバイスの種類を伝達することが重要です。この方法は、i/o マネージャーによって使用され、アプリケーションにレポートすることができます。

注 ファイルシステムとファイルシステムフィルタードライバでは、(*devicetype*パラメーターを `FILE_DEVICE_file_system` に設定しないでください。これは、このパラメーターの有効な値ではありません。(ファイル_デバイス_ファイル_システム定数は、FSCTL コードを定義する目的でのみ使用されます)。

(*Devicetype*パラメーターが重要であるもう1つの理由は、特定の種類のファイルシステムにのみ、多くのフィルターがアタッチされることです。たとえば、特定のフィルターはすべてのローカルディスクファイルシステムにアタッチできますが、CD-ROM ファイルシステムやリモートファイルシステムにはアタッチできません。このようなフィルターは、ファイルシステムまたはボリュームドライバースタック内の最上位のデバイスオブジェクトのデバイスの種類を調べることによって、ファイルシステムの種類を決定します。ほとんどの場合、スタック内の最上位のデバイスオブジェクトはフィルターデバイスオブジェクトです。したがって、添付されているすべてのフィルターデバイスオブジェクトが、基になるファイルシステムまたはボリュームデバイスオブジェクトと同じ種類のデバイスであることが不可欠です。

フィルター デバイス オブジェクトのターゲット デバイス オブジェクトへのアタッチ

2020/04/24 • [Edit Online](#)

[IoAttachDeviceToDeviceStackSafe](#)を呼び出して、フィルターデバイスオブジェクトをターゲットファイルシステムまたはボリュームのフィルタードライバースタックにアタッチします。

```
devExt = myLegacyFilterDeviceObject->DeviceExtension;

status = IoAttachDeviceToDeviceStackSafe(
    myLegacyFilterDeviceObject,      //SourceDevice
    DeviceObject,                    //TargetDevice
    &devExt->AttachedToDeviceObject); //AttachedToDeviceObject
```

ほかによってポイントされているデバイスオブジェクトの上に他のフィルターが既にチェーンされている場合は、*AttachedToDeviceObject* output パラメーターによって受け取るデバイスオブジェクトのポインターがほかと異なる場合があることに注意してください。

名前によるファイルシステムへのアタッチ

すべてのファイルシステムで、1つまたは複数の名前付きコントロールデバイスオブジェクトを作成する必要があります。特定のファイルシステムに直接アタッチする場合、ファイルシステムフィルタードライバは、適切なファイルシステムコントロールデバイスオブジェクトの名前を[IoGetDeviceObjectPointer](#)に渡して、デバイスオブジェクトポインターを取得します。次のコードスニペットは、RAW ファイルシステムの2つのコントロールデバイスオブジェクトのいずれかへのポインターを取得する方法を示しています。

```
RtlInitUnicodeString(&nameString, L"\\Device\\RawDisk");

status = IoGetDeviceObjectPointer(
    &nameString,          //ObjectName
    FILE_READ_ATTRIBUTES, //DesiredAccess
    &fileObject,           //FileObject
    &rawDeviceObject);     //DeviceObject

if (NT_SUCCESS(status)) {
    ObDereferenceObject(fileObject);
}
```

[IoGetDeviceObjectPointer](#)への呼び出しが成功した場合、ファイルシステムフィルタードライバは、[IoAttachDeviceToDeviceStackSafe](#)を呼び出して、返されたコントロールデバイスオブジェクトにアタッチできます。

注 コントロールデバイスオブジェクトポインター (*rawDeviceObject*)に加えて、[IoGetDeviceObjectPointer](#)は、ユーザーモードでデバイスオブジェクトを表すファイルオブジェクト (*fileObject*) へのポインターを返します。上記のコードスニペットでは、ファイルオブジェクトは必要ないため、[ObDereferenceObject](#)を呼び出すことによって閉じられます。[IoGetDeviceObjectPointer](#)によって返されるファイルオブジェクトの参照カウントをデクリメントすると、デバイスオブジェクトの参照カウントもデクリメントされることに注意する必要があります。したがって、[ObDereferenceObject](#)が呼び出される前に、[ObDereferenceObject](#)を呼び出した後に、*fileObject*と*rawDeviceObject*のポインターは両方とも無効と見なされる必要があります。ただし、デバイスオブジェクトの参照カウントが、ファイルオブジェクトに対してが呼び出される前に、[ObReferenceObject](#)に対する追加の呼び出しによってインクリメントされる場合は除きます。

メッセージを伝達する_バッファーに格納された _IO、および DO_直接_IO フラグ

2020/04/24 • [Edit Online](#)

ファイル システムまたはボリュームをフィルター デバイス オブジェクトをアタッチした後は必ずを設定するかオフにします_バッファーに格納された_IO、および DO_直接_IO フラグの下のデバイスの値と一致するように、必要に応じてドライバ スタック上のオブジェクト。(これらのフラグの詳細については、次を参照してください[メソッドにアクセスするデータ バッファの](#)。)。この例は次に示します。

```
if (FlagOn( DeviceObject->Flags, DO_BUFFERED_IO )) {
    SetFlag( myLegacyFilterDeviceObject->Flags, DO_BUFFERED_IO );
}
if (FlagOn( DeviceObject->Flags, DO_DIRECT_IO )) {
    SetFlag(myLegacyFilterDeviceObject->Flags, DO_DIRECT_IO );
}
```

上のコード スニペットでデバイス オブジェクトにデバイス オブジェクトへのポインターは、フィルター デバイス オブジェクトがアタッチされているだけです myLegacyFilter デバイス オブジェクトフィルター デバイス オブジェクトへのポインターです。自体。

ファイル_デバイス_セキュリティで保護された_開くフラグを伝達する

2020/04/24 • [Edit Online](#)

フィルターデバイスオブジェクトをファイルシステムにアタッチした後 (ボリュームではない)、常に、ファイル_デバイスに対して、必要に応じてフィルターデバイスオブジェクトの [セキュリティで保護された_開く] フラグが設定されていることを確認してください。ドライバースタック。(このフラグの詳細については、「カーネルアーキテクチャ設計ガイド」と「デバイス_オブジェクト」の「デバイスの特性の指定」を参照してください)。この例を次に示します。

```
if (FlagOn( DeviceObject->Characteristics, FILE_DEVICE_SECURE_OPEN )) {  
    SetFlag(myLegacyFilterDeviceObject->Characteristics, FILE_DEVICE_SECURE_OPEN );  
}
```

上記のコードスニペットでは、*DeviceObject*は、フィルターデバイスオブジェクトがアタッチされた直前のデバイスオブジェクトへのポインターです。*myLegacyFilter DeviceObject*は、フィルターデバイスオブジェクト自体へのポインターです。

DO_デバイス_初期化フラグをクリアしています

2020/04/24 • [Edit Online](#)

フィルターデバイスオブジェクトをファイルシステムまたはボリュームにアタッチした後は、常に、フィルターデバイスオブジェクトの [初期化] フラグをオフにして、[デバイスの__を初期化する] を選択します。(このフラグの詳細については、「カーネルリファレンス」の「[DEVICE_オブジェクト](#)」を参照してください)。これは、*ntifs*で定義されているclearflag マクロを使用して、次のように実行できます。

```
ClearFlag(NewDeviceObject->Flags, DO_DEVICE_INITIALIZING);
```

フィルターデバイスオブジェクトが作成されると、[IoCreateDevice](#)はデバイスオブジェクトの初期化フラグ_DO_デバイスを設定します。フィルターが正常にアタッチされたら、このフラグをクリアする必要があります。このフラグがオフになっていない場合は、[Ioattachdevicetodevicestacksafe](#)の呼び出しが失敗するため、これ以上フィルタードライバーをフィルターチェーンにアタッチすることはできません。

注意 driverentry で作成されたデバイスオブジェクトに対して、DO_デバイス_初期化フラグをクリアする必要はありません。これは、i/o マネージャーによって自動的に行われるためです。ただし、ドライバーは、作成する他のすべてのデバイスオブジェクトでこのフラグをクリアする必要があります。

IRP と高速 I/O のフィルタリング

2020/04/24 • [Edit Online](#)

NOTE

最適な信頼性とパフォーマンスを得るには、従来のファイルシステムフィルタードライバーではなく、フィルタマネージャーをサポートする[ファイルシステムミニフィルタードライバー](#)を使用します。レガシドライバーをミニフィルタードライバーに移植する方法については、「[レガシフィルタードライバーを移植するためのガイドライン](#)」を参照してください。

ファイルシステムフィルタードライバーは、1つまたは複数のファイルシステムまたはファイルシステムボリュームの i/o 要求をフィルタ処理します。各 i/o 要求は、i/o 要求パケット (IRP) または高速 i/o 要求として表示されます。Irp は、ドライバの IRP ディスパッチルーチンによって処理される i/o システム構造です。高速 i/o 要求は、ドライバの高速 i/o コールバックルーチンによって処理されます。

フィルタードライバーが初期化されると、その `Driverentry` ルーチンによって、フィルタードライバーの IRP ディスパッチルーチンと高速 i/o コールバックルーチンが登録されます。これらのルーチンのセットは、フィルタードライバーごとに1つだけ登録できます。

一部の種類の Irp は、高速 i/o に相当します。また、一部の高速 i/o 要求には、対応する IRP があります。しかし、Irp は、高速 i/o ではできないさまざまな種類の i/o を処理します。また、特定の特化した高速 i/o ルーチンを使用して、IRP を作成せずに、キャッシュマネージャーまたはメモリマネージャーのファイルシステムリソースを事前に取得します。そのため、ほとんどの場合、Irp と高速 i/o 要求は i/o 操作で個別のロールを実行します。

このセクションでは、次のトピックについて説明します。

[Irp は、高速 i/o とは異なります。](#)

[ファイルシステムフィルタードライバーデバイスオブジェクトの種類](#)

IRP と高速 I/O の違い

2020/04/24 • [Edit Online](#)

Irp は、I/O 操作を要求するための既定のメカニズムです。同期または非同期の I/O、およびキャッシュまたはキャッシュされていない I/O、Irp を使用できます。Irp が I/O のページングも使用されます。ファイル システムに適切な Irp を送信することによってページフォールトを Memory Manager プロセス。

高速の I/O は、具体的には、キャッシュされたファイルでの迅速な同期 I/O は適しています。高速の I/O 操作で、ファイル システムと記憶域ドライバースタックをバイパス ユーザー バッファとシステムのキャッシュ間で直接データが転送されます。(記憶装置ドライバは、高速な I/O を使用しないされません)。すべてのファイルから読み取られるデータは、高速の I/O の読み取りまたは書き込み要求を受信したときにシステム キャッシュに常駐場合、要求がすぐに満たされます。それ以外の場合、生成される 1 つまたは複数の Irp の原因と、ページフォールトに発生します。この場合、高速の I/O ルーチンがいずれかを返します FALSE、またはページフォールトが処理されるまでに待機状態に、呼び出し元を格納します。高速の I/O ルーチンを返す場合 FALSE、要求された操作に失敗しました、および呼び出し元は IRP を作成する必要があります。

Irp をサポートするために必要なファイル システムとファイル システム フィルターが、高速な I/O をサポートする必要がなくなります。ただし、ファイル システムとファイル システム フィルターは、高速の入出力ルーチンを実装する必要があります。返すための高速な I/O ルーチンを定義する必要がありますファイル システムとファイル システム フィルターが高速な I/O をサポートしていない場合でも FALSE (つまり、高速の I/O ルーチンが機能を実装しません任意)。I/O マネージャーは、(ページング I/O) 以外の同期のファイル I/O 要求を受信したときに最初に高速の I/O ルーチンを呼び出します。高速の I/O ルーチンを返す場合 TRUE 操作が高速の I/O ルーチンによってサービスを提供します。高速の I/O ルーチンを返す場合 FALSE、I/O マネージャーが作成し、IRP を代わりに送信します。

ファイル システム フィルター ドライバは、デバイス オブジェクトをコントロールで I/O をサポートする必要はありません。ただし、ファイル システムにアタッチされているデバイス オブジェクトをフィルター処理またはドライバースタックで、次の下位ドライバにすべての認識できない、または望ましくない Irp を渡すボリュームが必要です。さらに、ボリュームに関連付けられているデバイス オブジェクトをフィルターでは、FastIoDetachDevice を実装する必要があります。

ファイル システム フィルター ドライバーが使用するデバイス オブジェクトの種類

2020/04/24 • [Edit Online](#)

Irp と高速の I/O 要求の有効なディスパッチ ルーチンを記述するには、フィルター ドライバーがこれらの要求を受け取る対象のデバイス オブジェクトのさまざまな種類を理解する必要があります。

ディスク ドライブなどのプラグ アンド プレイ (PnP) デバイス用のドライバーとは異なり、ファイル システム フィルター ドライバーが機能または物理デバイス オブジェクトに作成できません。代わりに、デバイス オブジェクトにコントロールを作成し、デバイス オブジェクトをフィルター処理します。制御デバイス オブジェクトシステムとユーザー モード アプリケーション (CDO) が、フィルター ドライバーを表します。フィルター デバイス オブジェクト(フィルター操作を行います)、特定のファイル システム ボリュームまたはボリュームをフィルター処理の実際の作業を実行します。1 つは、ファイル システム フィルター ドライバーが通常作成 CDO と 1 つ以上が DOs をフィルター処理します。

フィルター ドライバーが受信した I/O 要求の種類ごとに、ターゲット デバイス オブジェクトは、次の 1 つ以上を指定できます。

- フィルター ドライバーの CDO は、ドライバー スタックにアタッチされていません。
- グローバルなファイル システムのキュー内のファイル システム CDO の上にチェーンされているフィルター デバイス オブジェクト
- マウントされたファイル システム ボリューム デバイス オブジェクト (提供) の上にチェーンされているフィルター デバイス オブジェクト

フィルター ドライバーは Irp のディスパッチ ルーチンのセットを 1 つだけと高速の I/O 要求の 1 つを登録できます。そのため、要求の各種類は、1 つのルーチンによって処理する必要があります。要求を適切に処理できるように、要求を受信した上記のどのターゲット デバイス オブジェクトが、このルーチンに確認する必要があります。

このセクションの内容:

[フィルター ドライバーの制御デバイス オブジェクト](#)

[ファイル システムに接続されたデバイス オブジェクトをフィルター処理します。](#)

[ボリュームに接続されているデバイス オブジェクトをフィルター処理します。](#)

フィルター ドライバーのコントロール デバイス オブジェクト

2020/04/24 • [Edit Online](#)

名前付きコントロールデバイスオブジェクト (CDO) を作成して使用するために必要なファイルシステムとは異なり、ファイルシステム フィルタードライバーには CDO が必要ではありません。設定されている場合、この CDO は、オプションで名前を付けることができ、システムへのフィルタードライバーを表します。その役割は、ユーザーモードアプリケーション (または、より一般的ではないカーネル モードドライバー) から i/o 要求を受信し、適切に動作させることです。

ほとんどのファイルシステムフィルタードライバーは、CDO を作成して使用します。ただし、CDO での i/o 要求のサポートはオプションです。このサポートを提供するには、フィルタードライバーが `IoCreateDevice` を呼び出して CDO を作成するときに、オブジェクトのデバイス名を指定する必要があります。ユーザーモードアプリケーションは、`CreateFile` を呼び出して、名前付き CDO へのハンドルを取得し、デバイス名のユーザーモードバージョンを提供できます。

たとえば、"MyLegacyFilter" という架空のカーネルモードドライバーを考えてみます。このドライバーは、次の名前の CDO を作成できます。

```
\Device\MyLegacyFilter
```

とは、この名前を同等のユーザーモードで表示される名前にリンクするために、`io` を呼び出します。これは、MyLegacyFilter のユーザーモードアプリケーションが名前を指定して、カーネルモードドライバーの CDO へのハンドルを開くことができるようにするためです。

```
\\.\MyLegacyFilter
```

が `CreateFile` を呼び出すとき。

フィルタードライバーのコントロールデバイスオブジェクトに送信される i/o 要求の種類

ファイルシステムフィルタードライバーは、コントロールデバイスオブジェクト (CDO) での i/o 操作をサポートするためには必要ありません。ただし、ほとんどのフィルターでは、次の種類の i/o 要求をフィルターの CDO に送信できます。

- `IRP_MJ_CREATE` (ターゲットデバイスオブジェクトへのハンドルを開き、そのハンドルをユーザーアプリケーションに渡す)
- `IRP_MJ_CLEANUP` (ターゲットデバイスオブジェクトへのユーザーモードアプリケーションのハンドルを閉じるため)
- `IRP_MJ_close` (ターゲットデバイスオブジェクトに対して開いている最後のハンドルを閉じるため)
- `Irp_MJ_デバイス_コントロール`、`irp_MJ_ファイル_システム_コントロール`、または `fasystem.servicemodel Odevicecontrol` (プライベート `ioctl` または FSCTLs をフィルタードライバーに送信する)

ただし、ファイルシステムフィルタードライバーによって作成される他のすべてのデバイスオブジェクトとは異なり、CDO はドライブ スタックにアタッチされません。フィルタードライバーの CDO の上または下にデバイスオブジェクトがアタッチされていません。そのため、受信した i/o 要求については、CDO は、それが唯一の目的の受信者であると想定することができます。これは、デバイスオブジェクトのフィルター処理やファイルシステムの CDOs には当てはまりません。したがって、CDO は、受信したすべての IRP を最終的に完了させる必要があります。高速な i/o 要求の場合は、TRUE または FALSE を返す必要があります。

ファイル システムにアタッチされるフィルター デバイス オブジェクト

2020/04/24 • [Edit Online](#)

全体のファイル システムをフィルターするには、ファイル システム フィルター ドライバーは、フィルター デバイス オブジェクトを作成し、グローバル ファイル システムのキュー内のファイル システム ドライバーの CDO 上結び付けます。

ファイル システムに送信される I/O 要求の種類

ファイル システム上に関連付けられているフィルター デバイス オブジェクトは、次の種類の I/O 要求を受信する概してできます。

IRP_MJ_DEVICE_CONTROL

IRP_MJ_ファイル_システム_コントロール

IRP_MJ_シャット ダウン

ファイル システムでは、コントロール、そのデバイス オブジェクトを開く処理をサポートする場合は、同様の I/O 要求の次の種類を確認するフィルターが想定されます。

IRP_MJ_クリーンアップ

IRP_MJ_CLOSE

IRP_MJ_CREATE

ドライバースタックで、次の下位ドライバにすべての認識できない、または望ましくない Irp を渡すには、既定ではファイル システムにアタッチされたファイル システム フィルター デバイス オブジェクトが必要です。

ボリュームにアタッチされるフィルター デバイス オブジェクト

2020/04/24 • [Edit Online](#)

フィルタードライバーは、ボリュームをフィルター処理するために、フィルターデバイスオブジェクトを作成し、ボリュームのボリュームデバイスオブジェクトの上に接続します。

ボリュームに送信される i/o 要求の種類

ボリュームの上にアタッチされるフィルターデバイスオブジェクトは、通常、次の種類の i/o 要求を受け取ることがあります。

IRP_MJ_クリーンアップ

IRP_MJ_閉じる

IRP_MJ_作成

IRP_MJ_デバイス_コントロール

IRP_MJ_DIRECTORY_コントロール

IRP_MJ_ファイル_システム_コントロール

IRP_MJ_フラッシュ_バッファー

IRP_MJ_内部_デバイス_コントロール

IRP_MJ_ロック_コントロール

IRP_MJ_PNP

IRP_MJ_クエリ_EA

IRP_MJ_クエリ_情報

IRP_MJ_クエリ_クォータ

IRP_MJ_クエリ_セキュリティ

IRP_MJ_クエリ_ボリューム_情報

IRP_MJ_読み取り

IRP_MJ_設定_EA

IRP_MJ_設定_情報

IRP_MJ_設定_クォータ

IRP_MJ_設定_セキュリティ

IRP_MJ__ボリューム_情報の設定

IRP_MJ_シャットダウン

IRP_MJ_書き込み

可能な場合

FastIoDetachDevice

Fa_ Odevicecontrol

Fa/Olock

Fa_ Oquerybasicinfo

Fa、Oquerynetworkopeninfo

Fa/Oqueryopen

Faの Oquerystandardinfo

FastIoRead

FastIoReadCompressed

Faと Ounlockall

FastIoUnlockAllByKey

Faと Ounlocksingle

高速

Fa/Owritecom押された状態

MdIoRead

MdIoReadComplete

MdIoReadCompleteCompressed

MdIoWriteComplete

MdIoWriteCompleteCompressed

作成した MdIoWrite

ボリュームにアタッチされているファイルシステムフィルターデバイスオブジェクトは、既定では、すべての認識されていない Irp または望ましくない Irp を、ドライバースタック上の次の下位のドライバに渡す必要があります。また、FastIoDetachDeviceを実装する必要があります。

注 MICROSOFT Windows XP 以降では、次の高速 i/o コールバックルーチンは互換性のために残されています。ファイルシステムフィルタードライバでは使用できません: AcquireForCcFlush

AcquireFileForNtCreateSection

AcquireForModWrite

ReleaseForCcFlush

Releasefileforntのファイル

ReleaseForModWrite

詳細については、[Fsrtlregisterfilesystemfiltercallbacks](#) バックのリファレンスエントリを参照してください。

IRP ディスパッチ ルーチンの記述

2020/04/24 • [Edit Online](#)

NOTE

最適な信頼性とパフォーマンスを得るには、従来のファイルシステムフィルタードライバーではなく、フィルターマネージャーをサポートする[ファイルシステムミニフィルタードライバー](#)を使用します。レガシドライバーをミニフィルタードライバーに移植する方法については、「[レガシフィルタードライバーを移植するためのガイドライン](#)」を参照してください。

ファイルシステムフィルタードライバーは、デバイスドライバーで使用されるものと似たディスパッチルーチンを使用します。ディスパッチルーチンは、1つまたは複数の種類の irp を処理します。(IRP の種類は、主要な関数コードによって決まります)。ドライバークの `driverentry` ルーチンは、ディスパッチルーチンのエントリポイントをドライバークオブジェクトのディスパッチテーブルに格納することによって登録します。IRP がドライバークに送信されると、i/o サブシステムは、IRP の主要な関数コードに基づいて適切なディスパッチルーチン呼び出しします。

各 IRP ディスパッチルーチンは、次のように定義されます。

```
NTSTATUS
(*PDRIVER_DISPATCH) (
    IN PDEVICE_OBJECT DeviceObject,
    IN PIRP Irp
);
```

ファイルシステムフィルタードライバークのディスパッチルーチンは、通常、ユーザーモードのアプリケーションスレッドである i/o 要求を発生させたスレッドのコンテキストで、IRQL PASSIVE_LEVEL で呼び出されます。ただし、このルールにはいくつかの例外があります。たとえば、ページフォールトでは、読み取りおよび書き込みのディスパッチルーチンが IRQL APC_LEVEL で呼び出されます。これらの例外は、[ディスパッチルーチンの IRQL とスレッドコンテキスト](#)の表にまとめられています。残念ながら、現時点では、フィルタークチェーンのドライバークが IRQL > PASSIVE_LEVEL で `IoCallDriver` を呼び出さないようにすることはできません (たとえば、スピンロックや高速ミューテックスの解放に失敗するなど)。ただし、フィルタークディスパッチルーチンは、呼び出されたのと同じ IRQL で常に `IoCallDriver` を呼び出すことを強くお勧めします。

ディスパッチルーチンは、カーネルモードドライバークアーキテクチャ設計ガイドの「[ドライバークのページングを変更する](#)」セクションで説明されている条件を満たす場合に、ページング可能にすることができます。

ファイルシステムフィルタードライバークにコントロールデバイスオブジェクト (CDO) がある場合、そのディスパッチルーチンは、IRP のターゲットデバイスオブジェクトが、マウントされたボリュームのボリュームデバイスオブジェクト (VDO) ではなく、CDO であるケースを検出して処理する必要があります。CDO の詳細については、[フィルタードライバークのコントロールデバイスオブジェクト](#)に関する説明を参照してください。

このセクションでは、次のトピックについて説明します。

[IRP の完了](#)

[IRP を下位レベルのドライバークに渡す](#)

[ディスパッチルーチンから状態を返す](#)

[例: 完了ルーチンを設定せずに IRP を渡す](#)

[ディスパッチルーチンの制約](#)

[ディスパッチルーチンの IRQL とスレッドコンテキスト](#)

IRP の完了

2020/04/24 • [Edit Online](#)

各ディスパッチルーチンは、*DeviceObject* パラメーターで IRP のターゲットデバイスオブジェクトへのポインターを受け取ります。フィルタードライバにコントロールデバイスオブジェクト (CDO) がある場合、ディスパッチルーチンは、IRP で処理を実行する前に、*DeviceObject* ポインターがフィルタードライバの *cdo* を指しているかどうかを確認する必要があります。

ファイルシステムフィルタードライバは、CDO に特に送信される i/o 操作をサポートするためには必要ありません。(一般的にサポートされる操作の詳細については、[フィルタードライバのコントロールデバイスオブジェクト](#)に関する説明を参照してください)。ただし、CDO は、送信されるすべての IRP を完了する必要があります。

IRP を完了するには、ディスパッチルーチンで次のすべての手順を実行する必要があります。

1. `Irp->IoStatus.status` を適切な NTSTATUS 値に設定します。
2. `IoCompleteRequest` を呼び出して、IRP を i/o マネージャーに返します。
3. 手順 1. と同じ状態値を呼び出し元に返します。

IRP の完了は、"成功" または "失敗" と呼ばれることもあります。

- IRP を成功させるには、正常に完了したか、ステータス `_success` などの情報の NTSTATUS 値を使用して IRP を完了します。
- IRP がエラーまたは警告を使用して完了するために失敗する場合は `_無効_デバイス_` の要求または状態 `_バッファー_オーバーフロー_` します。

NTSTATUS 値は、`ntstatus` に定義されています。これらの値は、成功、情報、警告、エラーの4つのカテゴリに分類されます。詳細については、「[NTSTATUS 値の使用](#)」を参照してください。

注 STATUS_pending は成功した NTSTATUS 値ですが、状態_保留中の IRP を完了するためのプログラミングエラーです。

IRP を下位レベルのドライバーに渡す

2020/04/24 • [Edit Online](#)

既定では、IRP のターゲットデバイスオブジェクトを確認した後、は **IoCallDriver** を呼び出すことによって、irp を次の下位レベルのデバイスドライバーに渡す必要があります。フィルタードライバーは、単純に失敗するのではなく、認識されないすべての Irp をパススルーすることが重要です。不明な Irp に失敗すると、オペレーティングシステム自体が予期しない方法で失敗する可能性があります。たとえば、ファイルシステムフィルタードライバーで IRP_MJ_PNP 要求が失敗した場合、システム休止状態を防ぐことによって電源管理が妨げられる可能性があります。これは、ファイルシステムフィルタードライバーが電源管理に含まれておらず、**IRP_MJ_の電源** 要求を受信していない場合にも当てはまります。

ディスパッチルーチンから返される状態

2020/04/24 • [Edit Online](#)

IRP を完了する場合を除き、完了ルーチンを設定しないディスパッチルーチンは、常にIoCallDriverから返された NTSTATUS 値を返す必要があります。この値が [保留中] の場合を除き、irp を完了したドライバーによって設定されたirp>IoStatus.statusの値と一致する必要があります。

IRP をワークキューにポストする可能性のある完了ルーチンを設定するディスパッチルーチンでは、次のいずれかを実行する必要があります。

- IoCallDriverによって返された NTSTATUS 値を返します。
- 完了ルーチンによってイベントが通知され、Irp>IoStatus.Statusの値が返されるまで待機します。
- IRP を保留中としてマークし、作業キューにポストして、状態_保留中に戻します。
- 完了ルーチンが IRP をワークキューにポストする可能性がある場合、ディスパッチルーチンは、IRP pending および return STATUS_PENDING とマークする必要があります。

これらの動作のうち、どれが適切であるか、または可能であるかは、特定の操作によって異なります。ディレクトリ変更通知など、一部の操作は同期できません。また、一部 (oplock など) を非同期にすることはできません。

ディスパッチルーチンから状態を返す方法の詳細については、「[ディスパッチルーチンの制約](#)」を参照してください。

例: 完了ルーチンを設定せずに IRP を渡す

2020/04/24 • [Edit Online](#)

完了ルーチンを設定せずに、IRP を下位レベルのドライバーに渡すには、ディスパッチルーチンで次の操作を行う必要があります。

- 現在の IRP スタックの場所を削除するには、**IoSkipCurrentIrpStackLocation** を呼び出します。これにより、i/o マネージャーは、irp での完了処理を実行するときに、完了ルーチンを検索しません。
- **IoCallDriver** を呼び出して、次の下位レベルのドライバーに IRP を渡します。

この手法を次のコード例に示します。

```
IoSkipCurrentIrpStackLocation ( Irp );  
return IoCallDriver ( NextLowerDriverDeviceObject, Irp );
```

または、同等です。

```
IoSkipCurrentIrpStackLocation ( Irp );  
status = IoCallDriver ( NextLowerDriverDeviceObject, Irp );  
/* log or debugprint the status value here */  
return status;
```

これらの例では、**IoCallDriver** の呼び出しの最初のパラメーターは、次の下位レベルのフィルタードライバーのデバイスオブジェクトへのポインターです。2番目のパラメーターは、IRP へのポインターです。

このアプローチの利点

上記の手法 (**IoSkipCurrentIrpStackLocation** を呼び出すことができます) は単純で効率的であり、ドライバーが完了ルーチンを登録せずに IRP をドライバースタックに渡すすべての場合に使用する必要があります。

このアプローチの欠点

IoCallDriver を呼び出すと、**IoCallDriver** に渡された IRP ポインターは無効になり、安全に逆参照できなくなります。IRP が下位レベルのドライバーによって処理された後、さらに処理やクリーンアップを実行する必要がある場合は、IRP をドライバースタックに送信する前に、完了ルーチンを設定する必要があります。入力と完了ルーチンの設定の詳細については、「[完了ルーチンの使用](#)」を参照してください。

IRP に対して **IoSkipCurrentIrpStackLocation** を呼び出す場合、その IRP の完了ルーチンを設定することはできません。

ディスパッチ ルーチンに関する制約

2020/04/24 • [Edit Online](#)

次のガイドラインでは、ファイルシステムフィルタードライバーのディスパッチルーチンでの一般的なプログラミングエラーを回避する方法について簡単に説明します。

IRQL に関連する制約

注: ページング i/o で使用される Irp の種類の詳細については、「[ディスパッチルーチンの IRQL とスレッドコンテキスト](#)」を参照してください。

- ページング i/o パスのディスパッチルーチンでは、APC_レベルより上の IRQL で [IoCallDriver](#) を呼び出すことはできません。ディスパッチルーチンが IRQL を発生させる場合は、[IoCallDriver](#) を呼び出す前にディスパッチルーチンの値を小さくする必要があります。
- 読み取りや書き込みなどのページングパスにあるディスパッチルーチンは、呼び出し元_を必要とするカーネルモードルーチンを安全に呼び出すことができません。
- ページングファイル i/o パスにあるディスパッチルーチンでは、呼び出し元が実行されている必要のあるカーネルモードルーチンを安全に呼び出すことができません。そのためには、IRQL < ディスパッチ_レベルを使用します。
- ページング i/o パスにないディスパッチルーチンは、パッシブ_レベル上の IRQL で [IoCallDriver](#) を呼び出すことはできません。ディスパッチルーチンが IRQL を発生させる場合は、[IoCallDriver](#) を呼び出す前にディスパッチルーチンの値を小さくする必要があります。

処理の Irp に対する制約

- IRP パラメーターにユーザー領域のアドレスが含まれている場合は、それらを使用する前に検証する必要があります。詳細については、「[バッファリングされた i/o のエラー](#)」を参照してください。
- さらに、IRP に 32 ビットプラットフォームから 64 ビットプラットフォームに送信された IOCTL または FSCTL バッファが含まれている場合は、バッファの内容を thunked にする必要がある場合があります。詳細については、「[64 ビットドライバでの 32 ビット i/o のサポート](#)」を参照してください。
- ファイルシステムとは異なり、ファイルシステムフィルタードライバーは、[ExAcquireFastMutexUnsafe](#) または [ExAcquireResourceExclusiveLite](#) を呼び出す前を除き、[fsrtlnterfilesystem](#) または [fsrtlnterfilesystem](#) を呼び出すことはできません。Fsrtlnterfilesystem および Fsrtlnterfilesystem は、ほとんどのファイルシステムで必要とされる通常のカーネル apc を無効にします。

Irp の完了に関する制約

- Irp を完了する場合、ファイルシステムフィルタードライバーは成功とエラーの状態の値のみを使用する必要があります。
- STATUS_PENDING は成功した NTSTATUS 値ですが、状態_保留中の IRP を完了するには、プログラミングエラーになります。
- ディスパッチルーチンが [IoCompleteRequest](#) を呼び出すと、IRP ポインターは無効になり、安全に逆参照できなくなります。

完了ルーチンの設定に関する制約

注: 完了ルーチンの設定の詳細については、「[完了ルーチンの使用](#)」を参照してください。

- ディスパッチルーチンが [IoSetCompletion Routine](#) を呼び出す場合、必要に応じて、指定された IRP を処理するときに使用する完了ルーチンの構造体にコンテキストポインターを渡すことができます。完了ルーチンを IRQL ディスパッチ_レベルと呼び出すことができるため、この構造体を非ページブルから割り当てる必要があります。
- ディスパッチルーチンによって、_状態を返す可能性のある完了ルーチンが設定されている場合は_処理_必要があります。

i/o マネージャーが IRP を途中で完了しないようにするには、次のいずれかを行う必要があります。

- IRP を pending としてマークし、**IoCallDriver**を呼び出し、状態を返す_保留中にします。
- **KeWaitForSingleObject**を呼び出して、完了ルーチンの実行を待機してから、**IoCompleteRequest**を呼び出して IRP を完了します。

Irp を渡すときの制約

- ディスパッチルーチンが**IoCallDriver**を呼び出すと、IRP ポインターは無効になり、安全に逆参照できなくなります。ただし、ディスパッチルーチンが、呼び出しが完了したことを通知するために完了ルーチンが待機する場合を除きます。
- ファイルシステムフィルタードライバーから**PocallDriver**を呼び出すと、プログラミングエラーになります。(PocallDriverは、IRP_MJ_の電源要求を下位レベルのドライバに渡すために使用されます。ファイルシステムフィルタードライバーは、IRP_MJ_の電源要求を受信することはありません)。

ステータスを返す場合の制約

- IRP を完了する場合を除き、完了ルーチンを設定しないディスパッチルーチンは、常に**IoCallDriver**から返された NTSTATUS 値を返す必要があります。この値が [保留中] の場合を除き、irp を完了したドライバによって設定された `irp->IoStatus. status`の値と一致する必要があります。
- **IoCallDriver**が `_pending` を返した場合、ディスパッチルーチンは、完了ルーチンがイベントを通知するまで待機しない限り、状態 `_pending` を返す必要があります。
- 後で処理するために IRP をワーカーキューにポストする場合、ディスパッチルーチンは、IRP pending および return STATUS_PENDING としてマークする必要があります。
- 後で処理するために IRP をワーカーキューに post する可能性のある完了ルーチンを設定する場合、ディスパッチルーチンは、IRP pending および return STATUS_PENDING とマークする必要があります。
- 保留中の IRP をマークするディスパッチルーチンは、状態 `_PENDING` を返す必要があります。
- Oplock 操作を保留 (worker キューに投稿) することはできません。また、ディスパッチルーチンは、保留中のステータス_を返すことはできません。

Irp を作業キューに投稿する際の制約

- ディスパッチルーチンが Irp を作業キューにポストする場合は、各 IRP をワーカーキューにポストする前に、**IoMarkIrppending**を呼び出す必要があります。それ以外の場合は、IRP をデキューして別のドライバルーチンによって完了し、**IoMarkIrppending**の呼び出しが発生する前にシステムによって解放され、クラッシュが発生する可能性があります。

ディスパッチルーチン IRQL とスレッドコンテキスト

2020/04/24 • [Edit Online](#)

次の表は、ファイルシステムフィルタードライバーのディスパッチルーチンの IRQL とスレッドコンテキストの要件をまとめたものです。

IRQL	IRQL:	スレッドコンテキスト:
クリーンアップ	PASSIVE_LEVEL	任意
Close	APC_LEVEL	勝手
[作成]	PASSIVE_LEVEL	任意
DeviceControl (ページング i/o を除く)	PASSIVE_LEVEL	任意
DeviceControl (ページング i/o パス)	APC_LEVEL	勝手
DirectoryControl	APC_LEVEL	勝手
FlushBuffers	PASSIVE_LEVEL	任意
FsControl (ページング i/o を除く)	PASSIVE_LEVEL	任意
FsControl (ページング i/o パス)	APC_LEVEL	勝手
LockControl	PASSIVE_LEVEL	任意
全て	PASSIVE_LEVEL	勝手
Quer	PASSIVE_LEVEL	任意
QueryInformation	PASSIVE_LEVEL	任意
QueryQuota	PASSIVE_LEVEL	任意
QuerySecurity	PASSIVE_LEVEL	任意

IRQL	IRQL:	IRQL:
QueryVolumeInfo	PASSIVE_LEVEL	任意
読み取り (ページング i/o を除く)	PASSIVE_LEVEL	任意
読み取り (ページング i/o パス)	APC_LEVEL	勝手
SetEa	PASSIVE_LEVEL	任意
SetInformation	PASSIVE_LEVEL	任意
SetQuota	PASSIVE_LEVEL	任意
SetSecurity	PASSIVE_LEVEL	任意
SetVolumeInfo	PASSIVE_LEVEL	任意
シャットダウン	PASSIVE_LEVEL	勝手
書き込み (ページング i/o を除く)	PASSIVE_LEVEL	任意
書き込み (ページング i/o パス)	APC_LEVEL	勝手

IRP 完了ルーチンの使用

2020/04/24 • [Edit Online](#)

NOTE

最適な信頼性とパフォーマンスを得るには、従来のファイルシステムフィルタードライバーではなく、フィルターマネージャーをサポートする[ファイルシステムミニフィルタードライバー](#)を使用します。レガシドライバーをミニフィルタードライバーに移植する方法については、「[レガシフィルタードライバーを移植するためのガイドライン](#)」を参照してください。

ファイルシステムフィルタードライバーは、デバイスドライバで使用するものと同様の完了ルーチンを使用します。完了ルーチンは、IRP で完了処理を実行します。次の下位レベルのドライバに IRP を渡すドライバルーチンは、[IoCallDriver](#)を呼び出す前に[IoSetCompletionRoutine](#) ルーチンを呼び出すことによって、必要に応じて irp の完了ルーチンを登録できます。

各 IRP 完了ルーチンは、次のように定義されます。

```
NTSTATUS
(*PIO_COMPLETION_ROUTINE) (
    IN PDEVICE_OBJECT DeviceObject,
    IN PIRP Irp,
    IN PVOID Context
);
```

完了ルーチンは、任意のスレッドコンテキストで、IRQL <= DISPATCH_LEVEL で呼び出されます。

IRQL DISPATCH_LEVEL で呼び出すことができるため、完了ルーチンは、[IoDeleteDevice](#)などの低い IRQL で呼び出す必要があるカーネルモードルーチンを呼び出すことはできません。同じ理由から、完了ルーチンで使用されるデータ構造は、非ページプールから割り当てられる必要があります。

このセクションでは、次のトピックについて説明します。

完了処理の実行方法

[PendingReturned](#) フラグを確認しています

完了ルーチンから状態を返す

例: 単純なパススルーディスパッチと完了

完了ルーチンの制約

完了処理の実行方法

2020/04/24 • [Edit Online](#)

完了処理は2段階で実行されます。最初の段階は、IRQL <= ディスパッチ_レベルで、任意のスレッドコンテキストで実行されます。この段階では、次のタスクが実行されます。

- IRP に登録されている各完了ルーチンは、最も低い IRP スタックの場所から順に呼び出されます。完了ルーチンが、必要な_処理_より多くの状態_返す場合、完了処理は停止されます。
- IRP にメモリ記述子リスト (MDL) が含まれている場合、MDL によってマップされた物理ページのロックは解除されます。
- I/O 完了の2番目のフェーズは、特別なカーネル APC としてターゲット (要求) スレッドのキューに登録されます。

2番目の段階は、I/O 要求を発信したスレッドのコンテキストで実行されます。これは特殊なカーネル APC として実行されるため、IRQL APC_レベルで実行されます。この段階では、次のタスクが実行されます。

- IRP がバッファリングされた操作を表す場合、`irp->AssociatedIrp`の内容が`UserBuffer->`にコピーされます。
- IRP に MDL が含まれている場合、MDL は解放されます。
- I/O 要求の発信者が操作の最終状態を確認できるように、`irp->IoStatus`の内容が`irp->UserIoStatus`にコピーされます。
- イベントが`irp->UserEvent`に指定されている場合は、シグナルが通知されます。それ以外の場合、この IRP 用のファイルオブジェクトがある場合、そのイベントはシグナル状態になります。
- `IoBuildDeviceIoControlRequest`または`IoBuildSynchronousFsdRequest`を呼び出すことによって IRP が作成された場合は、スレッドの保留中の I/O 要求一覧からデキューされます。
- 呼び出し元が要求した場合、ユーザー APC がキューに登録されます。
- IRP は解放されます。

完了ルーチンが_処理_必要な_の状態を返すために、IRP の完了処理が停止した場合、同じ Irp で`IoCompleteRequest`を呼び出すことによって再開できます。この場合、最初の段階の処理が再開されます。その後、完了ルーチンがステータスを返したドライバーのすぐ上にあるドライバーの完了ルーチンから、_処理_必要な_が増えます。

PendingReturned フラグの確認

2020/04/24 • [Edit Online](#)

完了ルーチンがイベントを通知しない場合は、Irp>PendingReturned たフラグを確認する必要があります。このフラグが設定されている場合、完了ルーチンは**IoMarkIrppending**を呼び出して、保留中の IRP をマークする必要があります。

完了ルーチンから返される状態

2020/04/24 • [Edit Online](#)

ファイルシステムフィルター ドライバー完了ルーチンでは、呼び出し元に次の 2 つの NTSTATUS 値のいずれかの通常返します。

- ステータス_成功
- ステータス_詳細_処理_必要な作業

ステータス_成功は、ドライバーは IRP が終了し、完了 IRP の処理を続行する I/O マネージャーを使用することを示します。

ステータス_詳細_処理_REQUIRED が I/O マネージャーの完了の IRP の処理を停止します。

その他の NTSTATUS 値が返された場合 I/O マネージャーにリセット状態_成功します。

状態の取得の詳細については_詳細_処理_必要に応じてを参照してください[完了ルーチンの制約](#)します。

例: 単純なパススルーディスパッチと完了

2020/04/24 • [Edit Online](#)

IRP の完了ルーチンを設定して IRP を停止するには、ディスパッチルーチンで次の操作を行う必要があります。

- **locopycurrent enti**を呼び出して、現在のスタックの場所から次の下位レベルのドライバーのパラメーターにコピーします。
- **losetcompletion ルーチン**を呼び出して、IRP の完了ルーチンを指定します。
- **IoCallDriver**を呼び出して、次の下位レベルのドライバーに IRP を渡します。

この手法を次のコード例に示します。

ディスパッチルーチン

```
IoCopyCurrentIrpStackLocationToNext( Irp );
IoSetCompletionRoutine( Irp,                                // Irp
                        MyLegacyFilterPassThroughCompletion, // CompletionRoutine
                        (PVOID)recordList,                   // Context
                        TRUE,                                 // InvokeOnSuccess
                        TRUE,                                 // InvokeOnError
                        TRUE);                                // InvokeOnCancel
return IoCallDriver ( NextLowerDriverDeviceObject, Irp );
```

この例では、**losetcompletion ルーチン**を呼び出して、IRP の完了ルーチンを設定します。

losetcompletion ルーチンの呼び出しの最初の2つのパラメーターは、IRP へのポインターと、完了ルーチンの名前です。3番目のパラメーターは、完了ルーチンに渡されるドライバー定義の構造体へのポインターです。この構造体には、IRP で完了処理を実行するときに完了ルーチンが必要とするコンテキスト情報が含まれます。完了ルーチンは IRQL ディスパッチ_レベルで呼び出すことができるので、非ページプールからコンテキスト構造を割り当てる必要があります。

losetcompletion ルーチンに渡される最後の3つのパラメーターは、i/o 要求が成功したとき、失敗したとき、または取り消されたときに、完了ルーチンを呼び出すかどうかを指定するフラグです。

完了ルーチン

ディスパッチルーチンが完了ルーチンを設定し、**IoCallDriver**を呼び出した後（上記のディスパッチルーチンに示されているように）すぐに戻る場合は、対応する完了ルーチンが IRP の pendingreturned フラグをチェックし、設定されている場合は、を呼び出します。**IoMarkIrpPending**。その後、次の例に示すように、STATUS_SUCCESS が返されます。

```
if (Irp->PendingReturned) {
    IoMarkIrpPending( Irp );
}
return STATUS_SUCCESS;
```

このアプローチの利点

完了ルーチンを設定すると、ドライバーは、下位レベルのドライバーによって処理された後に、IRP をさらに処理することができます。完了ルーチンは、要求された i/o 操作の結果に基づいて、IRP を処理する方法を決定できます。

このアプローチの欠点

IRQL <= ディスパッチ_レベルでは任意のスレッドコンテキストで実行されるため、完了ルーチンは、IRP に対して制限付きの処理のみを実行できます。

完了ルーチンに関する制約

2020/04/24 • [Edit Online](#)

次のガイドラインでは、ファイルシステムフィルタードライバーの完了ルーチンでの一般的なプログラミングエラーを回避する方法について簡単に説明します。

IRQL に関連する制約

完了ルーチンは IRQL ディスパッチ レベルで呼び出すことができるため、次の制約が適用されます。

- 完了ルーチンは、[IoDeleteDevice](#)や[ObQueryNameString](#)など、低い IRQL を必要とするカーネルモードルーチンを安全に呼び出すことができません。
- 完了ルーチンで使用するデータ構造は、非ページプールから割り当てられる必要があります。
- 完了ルーチンをページング可能にすることはできません。
- 完了ルーチンは、リソース、ミューテックス、または高速ミューテックスを取得できません。ただし、スピンロックを取得することはできます。

PendingReturned たフラグを確認しています

- 完了ルーチンがイベントを通知する場合を除き、[Irp>PendingReturned](#) たフラグを確認する必要があります。このフラグが設定されている場合、完了ルーチンは[IoMarkIrppending](#)を呼び出して、IRP を pending としてマークする必要があります。
- 完了ルーチンがイベントを通知する場合、[IoMarkIrppending](#)を呼び出すことはできません。

ステータスを返す場合の制約

- ファイルシステムフィルタードライバーの完了ルーチンでは、必要な `_処理_` の状態 `_成功` または状態のいずれかを返す必要があります。その他のすべての NTSTATUS 値は、i/o マネージャーによって `[状態]_[成功]` にリセットされます。

ステータスを返すための制約 `_処理_` 必要があります

完了ルーチンがステータスを返す必要がある場合は、次の3つのケース `_処理_` 必要があります。

- 対応するディスパッチルーチンが、完了ルーチンによってイベントが通知されるのを待機している場合。この場合は、完了ルーチンが返された後に i/o マネージャーが IRP を途中で完了しないようにするために、ステータス `_より多くの_処理_` 必要があります。
- 完了ルーチンが IRP をワーカーキューにポストし、対応するディスパッチルーチンが状態 `_PENDING` を返した場合。このケースでは、完了ルーチンが返された後に i/o マネージャーが IRP を途中で完了しないようにするために、ステータス `_より多くの_処理_` 必要があります。
- 同じドライバーが[IoAllocateIrp](#)または[IoBuildAsynchronousFsdRequest](#)を呼び出すことによって IRP を作成したとき。ドライバーは、上位レベルのドライバーからこの IRP を受信しなかったため、完了ルーチンで IRP を安全に解放できます。[IoFreeIrp](#)を呼び出した後、完了ルーチンは、それ以上の完了処理が不要であることを示すために必要な `_処理_` のステータス `_を返す_` 必要があります。

完了ルーチンは、oplock 操作に必要な `_処理_より多くの_状態` を返すことはできません。Oplock 操作を保留 (worker キューに投稿) することはできません。また、ディスパッチルーチンは、保留中のステータス `_を返す_` ことはできません。

Irp を作業キューに投稿する際の制約

- 完了ルーチンが Irp を作業キューにポストする場合は、各 IRP をワーカーキューにポストする前に、[IoMarkIrppending](#)を呼び出す必要があります。それ以外の場合は、IRP をデキューして別のドライバールーチンによって完了し、[IoMarkIrppending](#)の呼び出しが発生する前にシステムによって解放され、クラッシュが発生する可能性があります。

ファイル ストリーム、ストリーム コンテキスト、ストリーム別コンテキスト

2020/04/24 • [Edit Online](#)

ファイルストリームは、ファイルデータを保持するために使用されるバイトのシーケンスです。通常、ファイルには、ファイルストリーム (ファイルの既定のデータストリーム) が1つだけあります。ただし、複数のデータストリームをサポートするファイルシステムでは、各ファイルに複数のファイルストリームを含めることができます。これらのうちの1つは、名前のない既定のデータストリームです。その他の名前は代替データストリームです。ファイルを開くと、実際には、指定されたファイルのストリームが開かれます。

ファイルシステムがファイルストリームを初めて開くと、ファイル制御ブロック (FCB) やストリーム制御ブロック (SCB) などのファイルシステム固有のストリームコンテキスト構造が作成され、この構造体のアドレスが `fscontext` に格納されます。結果として得られるファイルオブジェクトのメンバー。

ローカルファイルシステムの場合、既に開いているファイルストリームが再び開かれた場合 (共有読み取りアクセスの場合など)、i/o サブシステムは別のファイルオブジェクトを作成しますが、ファイルシステムは新しいストリームコンテキストを作成しません。両方のファイルオブジェクトは、同じストリームコンテキスト構造のアドレスを受け取ります。したがって、ローカルファイルシステムの場合、ストリームコンテキストポインターはファイルストリームを一意に識別します。

ストリームごとのコンテキストをサポートするネットワークファイルシステムの場合、既に開いているファイルストリームが同じネットワーク共有名または IP アドレスを使用して再度開かれると、ローカルファイルシステムの場合と同じ動作になります。i/o サブシステムは新しいファイルオブジェクトを作成しますが、ファイルシステムは新しいストリームコンテキストを作成しません。代わりに、同じ `Fscontext` ポインター値が両方のファイルオブジェクトに割り当てられます。ただし、ファイルストリームが別のパス (たとえば、共有名を使用して以前に開かれたファイルの IP アドレス) を使用して開かれている場合は、ファイルシステムによって新しいストリームコンテキストが作成されます。このため、ストリームごとのコンテキストをサポートするネットワークファイルシステムでは、`Fscontext` ポインターはファイルストリームを一意に識別しません。

ストリームごとのコンテキストとは、そのメンバーの1つとして、[_ストリーム_コンテキスト構造の FSRTL_](#) を含むフィルター定義の構造体です。フィルタードライバーは、この構造体を使用して、ファイルシステムによって開かれている各ファイルストリームに関する情報を追跡します。

ファイルシステムでのストリームごとのコンテキストのサポート

Microsoft Windows XP 以降では、ストリームごとのコンテキストをサポートするファイルシステムでは、[FSRTL_ADVANCED_FCB_ヘッダー](#) 構造を含むストリームコンテキスト構造を使用する必要があります。

特定のファイルストリームに関連付けられているストリームごとのコンテキストのグローバルリストは、ファイルシステムによって所有されています。ファイルストリームの新しいストリームコンテキスト (`FSRTL_ADVANCED_FCB_HEADER` オブジェクト) がファイルシステムによって作成されると、[Fsrtlsetupadvanced ヘッダー](#) が呼び出され、このリストが初期化されます。ファイルシステムフィルタードライバーが `Fsrtlinsertperstreamcontext` を呼び出すと、フィルターによって作成されたストリームごとのコンテキストがグローバルリストに追加されます。

ファイルストリームのストリームコンテキストを削除すると、ファイルシステムは `FsRtlTeardownPerStreamContexts` を呼び出し、フィルターがファイルストリームに関連付けたストリームごとのコンテキストをすべて解放します。このルーチンは、グローバルリスト内のストリームごとのコンテキストごとに `FreeCallback` ルーチンを呼び出します。`FreeCallback` ルーチンは、ファイルストリームのファイルオブジェクトが既に解放されていることを前提としている必要があります。

ファイルシステムが、指定されたファイルオブジェクトによって表されるファイルストリームのストリームごとのコンテキストをサポートするかどうかを照会するには、ファイルオブジェクトに対して `Fsrtlsupportsperstreamcontexts` を呼び出します。ファイルシステムでは、ファイルの種類によってはストリームごとのコンテキストがサポートされる場合がありますが、それ以外はサポートされないことに注意してください。たとえば、NTFS および FAT では、現在、ページングファイルのストリームごとのコンテキストをサポートしていません。したがって、`Fsrtlsupportsperstreamcontexts` が1つのファイルストリームに対して `true` を返す場合、すべてのファイルストリームに対して `true` が返されるとは限りません。

ストリーム別コンテキストの作成

2020/04/24 • [Edit Online](#)

FSRTL_PER_STREAM_CONTEXT 構造体を含むストリームごとのコンテキスト構造を使用する従来のファイルシステムフィルタードライバーでは、MICROSOFT Windows XP 以降でのストリームごとのコンテキストサポートを利用できます。

ファイルシステムフィルタードライバーは、通常、ファイルストリームが最初に開かれたときに、ファイルストリームの**ストリームごとのコンテキスト構造**を作成します。ただし、任意の操作中に、ストリームごとのコンテキスト構造を作成し、ファイルストリームに関連付けることができます。

ストリームごとのコンテキストの割り当て

ストリームごとのコンテキスト構造は、ページングされたプールまたは非ページプールから割り当てることができます。ストリームごとのコンテキストを割り当てるには、次の例に示すように、**ExAllocatePoolWithTag**を呼び出します。

```
contextSize = sizeof(SPY_STREAM_CONTEXT) + fileName.Length;  
ctx = ExAllocatePoolWithTag(NonPagedPool,  
                             contextSize,  
                             MYLEGACYFILTER_CONTEXT_TAG);
```

NOTE

フィルターが、ストリームごとのコンテキスト構造をページプールから割り当てた場合、その作成完了ルーチンから**ExAllocatePoolWithTag**を呼び出すことはできません。これは、完了ルーチンは IRQL DISPATCH_LEVEL で呼び出すことができるためです。

ストリームごとのコンテキストの初期化

ファイルシステムフィルタードライバーは、ストリームごとのコンテキスト構造を初期化するために、**Fsrtlinitperstreamcontext**を呼び出します。このルーチンは、コンテキスト構造の FSRTL_PER_STREAM_CONTEXT の部分を初期化します。(構造体の残りの部分はフィルタードライバーに固有です)。

NOTE

フィルタードライバーがファイルストリームごとにストリームごとのコンテキスト構造を1つだけ作成する場合は、*InstanceId*パラメーターに**NULL**を渡す必要が 

フィルタードライバーは、ストリームごとのコンテキストをいつでも初期化できます。ただし、コンテキストをファイルストリームに関連付ける前に実行する必要があります。

ストリームごとのコンテキストのファイルストリームへの関連付け

2020/04/24 • [Edit Online](#)

ストリームごとのコンテキスト構造は、ファイルシステムが `IRP_MJ_CREATE` 要求を正常に処理してストリームを開くと、ファイルストリームに関連付けることができます。これは、ファイルシステムが `create` 要求を処理した後に、ファイルシステムフィルタードライバーによってファイルオブジェクトの `FsContext` ポインターが有効と見なされることがあるためです。`FsContext` ポインターはファイルストリームを一意に識別するため、ファイルオブジェクトが既に表示されているファイルストリームを表しているかどうかを確認する必要があります。この場合、フィルタは既にストリームごとのコンテキストを作成しています。このため、`create dispatch` (または "事前作成") パスにストリームごとのコンテキストを作成するフィルタは、重複していると判明しているため、作成完了 (または "作成後") パスでのみ削除することは珍しくありません。

同じファイルストリームを使用して、ストリームごとに別のコンテキストが既に関連付けられているかどうかを確認するために、ファイルシステムフィルタードライバーは `FsRtlLookupPerStreamContext` を呼び出します。

`FsRtlLookupPerStreamContext` が同じファイルストリームのストリームごとの既存のコンテキストを検出した場合、フィルタは、新しく作成されたストリームごとのコンテキストを削除する必要があります。

`FsRtlLookupPerStreamContext` が、ファイルストリーム用に既に作成済みのストリームコンテキストを検出できない場合、フィルタは `Fsrtlinstertperstreamcontext` を呼び出して、新しく作成されたストリームコンテキストをファイルストリーム。

ストリームごとのコンテキストに対して `Fsrtlinstertperstreamcontext` が呼び出された後、ファイルシステムはそのコンテキストを削除して解放することを前提としています。フィルタードライバーがストリームごとのコンテキストを割り当て、それに対して `Fsrtlinstertperstreamcontext` を呼び出さない場合でも、フィルタードライバーは、`exfreepool` を呼び出すことによってそれを解放する必要があります。

ストリーム別コンテキストの削除

2020/04/24 • [Edit Online](#)

ファイルストリームに関連付けられているストリームごとのコンテキストが不要になった場合は、削除する必要があります。ストリームコンテキストは、次の2つの方法のいずれかで削除されます。

- 手動で、フィルタードライバーが**`Fsrtlremoveperstreamcontext`**を呼び出します。
- 自動的に、ファイルシステムが**`FsRtlTeardownPerStreamContexts`**を呼び出し、その後、ストリームコンテキストの**`FreeCallback`**ルーチンが呼び出されます。

フィルターがストリームごとのコンテキストを削除する場合

ファイルストリームがまだ開いている間に、フィルタードライバーがファイルストリームのストリームごとのコンテキストを削除する必要がある場合、最初に**`Fsrtlremoveperstreamcontext`**を呼び出して、指定されたファイルに関連付けられているコンテキストのグローバルリストからコンテキストを削除します。**`Fsrtlremoveperstreamcontext`**を呼び出した後は、通常、このフィルターによってコンテキスト構造が解放されます。

注 は、フィルタードライバーが**`Fsrtlinsertperstreamcontext`**を呼び出してストリームごとのコンテキスト構造をファイルストリームに関連付けると、それを解放する前に、そのコンテキストに対して**`fsrtlremoveperstreamcontext`**を呼び出す必要があることに注意してください。それ以外の場合、ファイルストリームが閉じられると、システムがクラッシュします。

ストリームごとのコンテキストの **`FreeCallback`** が呼び出されたとき

ファイルストリームを閉じたり削除したりすると、ファイルシステムによってファイルストリームのストリームコンテキストが解放されます。この時点でも、ファイルシステムは**`FsRtlTeardownPerStreamContexts`**を呼び出し、そのファイルストリームのコンテキストのグローバルリストに含まれるすべてのストリームコンテキストに対して登録されている**`FreeCallback`**ルーチンを呼び出します。(**`FreeCallback`**ルーチンは、フィルタードライバーが**`Fsrtlinitperstreamcontext`**を呼び出してストリームごとのコンテキスト構造を初期化するときに登録されます。詳細については、「**`FSRTL_PER_STREAM_CONTEXT`**」を参照してください)。

注 フィルタードライバーが**`Fsrtlinsertperstreamcontext`**を呼び出してストリームごとのコンテキスト構造をファイルストリームに関連付けると、ファイルシステムは、ストリームへの開いている参照がなくなったときに、フィルターのストリームごとのコンテキストの**`FreeCallback`**ルーチンが呼び出されることを保証します。

レガシ ファイル システム フィルター ドライバー 内のファイル別コンテキストの追跡

2020/04/24 • [Edit Online](#)

NOTE

最適な信頼性とパフォーマンスを得るには、従来のファイルシステムフィルタードライバーではなく、フィルターマネージャーをサポートする[ファイルシステムミニフィルタードライバー](#)を使用します。レガシドライバーをミニフィルタードライバーに移植する方法については、「[レガシフィルタードライバーを移植するためのガイドライン](#)」を参照してください。

レガシファイルシステムフィルタードライバーは、[FSRTL_PER_FILE_CONTEXT](#)オブジェクトをユーザー定義のコンテキスト情報構造に関連付けることによって、ファイルのコンテキスト情報を記録できます。

NOTE

すべてのファイルシステムでファイルごとのコンテキストオブジェクトがサポートされるわけではありません。ファイルがサポートされているファイルシステムに関連付けられているかどうかを確認するには、[FsRtlSupportsPerFileContexts](#)マクロを使用します。

[FsRtlInitPerFileContext](#)マクロを使用して、FSRTL_PER_FILE_CONTEXT オブジェクトを初期化します。次に、[FsRtlInsertPerFileContext](#)ルーチンを使用して、ファイルを任意のコンテキストオブジェクトに関連付けます。

ファイルシステムランタイムライブラリ (FSRTL) パッケージでファイルコンテキストを追跡するために使用されるポインターを取得するには、[FsRtlGetPerFileContextPointer](#)マクロを使用します。

フィルタードライバーは、[FsRtlLookupPerFileContext](#)ルーチンを使用して、ファイルに関連付けられているファイルコンテキストオブジェクトを見つけることができます。ルーチンでは、構造体の所有者または構造体のインスタンスを指定して、検索を絞り込むことができます。

フィルタードライバーは、[FsRtlRemovePerFileContext](#)を使用してコンテキストオブジェクトを削除できます。ルーチンでは、構造体の所有者または構造体のインスタンスを指定して、検索を絞り込むことができます。

NOTE

ファイルがまだ開いている間は、[FsRtlRemovePerFileContext](#)ルーチンを使用してコンテキストオブジェクトを削除してください。[FsRtlTeardownPerFileContexts](#)と混同しないようにしてください。

ファイルシステムは[FsRtlTeardownPerFileContexts](#)を呼び出して、ファイル制御ブロック構造 (FCB) に関連付けられているすべてのフィルターコンテキストが解放されるのを解放します。FsRtlTeardownPerFileContextsルーチンは、各フィルターコンテキストの FSRTL_PER_FILE_CONTEXT オブジェクトに指定されている[FreeCallback](#)ルーチンを呼び出します。

レガシ ファイル システム フィルター ドライバー のブロック

2020/04/24 • [Edit Online](#)

Windows 10 バージョン1607以降では、管理者とドライバー開発者はレジストリ設定を使用して、レガシファイルシステムフィルタードライバーをブロックできます。レガシファイルシステムフィルタードライバーは、ファイルシステムスタックに直接アタッチされるドライバーであり、フィルターマネージャーは使用しません。このトピックでは、レガシファイルシステムフィルタードライバーをブロックおよびブロック解除するためのレジストリ設定について説明します。また、レガシファイルシステムフィルターがブロックされたときにシステムイベントログに入力されたイベントについて説明し、OS でレガシファイルシステムドライバーが実行されているかどうかを確認する方法についても説明します。

NOTE

最適な信頼性とパフォーマンスを得るには、従来のファイルシステムフィルタードライバーではなく、フィルターマネージャーをサポートする[ファイルシステムミニフィルタードライバー](#)を使用します。レガシドライバーをミニフィルタードライバーに移植する方法については、「[レガシフィルタードライバーを移植するためのガイドライン](#)」を参照してください。

レガシドライバーをブロックする方法

IoBlockLegacyFsFiltersレジストリキーを使用して、システムがレガシファイルシステムフィルタードライバーをブロックするかどうかを指定します。ブロックされると、すべてのレガシファイルシステムフィルタードライバーの読み込みがブロックされます。レジストリの変更を有効にするには、システムの再起動を実行します。

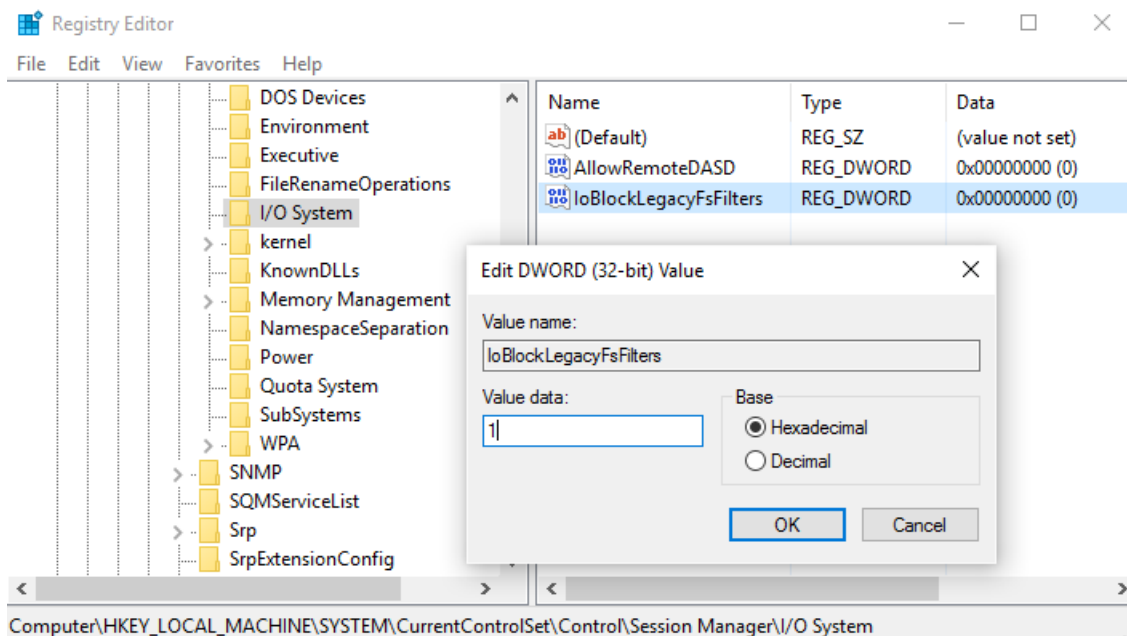
レジストリキーは、次のレジストリパスの下に作成する必要があります。

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\I/O System
```

IoBlockLegacyFsFiltersキーの有効な DWORD 値は次のとおりです。

IOBLOCKLEGACYFSFILTERSI	II
1	レガシファイルシステムフィルタードライバーは、記憶域ボリュームへの読み込みまたはアタッチがブロックされています。
0	レガシファイルシステムフィルタードライバーはブロックされません。このリリースでは、これが既定の動作です。

レジストリエディターでは、次のようなキーが表示されます。



例: レガシドライバーの読み込みがブロックされている場合

次に示すように、レガシファイルシステムフィルタードライバーの読み込みがブロックされると、エラーイベントがシステムイベントログに記録されます。

ログの名前	System
Source	Microsoft-Windows-カーネル-IO
日付	12/29/2015 2:55:05 PM
イベント ID	1205
タスク カテゴリ	なし
レベル	エラー
キーワード	
ユーザー	CONTOSO\user
コンピューター	user.domain.corp.contoso.com
説明	Windows は、レガシファイルシステムフィルタをブロックするように構成されています。フィルタ名: \Driver\sfiler

レガシドライバーが実行されているかどうかを確認する方法

どのフィルタがレガシファイルシステムフィルタードライバーであるかがわからない場合、または実行されていないことを確認するには、次の操作を実行します。

1. Cmd.exeアイコンを右クリックし、[管理者として実行] をクリックして、管理者特権でのコマンドプロンプトを開きます。
2. 型: `fltmc filters`

3. レガシドライバーを検索します。これは、フレーム値が <レガシ> のものです。

この例では、AVLegacy および EncryptionLegacy という名前のレガシファイルシステムフィルタードライバーは、<従来の> フレーム値でマークされています。AVMiniFilter という名前のファイルシステムドライバーには、ミニフィルタードライバー (ファイルシステムスタックに直接アタッチされず、フィルターマネージャーを使用) であるため、<レガシ> フレーム値がありません。

```
C:\Windows\system32>fltmc filters
```

Filter Name	Num Instances	Altitude	Frame
AVLegacy		389998.99	<Legacy>
EncryptionLegacy		149998.99	<Legacy>
AVMiniFilter	3	328000	0

レガシドライバーが実行中であることがわかっている場合は、従来のファイルシステムフィルタードライバーをブロックした後で、IoBlockLegacyFsFiltersレジストリキーを設定した後にシステムを再起動してください。この設定は、再起動後に有効になります。

システムでレガシファイルシステムフィルタードライバーが使用されている場合は、それぞれの Isv と連携して、ファイルシステムドライバーのミニフィルターバージョンを取得します。フィルターマネージャーモデルを使用するミニフィルタードライバーに従来のファイルシステムフィルタードライバーを移植する方法の詳細については、「[レガシフィルタードライバーを移植するためのガイドライン](#)」を参照してください。

リモート ファイル システムの概要

2020/04/24 • [Edit Online](#)

リモート ファイル システムには、別のコンピューターに格納されているファイルにアクセスするクライアントコンピューターで実行されるアプリケーションが有効にします。リモート ファイル システム (リモート プリンター、たとえば) その他のリソースを作成も多くの場合、クライアントコンピューターからアクセスできます。リモートのファイルとリソースのアクセスを使用して、何らかの形式のローカル エリア ネットワーク (LAN)、ワイド エリア ネットワーク (WAN)、ポイント ツー ポイントリンク、またはその他の通信メカニズムを使用して行われます。これらのファイル システムは多くの場合、ネットワーク ファイル システムまたは分散ファイル システムと呼ばれます。

Microsoft ネットワークは、Windows Server 2003、Windows XP、および Windows 2000 に含まれるネイティブのリモート ファイル システムです。Microsoft ネットワークは、ファイルへのリモート アクセスと、リモート プリンターやプロッターへのアクセスを提供します。Microsoft ネットワークには、Windows の以前のバージョンで LAN Manager のネットワークが呼び出されました。

Microsoft では、Windows 上の他のいくつかのリモート ファイル システムのサポートも含まれます。

- NetWare 中核プロトコル (NCP)
- WebDAV (リモート web サーバーを使用するファイル アクセス)
- AppleTalk ファイルとプリンターの共有 (サポートへの接続 Macintosh クライアントから Windows Server 2003 および Windows 2000 Server を実行しているシステム)
- Network File System (NFS)
- IBM メインフレームと VSAM/400 ファイル アクセス、Microsoft Host Integration Server 2000 に含まれています。

注 より前の Windows Server 2003 R2、Windows オペレーティング システムのバージョンでは、Microsoft Services for UNIX をインストールすることで NFS を取得する可能性があります。Windows Server 2003 R2、Windows オペレーティング システムの以降のバージョンの NFS は、Windows のオプション コンポーネントとして含まれています。

リモート ファイル システムは、ソフトウェア コンポーネントのコレクションによって実装されます。必要なソフトウェア コンポーネントの複雑さと数は、設計と、リモート ファイル システムの複雑さに基づいて異なります。

クライアント システム上のソフトウェアは、リモート ファイル とリソースへのアクセスを提供します。このクライアントソフトウェアは、ローカルの呼び出しをいくつかのリモート サーバーへのファイル操作の転送の「ネットワーク リダイレクター」として機能します。このネットワーク リダイレクターにより、リモート リソースのローカルな場合と表示されます。

サーバー システムのソフトウェアで実装して、リモート ファイル サーバーの操作そのローカル ファイル ストレージにアクセスまたはリソース サーバーします。ファイル サーバーで処理されるクライアント ネットワーク リダイレクターから要求を受信して、応答がクライアントに送信されます。

システムは、他のクライアントにサーバーと一部のシステムに、クライアントの両方として動作できます。そのため、1 つのシステムで実行されているクライアントとサーバーの両方のソフトウェアを見つけるには一般的です。

このセクションには、次のトピックが含まれています。

[リモート ファイル システムのコンポーネント](#)

リモート ファイル システムのコンポーネント

2020/04/24 • [Edit Online](#)

リモートファイルシステムを実装するには、次の基本的な要素が必要です。

- クライアントにインストールされているネットワークリダイレクターソフトウェア。
- 通信に使用される、適切に定義されたトランスポートプロトコル。
- サーバーにインストールされているファイルサーバーソフトウェア。

たとえば、Microsoft ネットワークリモートファイルシステムは次のように実装されます。

- Microsoft ネットワーク用クライアントソフトウェアには、クライアントネットワークリダイレクターコンポーネントが用意されています。
- Common Internet File System (CIFS) は、サーバーメッセージブロック (SMB) プロトコルとも呼ばれ、通信に使用されるネットワークトランスポートプロトコルを定義します。
- LAN Manager サーバー (SMB サーバーとも呼ばれます) は、ファイルサーバーサービスを提供します。

クライアントにインストールされるネットワークリダイレクターソフトウェアは、ユーザーモードで動作し、カーネルモードで動作するいくつかのソフトウェアコンポーネントで構成されています。

ネットワーク リダイレクターについて

2020/04/24 • [Edit Online](#)

ネットワークリダイレクターは、ファイルとリモート システム上の他のリソース (プリンターやプロッターの例では、) にアクセスするために使用するクライアント コンピューターにインストールされているソフトウェア コンポーネントで構成されます。ネットワーク リダイレクターを送信します (またはリダイレクト)、要求の処理をリモート サーバーにローカル クライアント アプリケーションからのファイル操作の要求。ネットワーク リダイレクターは、ローカルのアプリケーションに返される、リモート サーバーから応答を受信します。ネットワーク リダイレクター ソフトウェアは、ローカル ファイルとリソースとしては、同じリモート ファイルとリソース、および使用し、同じ方法で操作することができますにクライアント システム上、外観に作成します。

ネットワーク リダイレクターがリモート リソースにアクセスをローカル クライアント アプリケーションをできるだけ透過的に作成しようとする とします。ネットワークの問題がある場合は何回か前に、ネットワーク リダイレクターを放棄し、クライアント アプリケーションにエラーが返されますを要求に再試行されます。

このセクションには、次のトピックが含まれます。

[ネットワーク リダイレクターの基本的なアーキテクチャ](#)

ネットワーク リダイレクターの基本的なアーキテクチャ

2020/04/24 • [Edit Online](#)

ネットワークリダイレクターの一部として、次の基本的なソフトウェアコンポーネントが必要です。

- ネットワークリダイレクター機能を提供するカーネルモードのファイルシステムデバイスドライバ (SYS)。
- ユーザーモードのダイナミックリンクライブラリ (DLL)。クライアントユーザーアプリケーションに対して、ファイル以外の操作のためのネットワークプロバイダーインターフェイスへのアクセスを提供し、ネットワークリダイレクター機能を提供するカーネルモードのファイルシステムドライバとの通信を可能にします。

ネットワークリダイレクターのカーネルモードファイルシステムドライバコンポーネントは、リモートファイルアクセスをオペレーティングシステムに統合するために、オブジェクトマネージャー、キャッシュマネージャー、メモリマネージャー、i/o システム、およびその他のカーネルシステムとやり取りします。ドライバは、リモートファイルアクセスのために、i/o マネージャーを介してクライアントアプリケーションから要求を受信し、必要な情報をネットワーク経由でリモートファイルサーバーに送信します。また、ドライバは、リモートファイルサーバーからの応答を受信し、i/o マネージャーを介してクライアントアプリケーションに情報を返します。また、ネットワークリダイレクタードライバは、リモートアクセスに使用するネットワークプロトコルを実装するか、このネットワークプロトコルまたは通信機構を実装する別のカーネルドライバ、ユーザーモードアプリケーション、またはサービスの呼び出しを提供する必要があります。このネットワーク通信を実装する最も一般的な方法は、Transport Driver Interface (TDI) を使用して、既存のネットワークプロトコルスタック (TCP/IP など) を呼び出すことです。ネットワークプロトコルスタックは、ネットワークドライバインターフェイス仕様 (NDIS) とネットワークインターフェイスカードを使用して、最下層で通信します。この通信機構には、アプリケーション固有のインターフェイスを使用することもできます。

ユーザーモード DLL は、ファイル操作ではないネットワークプロバイダーインターフェイスを使用して、クライアントからの特別な要求を受信します。これらの要求を使用して、ネットワークプロバイダーの機能の決定、リモートネットワークリソースの列挙、リモートネットワークへのログオン、ネットワークパスワードの変更、リモートサーバーとの接続、リモートネットワーク共有またはリソースのマウントを行うことができます。リモート共有から切断しています。ユーザーモード DLL は、リモートファイルシステムにアクセスするための WNet ユーザーモード Api と同等の実装しています。次に、この DLL は、特殊な IOCTL 呼び出しを通じてカーネルモードのネットワークリダイレクタードライバと通信し、これらのクライアント要求を満たします。たとえば、この DLL は、ネットワークリソースの列挙 (ネットワークコンピューターの参照) またはリモートファイル共有とリソース (プリンター) への接続時に、エクスプローラーによって呼び出されます。この DLL は、リモートリソース (`net view \\computername`) を表示し、リモートファイル共有 (`net use`) に接続または切断する `net` コマンドラインツールからも呼び出されます。この DLL は、複数のプロバイダールーター (MPR) の公開された WNet Api を使用して、カスタムアプリケーションから呼び出すこともできます。

ネットワークリダイレクターには、他にもいくつかのコンポーネントが必要になる場合があります。

- グローバルデータテーブルをユーザーモードの DLL に安全に格納する必要がある場合は、ユーザーモードの DLL とカーネルモードドライバの仲介者として機能するためにサービスアプリケーションが必要になることがあります。ユーザーモードの DLL は、クライアントアプリケーションのプロセス空間でインスタンス化されるため、すべてのクライアントアプリケーションでグローバルにする必要がある内部データテーブルをセキュリティで保護することはできません。クライアントごとのデータは、ユーザーモードのネットワークプロバイダー DLL に接続するときに、各クライアントアプリケーションに対してプライベートにすることができますが、グローバルデータは含まれません。サービスアプリケーション (.exe ファイル) は、ユーザーモードの DLL と、グローバルデータテーブルが格納されているカーネルドライバとの間の仲介役として使用できます。たとえば、Microsoft ネットワークでは、`svchost.exe` に組み込まれているワークステーションサービスを使用して、"`net use`" 接続のグローバルテーブルを格納し、カーネルモードドライバの仲介役として機能します。Microsoft ネットワークの場合、ユーザーモード DLL (`ntlanman.dll`) は `svchost.exe` でワークステーションサービスを呼び出します。これにより、ネットワークリダイレクター機能を提供するカーネルモードドライバ `mrxsmb` が呼び出されます。
- システムにネットワークリダイレクターコンポーネントをインストールし、適切に登録するためのシステム。必要なレジストリエントリの作成が含まれている必要があります。また、ソフトウェアが不要になった場合は、ネットワークリダイレクターをアンインストール

トールする方法もあります。Microsoft Systems Installer (MSI) または Windows INF のスクリプトファイルを使用して、このセットアップを実行できます。このインストールとセットアップは、カスタムインストールプログラムを使用して行うこともできます。

- ネットワークリダイレクターがオペレーティングシステムに含まれていないネットワークプロトコル (たとえば、Xerox Network Services) を使用している場合は、カスタムカーネルモードの TDI ドライバーをインストールする必要があります。通信に使用するネットワークプロトコルを実装する新しい TDI ドライバーをインストールする必要があります。カーネルモードのネットワークリダイレクタードライバは、カスタム TDI ドライバーの上端に接続します。カスタム TDI ドライバーは、NDIS ドライバーを使用して、より低いエッジで通信するようになります。

ネットワークリダイレクターで TCP/IP プロトコルが使用されている場合、カスタムの TDI ドライバーは必要ありません。この場合、ネットワークリダイレクターは、TCP/IP プロトコルの TDI ドライバーの上端に接続します。

- 特別な構成、診断、および管理のために、ユーザーモードからカーネルモードドライバへのアクセスを提供するために、管理ツールが必要になることがあります。このツールを使用すると、問題のトラブルシューティングのためにトレースとログ記録を有効または無効にすることもできます。このツールは、さまざまなカスタムプライベート ioctl を使用して、カーネルドライバと通信します。このツールを使用すると、サービスコントロールマネージャー (SCM) にアクセスして、グローバルデータがユーザーモードネットワークプロバイダー DLL 用に安全に格納されている中間サービスを管理することもできます。

Windows Vista 以降では、Microsoft Windows のバージョンでは TDI はサポートされません。代わりに、[Windows フィルタリングプラットフォーム](#)または[Winsock カーネル](#)を使用してください。

ネットワーク リダイレクターの設計とパフォーマンス

2020/04/24 • [Edit Online](#)

ネットワークリダイレクターで高いパフォーマンスを実現するための2つの異なるパスがあります。何より、ネットワークに簡単に取得する最初のアプローチ応力します。2番目の方法が、;のネットワーク操作を最小限に抑えるための高度な技法を使用しようとしています。これらの操作の時間がかかり、多くの場合とを使用して、不足しているネットワークリソース(使用可能な帯域幅、たとえば)をセットアップします。開発者は、最初に成功したときに一般的に、2番目の方法を有効にします。

含まれるショートコードパスの特典を割り引くことはありません、中には、「実行する-ネットワーク」アプローチは、システムの操作がいくつかの基本的なボトルネックが発生しない場合にのみ、パフォーマンスの向上を提供します。たとえば、実際の転送速度が1.25 MB/秒に制限されていますが、10メガビットイーサネットネットワークがあるとします。これは、いくつかのディスクよりも高速、中には、メモリアクセスのスピードよりもかなり遅くなります。このため、頻繁にアクセスされるデータのメモリ内キャッシュを提供するシステムにはない1よりも高いパフォーマンス、多くの場合、提供されます。逆に、クライアント上のネットワークリダイレクターが好ましくない状況でキャッシュしようと主張している場合は、パフォーマンスが低下する可能性があります。たとえば、キャッシュマネージャーは、x86ベースのハードウェアのページの粒度に依存する場合があります完全4KBのブロックの読み取りし、書き込みに対して、アプリケーションによって開始される各32バイトの書き込みはこと。明らかに、最適なパフォーマンスを得るにネットワークリダイレクターを高度なキャッシュの管理の戦略を実装する必要がありますが、動的に決定する準備が均等にある必要がありますそれらを使用するかどうか。また、ネットワークリダイレクターがネットワークに対して実行する準備済みである必要がありますキャッシュが無効になっている場合。

CPUの飽和状態サーバーの場合、2番目の例が提供されます。リモートファイルシステムのパフォーマンスは、メジャーは、サーバーでの輻輳を軽減するために使われていない場合が大幅に低下可能性があります。たとえば、彩度の条件下で、サーバーには、アンロードのサーバーよりもパケットを削除するために大きい可能性があります。クライアントネットワークリダイレクターはペース、パケットが破棄されたパケットの確率を小さくするため、このような方法で送信することが推奨になります。さらに、リダイレクターには、単一のパケットにいくつかの操作を結合しようとするメジャーがかかる場合があります。勝利を飽和させられたサーバーの場合に、サーバー1パケットあたりの操作の詳細を取得するように、操作を組み合わせることです。

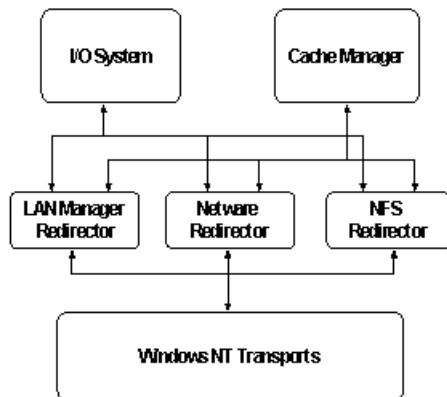
システムがあるトラフィックと可能性の高いネットワーク条件の種類に適したネットワークモードを使用する場合にも、パフォーマンスが低下します。エクスペリエンスは、データグラムベースモードを使用すると、小さなI/O要求のパフォーマンス向上を可能な相関関係を示します。この機能強化が全体的なかどうか、またはデータグラムの配信の確率が非常に高く、そのようなシナリオでのみ適用されるかどうかをオフにしないことをお勧めします。使用可能なデータの実装は確かに、接続指向モードだけでなく価値が「コネクションレス」モードです。理想は、両方のモードがあるネットワークプロトコルによってサポートされていると、使用可能なため、リダイレクターのトラフィックを実行する方法が動的に選択ネットワークリダイレクターになります。

最後に、賢明な意思決定を行うための操作中に取得した適切なデータがある場合、パフォーマンスを向上する可能性があります。これには、Windowsのさまざまな例があります。たとえば、キャッシュマネージャーは読み取りコマンドの短い履歴を保持し、トラックのキャッシュヒットとミス数は、先行読み取りのスケジュール設定の決定事項をガイドするために使用します。2番目の例には、「環境情報」の決定を行うための使用可能性があります。情報をファイル名、共有名、または、ユーザーや管理者パフォーマンスが大幅に向上するために使用できます。たとえば、応答は使用されているデータファイルをキャプチャするキャッシュサイズのパラメーターを変更するだけでパフォーマンスを向上させるためにチューニング場合があります。この「環境ナレッジ」のアクセスのパターンのパフォーマンスを向上させるため(ヒント)の動的な変更を提案する特定のシナリオが使用可能性がある可能性があります。

Windows NT におけるネットワーク リダイレクターの設計

2020/04/24 • [Edit Online](#)

ネットワークリダイレクターのカーネルモードドライバを実装するアーキテクチャは、時間の経過と共に変更されました。ネットワークリダイレクターの一般的なモデルは通常 Microsoft ネットワーク (LAN Manager のクライアント) クライアントのカーネルドライバを実装するアーキテクチャに基づいています。Windows NT 3.0 で導入された元のスキームを使用して、共有コンポーネントは使用されずが限られていました。このモデルは、元の rdr ドライバ モデルと呼ばれる多くの場合、(rdr がリダイレクターの省略形)。ネットワークリダイレクターを作成するプロセスを簡略化する、オペレーティングシステムからの特別なサポートが提供されません。各カーネルモードドライバには、すべてのネットワークリダイレクターに必要な関数を実装されています。その結果、各カーネルドライバには、大量 I/O マネージャー、キャッシュ マネージャー、およびメモリ マネージャーとのやり取りのコードには含まれます。Windows にインストールする各ネットワークリダイレクター (LAN Manager、NetWare、および例については、NFS) は、これらの関数自体のすべてを実装する必要があります。この設計モデルは、Windows NT 4.0 でのネットワークリダイレクター ドライバに使用されました。このアーキテクチャで、複数のリダイレクターのダイアグラムを次に示します。



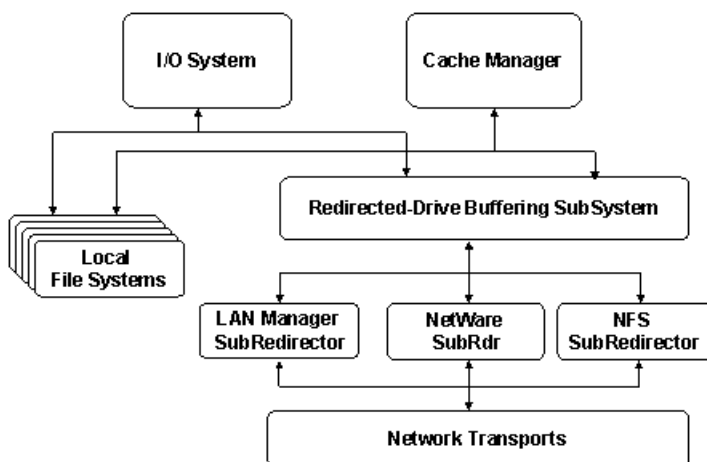
Windows 2000 でネットワーク リダイレクターの設計

2020/04/24 • [Edit Online](#)

ネットワークリダイレクターの設計で重要な課題とは、ユーザーが開始した操作から、操作の選択とタイミングに関しては、両方の低レベルのネットワーク操作に実行される、比較的複雑な変換です。Windows I/O システムで処理する、キャッシュ マネージャー、およびメモリ マネージャー、比較的複雑な作業です。これは、さまざまな場所、速度と信頼性が大きく異なるコンピューター ネットワークなどのリモート通信メカニズムに適した可能性のあるモードのバッファリングを検討する場合に特に当てはまります。ネットワークリダイレクターでこれらのバッファ処理操作の実装では、理想的には共有されるドライバーで再利用を関数にかなりの投資を表します。

Windows 2000 では、階層化またはミニポート ドライバーのアプローチに基づくネットワーク リダイレクター (ミニリダイレクターのアーキテクチャまたは rdr2 呼ばれます)、新しいドライバー モデルが導入されました。ではなくバッファリングと、I/O との対話に使用される複雑なコードを再実装するためにマネージャーとドライバーごとに、この大きなコード ブロックのキャッシュ マネージャーが引き出されているとすべての潜在的なネットワークリダイレクターが利用できます。共有の共通のバッファ処理コードは、リダイレクトされたドライブのバッファリング サブシステム (RDBSS) と呼ばれます。

このアーキテクチャをリダイレクターの複数のモデルは、以下に示します。



この RDBSS 設計の変更では、次の利点のいくつか用意されています。

- ネットワークリダイレクターの一般的なコード量が多いため、カーネル ドライバーを作成するプロセスを簡略化が必要なキャッシュ マネージャーでは、I/O システムで処理して、メモリ マネージャーが提供されます。
- 使用できるように他のネットワークリダイレクター ドライバー バッファリング アルゴリズムに基づいてパフォーマンスの向上、および Microsoft ネットワーク用に開発されたカーネル最適化の非常に長い時間。
- 開発して維持バッファリングのコードのコピーを 1 つだけ必要があるために、メンテナンスを簡略化します。以前のモデルでは、コピーを各リダイレクターの必要でした。
- ドライバー開発者は、ネットワークリダイレクターのそれぞれのアプリケーションや製品に固有である次の側面に集中できるように、ネットワークリダイレクターのネットワーク プロトコルに固有のコンポーネントの強力なカプセル化を提供します。
- この階層型アプローチに基づく分離を提供することでネットワークリダイレクター ドライバーのデバッグを簡略化します。

RDBSS モデルは、Windows 2000 で導入されました。この同じモデルは、Windows Server 2003 および Windows XP にも使

用されます。

リダイレクトされたドライブのバッファリング サブシステム

2020/04/24 • [Edit Online](#)

カーネル モードのファイル システム ドライバーとして、リダイレクトされたドライブのバッファリング サブシステム (RDBSS) が提供される *rdcss.sys*、オペレーティング システムと、スタティック ライブラリに含まれている *rdcsslib.lib* を Windows Driver Kit (WDK) に含まれています。スタティック ライブラリ内のコードの重複、*rdcss.sys* カーネル モード ドライバー。

元の設定がすべてのネットワーク ミニ リダイレクターと動的にリンクは、*rdcss.sys* カーネル ドライバー。動的にリンクされているドライバー *rdcss.sys* モノリシック以外のドライバーと呼ばれます。ただし、この機能が完全に実装されて、Microsoft SMB のネットワークが動的にミニ リダイレクター リンクのみ *rdcss.sys* モノリシック以外のドライバーとします。すべての他のネットワーク ミニ-リダイレクターなど、オペレーティング システムに含まれているその他の Microsoft リダイレクターがに対してリンク、*rdcsslib.lib* ライブラリと同じコードを静的に含めると、*rdcss.sys* カーネル ドライバー。同じ RDBSS ソース コードは、両方で使用されるときに、*rdcss.sys* ドライバーと *rdcsslib.lib* ライブラリ、モノリシック ネットワーク ミニ リダイレクター ドライバーのサイズが大きいでコードのほとんどから *rdcss.sys* ドライバーごとに含まれています。たとえば、Windows Server 2003 では、さまざまなドライバーのファイル サイズとしては、

DRIVER (KB)	KB	
<i>rdcss.sys</i>	156 KB	RDBSS のデバイス ドライバーは Microsoft SMB ミニリダイレクターでのみ使用されます。
<i>mrxdav.sys</i>	179 KB	リンクの多くの Microsoft WebDAV ネットワーク ミニ リダイレクター ドライバー <i>rdcsslib.lib</i> します。
<i>rdrrpc.sys</i>	185 KB	リンクの多くの Microsoft ターミナル サービス ネットワーク ミニ リダイレクター ドライバー <i>rdcsslib.lib</i> します。

ミニ リダイレクター、およびその他のカーネル システム、メモリ マネージャーではネットワーク、RDBSS 通信 マネージャーでは、I/O、キャッシュ マネージャー。RDBSS には、次の機能がサポートされています。

- RDBSS とミニ リダイレクター ドライバー間の通信に適切に定義されたメカニズムが定義されます。
- 複数のバッファリング モードがサポートされています。ライブの共有を有効にするために動的にバッファリング モードを変更するネットワーク ミニリダイレクターに役立つサポート ルーチンもあります。
- ネットワークのミニ リダイレクターは、名前付け、印刷、パイプ、メール スロット、および清掃 (一部のデータ構造を再利用) の UNC 名の大文字小文字を区別を含む、さまざまなオプションのサポートを指定できます。
- ネットワークのミニ リダイレクターは、名前空間、ファイルの情報、および接続が不要なネットワークの操作を減らすためにキャッシュをサポートします。

RDBSS は、通信を適切に定義されたメカニズムを使用します。RDBSS は、多数のミニ リダイレクター ドライバーとオプションを設定し、さまざまな操作を実行するには、その他のカーネル システムで呼び出すことができる関数をエクスポートします。

ミニ リダイレクターは、多数の RDBSS でミニ リダイレクター ドライバーとの通信に使用されるコールバック関数を実装します。ミニ リダイレクター ドライバーを最初に起動すると (でその `DriverEntry` ルーチン)、ドライバーは、これらのコールバック関数 (ディスパッチヘ

のポインターを持つ構造体のミニ リダイレクター ドライバーとパスを登録するため RDBSS 関数を呼び出すいくつかの構成データ値と共にテーブル)。

ミニ リダイレクター ドライバーに実装する必要があります、コールバック関数は、次の基本的なカテゴリに分類されます。

- ネットワークのミニ リダイレクター データ構造の管理にコールバックします。
- ファイル I/O 操作のコールバック。
 - 同期のファイル I/O ルーチン (作成、close、および例については、クリーンアップ)。
 - 非同期ファイル I/O ルーチン (読み取り、書き込み。FSTCL、IOCTL、および例については、ロック)。歴史的な理由から、ネットワークのミニ リダイレクターでこれらの同期のファイル I/O ルーチンは低い I/O 操作を多くの場合、呼び出されました。

ファイル I/O のコールバックは、低い I/O カテゴリに分類する関数を除く、同期する必要があります。低い I/O ルーチンでは、非同期的に動作でき、そのため、キャンセルをサポートする必要があります。

基本的なデータ構造体、RX がよく使用 RDBSS とミニ リダイレクター_情報を渡すためのコンテキスト。RX_コンテキストは、大量の情報をカプセル化し、現在 IRP へのポインターが含まれています。この RX の既定のサイズがある_コンテキスト データ構造体。この RX のサイズ_CONTEXT 構造は、リモート ファイル システムの要件に基づいて拡張して、ミニ リダイレクターがネットワークします。このために使用するサイズ拡張 RX_ミニ リダイレクターの RDBSS (指定されたデータ値の 1 つ) をまず登録時にコンテキストが定義されます。

RDBSS とミニ リダイレクターも広範に利用の次のデータ構造の抽象化します。

- サーバーを呼び出すコンテキスト (SRV_呼び出す)--各既知のファイル システムのサーバーに関連付けられているコンテキスト。
- ルートの net (NET_ルート)--共有のルートが、ユーザーが開いています。
- Net の仮想ルート (V_NET_ルート)--サーバー上の共有のビュー。複数のディメンションからビューを制約することができます。例として、ビューは関連付けログオン id には、共有上で実行できる操作を制限することができます。
- ファイル制御ブロック (FCB)--クライアントで開いている各一意のファイルに関連付けられている RDBSS データ構造体。
- ファイルオブジェクトの拡張子 (FOBX)--NT ファイルに RDBSS 拡張子_オブジェクト。
- サーバー側の開くコンテキスト (SRV_開きます)--リモート共有上のファイルを開いているハンドル。

RDBSS のバッファリング用の非常に柔軟なサポートしています。RDBSS は個別にバッファの読み取りおよび書き込み要求を設定できます。ファイルのサイズとファイルの時刻はキャッシュできます。明示的なバッファリング戦略が許可され、ファイル サーバー上の共有を開いたときにこれらを設定することができます。個々のファイルごとにバッファを設定することもできます。

RDBSS はアクセス履歴を使用して、不要なネットワーク要求の必要性が軽減しようとしています。たとえば、リモート ファイルのオープンの要求が失敗した場合、この情報は RDBSS によってキャッシュします。クライアントは、すぐに場合は、ファイル名の変更と同様のファイルを開く要求、RDBSS ファイル名の大文字小文字を区別するリモート ファイル システムの要求は失敗します。

概要

2020/04/24 • [Edit Online](#)

便宜的ロック (oplock) は、ファイルサーバクライアント (SMB および SMB2 プロトコルを使用するものなど) が、パフォーマンスを向上させるために一貫した方法で特定のファイルまたはストリームのバッファリング方法を動的に変更できるようにするメカニズムを提供します。ネットワークの使用。リモートファイル操作のネットワークパフォーマンスを向上させるために、クライアントはファイルデータをローカルにバッファリングできます。これにより、ネットワークパケットを送受信する必要がなくなります。たとえば、他のプロセスがデータにアクセスしていないことをクライアントが認識している場合、クライアントはリモートサーバ上のファイルに情報を書き込む必要がない可能性があります。同様に、クライアントは、他のプロセスがリモートファイルにデータを書き込んでいないことをクライアントが認識した場合に、リモートファイルから先行読み取りデータをバッファリングできます。アプリケーションとドライバーは、oplock を使用して、ファイルを使用する必要がある他のアプリケーションに影響を与えることなく、透過的にファイルにアクセスすることもできます。

NTFS などのファイルシステムは、ファイルごとに複数のデータストリームをサポートします。Oplock はストリームハンドルに付与されます。つまり、操作はファイルの指定されたオープンストリームに適用され、一般に、1つのストリームに対する操作は別のストリームの oplock には影響しません。これには例外があります。これは、以下に明示的に記載されています。FAT などの代替データストリームがサポートされていないファイルシステムでは、このドキュメントが "stream" を参照している場合は "file" と考えてください。

現在、8つの異なる種類の oplock があります。

- **レベル 2 (または共有) oplock** は、ストリームのリーダーが複数あり、ライターがないことを示します。これは、クライアントの読み取りキャッシュをサポートします。
- **レベル 1 (または排他) oplock** を使用すると、クライアントは排他アクセス用にストリームを開き、クライアントが任意のバッファリングを実行できるようになります。これは、クライアントの読み取りキャッシュと書き込みキャッシュをサポートします。
- **バッチ oplock (または排他的)** を使用すると、クライアントコンピューターのローカルアクセサがストリームを閉じた場合でも、クライアントはサーバでストリームを開いたままにすることができます。これは、バッチスクリプトの実行中などに、クライアントが同じファイルを繰り返し開いたり閉じたりする必要があるシナリオをサポートします。これは、クライアントの読み取りキャッシュ、書き込みキャッシュ、およびハンドルのキャッシュをサポートしています。
- **フィルター oplock (または排他的)** を使用すると、アプリケーションとファイルシステムフィルター (ミニフィルターを含む) を使用して、ストリームデータを開いて読み取ることができます。また、他のアプリケーション、クライアント、またはその両方が同じストリームにアクセスしようとしたときに "戻る" ことができます。これは、クライアントの読み取りキャッシュと書き込みキャッシュをサポートします。
- **読み取り(R) oplock (共有)** は、ストリームのリーダーが複数あり、ライターがないことを示します。これは、クライアントの読み取りキャッシュをサポートします。
- **読み取りハンドル(RH) oplock (shared)** は、ストリームに対して複数のリーダーが存在し、ライターがないこと、およびクライアントコンピューター上のローカルアクセサがストリームを閉じている場合でも、クライアントがサーバでストリームを開いたままにできることを示します。これは、クライアントの読み取りキャッシュとハンドルのキャッシュをサポートしています。
- **読み取り/書き込み(RW) oplock (排他)** を使用すると、クライアントは排他アクセス用にストリームを開き、クライアントが任意のバッファリングを実行できるようになります。これは、クライアントの読み取りキャッシュと書き込みキャッシュをサポートします。
- **読み取り/書き込みハンドル(RWH) oplock (排他)** を使用すると、クライアントコンピューターのローカルアクセサがストリームを閉じた場合でも、クライアントはサーバでストリームを開いたままにすることができます。これは、クライアントの読み取りキャッシュ、書き込みキャッシュ、およびハンドルのキャッシュをサポートしています。

レベル1、レベル2、および Batch の oplock が Windows NT 3.1 で実装されました。フィルター oplock が Windows 2000 で追加されました。Windows 7 では、R、RH、RW、RWH の oplock が追加されました。

一部の oplock は非常に似ています。特に、R は Level 2 に似ています。RW は Level 1 に似ており、RWH は Batch に似てい

ます。R、RH、RW、および RWH の各 oplock (総称して "Windows 7 の oplock" と呼ばれます) が Windows 7 に追加され、呼び出し元がキャッシュの意図を表現し、oplock の中断とアップグレード (つまり、1つのレベルからより高いレベルのキャッシュに対する oplock 状態たとえば、読み取り oplock を読み取り/書き込み oplock にアップグレードします。この柔軟性は、レベル2、レベル1、バッチ、およびフィルターの oplock によって実現することはできません (」は、総称して "レガシ oplock" と呼ばれます)。Windows 7 の oplock と従来の oplock の違いについては、このドキュメントで後ほど説明します。

Oplock パッケージのコア機能は、カーネルに実装されます (主に *Fsrtlxxx* レーチンを通じて)。ファイルシステムは、ファイルシステムに oplock 機能を実装するために、このパッケージを呼び出します。このドキュメントでは、NTFS ファイルシステムがカーネル oplock パッケージと相互運用する方法について説明します。他のファイルシステムも同様の方法で機能しますが、若干の違いがある可能性があります。

Oplock は、ストリームハンドルで許可されます。これは、指定されたストリームのオープンに対して oplock が付与されることを意味します。Windows 7 以降、ストリームハンドルは *oplock* キーに関連付けることができます。これは、同じクライアントキャッシュビューに属する複数のハンドルを識別するために使用される GUID 値です (このトピックの後半の「注」を参照してください)。ハンドルが作成されるときに、oplock キーを明示的に指定することができます (*IoCreateFileEx*)。ハンドルが作成されるときに oplock キーが明示的に指定されていない場合、そのキーが他のハンドルの他のキーと異なるように、ハンドルは一意の oplock キーとして扱われます。Oplock が許可されたハンドル以外のハンドルでファイル操作を受信し、oplock のハンドルに関連付けられている oplock キーが操作のハンドルに関連付けられているキーと異なる場合、その操作は、現在 oplock が許可されている場合、oplock は解除されます。Oplock は、互換性のない操作を実行しているプロセスまたはスレッドの場合でも中断されます。たとえば、排他的 oplock が付与されているストリームをプロセスが開き、同じプロセスが別の (またはない) oplock キーを使用して同じストリームを再び開く場合、排他的な oplock は直ちに解除されます。

Oplock キーはハンドルに存在し、ハンドルが作成されるとハンドルに "配置" されることに注意してください。Oplock が付与されていない場合でも、ハンドルを oplock キーに関連付けることができます。

、oplock キーは、ストリームハンドルが参照する [ファイル_オブジェクト](#) 構造に関連付けられていることに注意してください。この区別は、[DuplicateHandle](#) などを使用してハンドルが複製されるときに重要になります。重複する各ハンドルは、基になる同じ [ファイル_オブジェクト](#) 構造体を参照します。

Oplock の付与

2020/04/24 • [Edit Online](#)

Oplock は **FSCTLs** によって要求されます。次の一覧に、さまざまな oplock の種類の FSCTLs を示します (ユーザーモードのアプリケーションとカーネルモードドライバによって発行される可能性があります)。

- FSCTL_要求_OPLOCK_レベル_1
- FSCTL_要求_OPLOCK_レベル_2
- FSCTL_要求_バッチ_OPLOCK
- FSCTL_要求_フィルター_OPLOCK
- FSCTL_要求_OPLOCK

リスト内の最初の4つの FSCTLs は、レガシ oplock を要求するために使用されます。最後の FSCTL は、要求に対して Windows 7 の oplock を要求するために使用され、_OPLOCK_、要求の **Flags** メンバーに指定されている入力_フラグ_要求フラグ 要求の_の入力_バッファ構造 **DeviceIoControl** の *lpinbuffer* パラメーターとして渡された。_ 同様の方法で、 **Zwfscontrolfile** を使用して、カーネルモードから Windows 7 の oplock を要求できます。ファイルシステムミニフィルターでは、 **FltAllocateCallbackData** と **FltPerformAsynchronousIo** を使用して Windows 7 oplock を要求する必要があります。4つの Windows 7 oplock が必要であることを指定するために、1つまたは複数のフラグ OPLOCK_レベル_キャッシュ_読み取り、OPLOCK_レベル_キャッシュ_ハンドル、または_レベル_キャッシュ_WRITE は、要求の **RequestedOplockLevel** メンバーに設定されています。_OPLOCK_入力_バッファ構造体です。詳細については、「 **FSCTL_REQUEST_OPLOCK** 」を参照してください。

Oplock に対する要求が行われ、oplock が許可されると、ファイルシステムは状態 **PENDING** を返します (このため、oplock は同期 i/o には許可されません)。FSCTL IRP は、oplock が解除されるまで完了しません。Oplock を許可できない場合は、適切なエラーコードが返されます。最も一般的に返されるエラーコードは、ステータス **OPLOCK__無効な_パラメーター** (およびそれと同等のユーザーモードの類似) 状態です。

前述のように、フィルター oplock を使用すると、他のアプリケーション/クライアントが同じストリームにアクセスしようとしたときに、アプリケーションが "バックアウト" できます。このメカニズムにより、アプリケーションは、ストリームを開こうとしたときに、ストリームの他のアクセサが共有違反を受信しなくても、ストリームにアクセスできます。共有違反を回避するには、特別な3段階の手順を使用してフィルター oplock を要求します (FSCTL_REQUEST_フィルター_OPLOCK)。

1. ファイルに必要なアクセス_権を使用してファイルを開き_属性を読み取り、ファイルの共有モード_共有_読み取り | ファイル_共有_書き込み | ファイル_共有_削除します。
2. 手順 1. のハンドルに対してフィルター oplock を要求します。
3. 読み取りアクセス用にもう一度ファイルを開きます。

手順 1. で開かれたハンドルは、他のアプリケーションが共有の違反を受け取ることはありません。これは、データアクセス (ファイル_読み取り_データ) ではなく、属性アクセス (ファイル_読み取り_属性) に対してのみ開かれているためです。このハンドルは、フィルター oplock の要求に適していますが、データストリームでの実際の i/o の実行には適していません。手順 3. で開かれたハンドルを使用すると、oplock の所有者は、ストリームで i/o を実行できます。手順 2. で許可されている oplock を使用すると、oplock の所有者は、ストリームへのアクセスを試みる別のアプリケーションへの共有違反を回避できます。

NTFS ファイルシステムでは、ファイル_予約_OPFILTER 作成オプションフラグを使用して、この手順の最適化を行います。前の手順の手順 1. でこのフラグが指定されている場合は、ファイルシステムが手順2に失敗したと判断した場合に、ファイルシステムによってステータス **OPLOCK__失敗** します。手順 1. が成功した場合、create 要求に対してファイル_予約_OPFILTER が指定されていても、手順2が成功するという保証はありません。

次の表は、oplock を与えるために必要な条件を示しています。

=====	==
レベル 1 フィルター Batch (バッチ)	次のすべての条件に該当する場合にのみ許可されます。 ● 要求は、ファイルの指定されたストリームを対象としています。 ○ ディレクトリの場合、STATUS_INVALID_PARAMETER が返されます。 ● ストリームは非同期アクセス用に開かれています。 ○ 同期アクセスのために開かれた場合は STATUS_OPLOCK_NOT_GRANTED が返されます (同期 i/o 要求に対して oplock は付与されません)。 ● ファイルのストリームには TxF トランザクションがありません。 ○ それ以外の場合 STATUS_OPLOCK_NOT_GRANTED が返されます。 ● ストリームには他に開いているものはありません (同じスレッドでも)。 ○ それ以外の場合 STATUS_OPLOCK_NOT_GRANTED が返されます。 現在の oplock 状態があるのである場合は、次の点に注意してください。 ● No Oplock: 要求が許可されます。 ● レベル 2: 元のレベル2の要求が FILE_OPLOCK_BROKEN_TO_NONE で破損しています。要求された排他 oplock が付与されます。 ● レベル1、バッチ、フィルター、読み取り、読み取りハンドル、読み取り/書き込み、または読み取り/書き込みハンドル: STATUS_OPLOCK_NOT_GRANTED が返されます。

レベ	レ
レベル 2	次のすべての条件に該当する場合にのみ許可されます。 ● 要求は、ファイルの指定されたストリームを対象としています。 ○ ディレクトリの場合、STATUS_INVALID_PARAMETER が返されます。 ● ストリームは非同期アクセス用に開かれています。 ○ 同期アクセスのために開かれた場合、STATUS_OPLOCK_NOT_GRANTED が返されます。 ● ファイルに TxF トランザクションがありません。 ○ それ以外の場合 STATUS_OPLOCK_NOT_GRANTED が返されます。 ● ストリームに現在のバイト範囲ロックがありません。 ○ それ以外の場合 STATUS_OPLOCK_NOT_GRANTED が返されます。 ○ Windows 7 より前のオペレーティングシステムでは、バイト範囲ロックが最後に開かれてからストリームに存在していたかどうかを確認され、要求が失敗した場合はエラーが発生します。 現在の oplock 状態があるのである場合は、次の点に注意してください。 ● No Oplock: 要求が許可されます。 ● レベル2または読み取り: 要求が付与されます。同じストリームに対して複数のレベル 2/読み取り oplock を同時に許可することができます。同じハンドルに複数のレベル 2 (読み取り不可) oplock を存在させることもできます。 ○ 読み取り oplock が既に許可されているハンドルで読み取り oplock が要求されると、2 番目の読み取り oplock が付与される前に、最初の読み取り oplock の IRP が STATUS_OPLOCK_SWITCHED_TO_NEW_HANDLE で完了します。 ● レベル1、バッチ、フィルター、読み取りハンドル、読み取り/書き込み、読み取り/書き込みハンドル: STATUS_OPLOCK_NOT_GRANTED が返されます。

Read	次のすべての条件に該当する場合にのみ許可されます。 ● 要求は、ファイルの指定されたストリームを対象としています。 ○ ディレクトリの場合、STATUS_INVALID_PARAMETER が返されます。 ● ストリームは非同期アクセス用に開かれています。 ○ 同期アクセスのために開かれた場合、STATUS_OPLOCK_NOT_GRANTED が返されます。 ● ファイルに TxF トランザクションがありません。 ○ それ以外の場合 STATUS_OPLOCK_NOT_GRANTED が返されます。 ● ストリームに現在のバイト範囲ロックがありません。 ○ それ以外の場合 STATUS_OPLOCK_NOT_GRANTED が返されます。 現在の oplock 状態がである場合は、次の点に注意してください。 ● No Oplock: 要求が許可されます。 ● レベル2または読み取り: 要求が付与されます。同じストリームに対して複数のレベル 2/読み取り oplock を同時に許可することができます。 ○ また、既存の oplock に新しい要求と同じ oplock キーがある場合、その IRP は STATUS_OPLOCK_SWITCHED_TO_NEW_HANDLE で完了します。 ● 読み取りハンドルおよび既存の oplock には、新しい要求とは異なる oplock キーがあります。要求が許可されます。複数の読み取りと読み取りハンドルの oplock を同じストリームに共存させることができます (この表の後の注を参照してください)。 ○ それ以外の場合 (oplock キーは同じ) STATUS_OPLOCK_NOT_GRANTED が返されます。 ● レベル1、バッチ、フィルター、読み取り/書き込み、読み取り/書き込みハンドル: STATUS_OPLOCK_NOT_GRANTED が返されます。

読み取りハンドル	次のすべての条件に該当する場合にのみ許可されます。 ● 要求は、ファイルの指定されたストリームを対象としています。 ○ ディレクトリの場合、STATUS_INVALID_PARAMETER が返されます。 ● ストリームは非同期アクセス用に開かれています。 ○ 同期アクセスのために開かれた場合、STATUS_OPLOCK_NOT_GRANTED が返されます。 ● ファイルに TxF トランザクションがありません。 ○ それ以外の場合 STATUS_OPLOCK_NOT_GRANTED が返されます。 ● ストリームに現在のバイト範囲ロックがありません。 ○ それ以外の場合 STATUS_OPLOCK_NOT_GRANTED が返されます。 現在の oplock 状態がである場合は、次の点に注意してください。 ● No Oplock: 要求が許可されます。 ● Read: 要求が付与されます。 ○ 既存の読み取り oplock に新しい要求と同じ oplock キーがある場合、その IRP は STATUS_OPLOCK_SWITCHED_TO_NEW_HANDLE で完了します。これは、oplock が Read から Read Handle にアップグレードされることを意味します。 ○ 新しい要求と同じ oplock キーを持たない既存の読み取り oplock は、変更されずに残ります。 ● レベル2、レベル1、バッチ、フィルター、読み取り/書き込み、読み取り/書き込みハンドル: STATUS_OPLOCK_NOT_GRANTED が返されます。
読み取り/書き込み	次のすべての条件に該当する場合にのみ許可されます。 ● 要求は、ファイルの指定されたストリームを対象としています。 ○ ディレクトリの場合、STATUS_INVALID_PARAMETER が返されます。 ● ストリームは非同期アクセス用に開かれています。 ○ 同期アクセスのために開かれた場合、STATUS_OPLOCK_NOT_GRANTED が返されます。 ● ファイルに TxF トランザクションがありません。 ○ それ以外の場合 STATUS_OPLOCK_NOT_GRANTED が返されます。 ● (同じスレッドでも) ストリームに他のオープンがある場合は、同じ oplock キーを持つ必要があります。 ○ それ以外の場合 STATUS_OPLOCK_NOT_GRANTED が返されます。 現在の oplock 状態がである場合は、次の点に注意してください。 ● No Oplock: 要求が許可されます。 ● 読み取りまたは読み取り/書き込み、既存の oplock には、要求と同じ oplock キーがあります。既存の oplock の IRP が STATUS_OPLOCK_SWITCHED_TO_NEW_HANDLE で完了し、要求が付与されます。 ○ それ以外の場合 STATUS_OPLOCK_NOT_GRANTED が返されます。 ● レベル2、レベル1、バッチ、フィルター、読み取りハンドル、読み取り/書き込みハンドル: STATUS_OPLOCK_NOT_GRANTED が返されます。

読み取り/書き込み-ハンドル	次のすべてに該当する場合にのみ許可されます。 - 要求は、ファイルの指定されたストリームを対象としています。 - ディレクトリの場合、STATUS_INVALID_PARAMETER が返されます。 - ストリームは非同期アクセス用に開かれています。 - 同期アクセスのために開かれた場合、STATUS_OPLOCK_NOT_GRANTED が返されます。 - ファイルに TxF トランザクションがありません。 - それ以外の場合 STATUS_OPLOCK_NOT_GRANTED が返されます。 (同じスレッドでも) ストリームに他のオープン要求がある場合は、同じ oplock キーを持つ必要があります。 - それ以外の場合 STATUS_OPLOCK_NOT_GRANTED が返されます。 現在の oplock 状態がである場合は、次の点に注意してください。 - No Oplock: 要求が許可されます。 - 読み取り、読み取りハンドル、読み取り/書き込み、または読み取り/書き込みハンドル。既存の oplock には、要求と同じ oplock キーがあります。既存の oplock の IRP は STATUS_OPLOCK_SWITCHED_TO_NEW_HANDLE で完了し、要求は許可されます。 - それ以外の場合 STATUS_OPLOCK_NOT_GRANTED が返されます。 - レベル2、レベル1、バッチ、フィルター: STATUS_OPLOCK_NOT_GRANTED が返されます。

注 読み取りとレベル2の oplock を同じストリームに共存させることができ、読み取りと読み取りハンドルの oplock を共存させることはできますが、レベル2と読み取りハンドルの oplock は共存できない可能性があります。

Oplock の破損

2020/04/24 • [Edit Online](#)

Oplock が付与されると、その oplock の所有者は、要求された oplock の種類に基づいて、ストリームにアクセスできるようになります。受信した操作が現在の oplock と互換性がない場合、oplock は解除されます。

Oplock が付与されると、要求元の IRP は保留されます。Oplock が解除されると、保留中の oplock 要求の IRP は状態 `_SUCCESS` で完了します。レベル1、バッチ、およびフィルターについては、`IoStatus` を適用します。IRP の情報メンバーは、oplock が中断されているレベルを示すように設定されます。これらのレベルは次のとおりです。

- `FILE_OPLOCK_BREAK_IN_PROGRESS` - `None`-oplock が解除され、ストリームに現在の oplock がありません。Oplock は "None に破損しています" と言います。
- `FILE_OPLOCK_LEVEL_2` に破損しています。現在の oplock (レベル1またはバッチ) はレベル2の oplock に変換されました。フィルターの oplock がレベル2になることはないことに注意してください。これは常に [なし] になります。

読み取りハンドル、読み取り/書き込み、および読み取り/書き込みハンドルの oplock は、oplock を解除するレベルを、0個以上のフラグ `OPLOCK` の組み合わせとして記述されます。レベル_キャッシュ_読み取り、`OPLOCK_LEVEL_CACHE_HANDLE`、または `OPLOCK_LEVEL_CACHE_REQUEST` 要求の `DeviceIoControl` の `IoPseudoBuffer` パラメーターとして渡されたバッファー構造を書き込みます。同様の方法で、`FltFsControlFile` と `zwFsControlFile` を使用して、カーネルモードから Windows 7 の oplock を要求できます。詳細については、「`FSCTL_REQUEST_OPLOCK`」を参照してください。

レベル1、バッチ、フィルター、読み取り/書き込み、読み取り/書き込みハンドル、または特定の状況下で (注を参照)、読み取りハンドルの oplock を解除すると、oplock パッケージによって保留中の oplock 要求 IRP が完了し、oplock の原因となった操作が実行されます。保留 (操作が同期ハンドルに対して実行される場合、または同期ハンドルに対して実行される場合は、`IRP_MJ_CREATE` (常に同期される) であることに注意してください。i/o マネージャーによって操作がブロックされます。返される状態 `_PENDING`) は、oplock の所有者からの受信確認を待機して、処理が完了したことを oplock パッケージに伝えます。また、保留中の操作を続行することも安全です。この遅延の目的は、oplock の所有者が、現在の操作を続行する前にストリームを一貫性のある状態に戻ることができるようにすることです。タイムアウトがないため、システムは無期限に受信確認を受信します。そのため、oplock の所有者は、適切なタイミングで中断を確認する必要があります。保留中の操作の IRP は、キャンセル可能な状態に設定されます。待機を実行しているアプリケーションまたはドライバが終了した場合、or the IRP は状態_取り消された状態で直ちに完了します。

`IRP_MJ_CREATE` IRP では、oplock 解除確認の一部としてブロックされないように `_OPLOCKED create` オプションを指定すると、`FILE` 完了するように指定できます。このオプションは、oplock 解除の受信確認が受信されるまで、create IRP をブロックしないように oplock パッケージに指示します。代わりに、作成を続行できます。作成が成功した場合、oplock ブレークが発生すると、ステータス `_SUCCESS` ではなく、`_PENDING` の進行状況で `_OPLOCKED` が返されます。`_OPLOCKED` フラグがデッドロックを回避するために通常使用される場合は、`FILE` 完了します。たとえば、クライアントがストリームで oplock を所有していて、同じクライアントが同じストリームを開いた場合、クライアントは、それ自体が oplock の中断を確認するのを待機してブロックします。このシナリオでは、`_OPLOCKED` フラグによってデッドロックが回避される場合、`FILE` を使用します。

NTFS ファイルシステムは、共有違反を確認する前に、バッチとフィルターの oplock に対して oplock の解除を開始するため、指定された `FILE` が完了したとしても、`_OPLOCKED` が状態_共有に失敗する可能性があり、違反が発生しても、バッチまたはフィルターの oplock は中断されます。この場合は、`IO_STATUS_BLOCK` 構造体の情報メンバーが `FILE_opbatch_BREAK` 実行中に設定され、呼び出し元がこのケースを検出できるようになります。

読み取りハンドルと読み取り/書き込みハンドルの oplock の場合、共有違反を検出して検出すると、oplock の解除が開始されます。これにより、これらの oplock を解除してハンドルを終了し、ユーザーに共有違反を返すことができないようにする機会が与えられます。また、oplock によってキャッシュされたハンドルが新しい create と競合しない場合に、無条件で oplock が解除されるのを回避できます。

レベル2、読み取り、および、特定の状況下 (注を参照)、読み取りハンドル oplock 解除、システムは受信確認を待機しません。

これは、他のクライアントからのアクセスを許可する前に、ファイルに復元する必要があるストリームには、キャッシュされた状態がないためです。

現在の oplock 状態をチェックして oplock を解除する必要があるかどうかを判断する、特定のファイルシステム操作があります。次のセクションでは、各操作の一覧を示し、oplock 解除のトリガー、oplock が中断されるレベルを決定する方法、および中断の確認が必要かどうかについて説明します。

- [IRP_MJ_CREATE](#)
- [IRP_MJ_READ](#)
- [IRP_MJ_WRITE](#)
- [IRP_MJ_CLEANUP](#)
- [IRP_MJ_LOCK_CONTROL](#)
- [IRP_MJ_SET_INFORMATION](#)
- [IRP_MJ_FILE_SYSTEM_CONTROL](#)

Windows 7 の oplock を解除するには、要求のFlagsメンバーで_出力_フラグ_ACK_必要なフラグが設定されている場合は、受信確認が必要_要求_OPLOCK_出力_バッファー [DeviceIoControl](#)(*Lpoutbuffer*)、[fltfscontrolfile](#)(*Outbuffer*)、または [zwfscontrolfile](#)(*outbuffer*) の出力パラメーターとして渡された構造体。詳細については、「[FSCTL_REQUEST_OPLOCK](#)」を参照してください。

ここに示されている操作ごとのトピックでは、読み取りハンドルの oplock が解除されたときに、oplock を壊す操作が保留になった場合の詳細について説明します。たとえば、[IRP_MJ_CREATE](#) トピックには、関連付けられている読み取りハンドルの詳細が含まれています。

Irps_create 用操作の Oplock の状態を確認

2020/04/24 • [Edit Online](#)

次は、ファイルの既存のストリームが開いているときにのみ適用されます (つまり、新しく作成されたストリームではできませんの既存の各 oplock に)。

注 FILE_RESERVE_OPFILTER が無い限り、必要なアクセスは何も FILE_READ_ATTRIBUTES、FILE_WRITE_ATTRIBUTES、または同期以外が含まれる場合は、任意の oplock の Irps_create 用を処理するとき、oplock は中断されません指定します。作成が成功した場合、oplock の結果 FILE_RESERVE_OPFILTER を常に指定します。簡潔にするため、わかりやすくするための次の表は、上記に oplock のすべてに適用するため省略します。

状態	説明
レベル 1	Irps_create 用に壊れている場合。 - 関連付けられた open が発生している FILE_OBJECT oplock キーは、oplock を所有している FILE_OBJECT に関連付けられている oplock キーと異なる。 Oplock が破損している場合。 [なし] に中断■ - FILE_RESERVE_OPFILTER フラグを設定します。 OR - 次のいずれかは、値が指定された処理を作成します。 - FILE_SUPERSEDE - FILE_OVERWRITE - FILE_OVERWRITE_IF ■ - レベル 2 にブレイクします。 - 操作を続行する前に、受信確認を受信する必要があります。
レベル 2	Irps_create 用に壊れている場合。 - 関連付けられた open が発生している FILE_OBJECT oplock キーは、oplock を所有している FILE_OBJECT に関連付けられている oplock キーと異なる。 ■: - FILE_RESERVE_OPFILTER フラグを設定します。 OR - 次のいずれかは、値が指定された処理を作成します。 - FILE_SUPERSEDE - FILE_OVERWRITE - FILE_OVERWRITE_IF

	Oplock が破損している場合。 • [なし] に中断します。 • 受信確認は必要ありません、すぐに、操作を続行します。
Batch (バッチ)	lrp_mj_create 用に壊れている場合。 • 関連付けられた open が発生している FILE_OBJECT oplock キーは、oplock を所有している FILE_OBJECT に関連付けられている oplock キーと異なる。
	Oplock が破損している場合。 • [なし] に中断■: ◦ FILE_RESERVE_OPFILTER フラグが設定されます。 - OR ◦ 次のいずれかは、値が指定された処理を作成します。 ◦ FILE_SUPERSEDE ◦ FILE_OVERWRITE ◦ FILE_OVERWRITE_IF ■ ◦ レベル 2 にブレイクします。 • 操作を続行する前に、受信確認を受信する必要があります。
フィルター	lrp_mj_create 用に壊れている場合。 • 関連付けられた open が発生している FILE_OBJECT oplock キーは、oplock を所有している FILE_OBJECT に関連付けられている oplock キーと異なる。 • ■: ◦ FILE_SHARE_READ アクセス用に開かれませんが、ストリームで目的の「書き込み」アクセスが要求されました。「書き込み」アクセスが以外の任意の属性として定義されていることを確認してください。 ◦ FILE_READ_ATTRIBUTES ◦ FILE_WRITE_ATTRIBUTES ◦ FILE_READ_DATA ◦ FILE_READ_EA ◦ FILE_EXECUTE ◦ SYNCHRONIZE ◦ READ_CONTROL
	Oplock が破損している場合。 • [なし] に中断します。 • 操作を続行する前に、受信確認を受信する必要があります。

Read	Irp_mj_create 用に壊れている場合。 • 関連付けられた open が発生している FILE_OBJECT oplock キーは、oplock を所有している FILE_OBJECT に関連付けられている oplock キーと異なる。 • ■: ◦ FILE_RESERVE_OPFILTER フラグを設定します。 OR ◦ 次のいずれかは、値が指定された処理を作成します。 ◦ FILE_SUPERSEDE ◦ FILE_OVERWRITE ◦ FILE_OVERWRITE_IF
読み取りハンドル	Irp_mj_create 用に壊れている場合。 • 既存の現在開いている競合は、共有違反が発生するように開きます。 OR • FILE_RESERVE_OPFILTER フラグが設定されます。 OR • 次のいずれかは、値が指定された処理を作成します。 ◦ FILE_SUPERSEDE ◦ FILE_OVERWRITE ◦ FILE_OVERWRITE_IF (の上記の 3 つの条件のいずれか) • 関連付けられた open が発生している FILE_OBJECT oplock キーは、oplock を所有している FILE_OBJECT に関連付けられている oplock キーと異なる。

	Oplock が破損している場合。 • [なし] に中断■ ◦ FILE_RESERVE_OPFILTER フラグが設定されま す。 OR ◦ 次のいずれかは、値が指定された処理を作成します。 ◦ FILE_SUPERSEDE ◦ FILE_OVERWRITE ◦ FILE_OVERWRITE_IF ■ ◦ 読み取り専用に中断します。 • 共有違反が発生するように、既存の現在開いている競合が 開くために oplock が破損した場合は、操作を続行する前に 受信確認が受信する必要があります。 • 中断の受信確認は必須ですが、その他の何らかの理由で、 oplock が破損した場合は、すぐに (受信確認を待機すること がなくなと) の操作が続行されます。
読み取り/書き込み	Irp_mj_create 用に壊れている場合。 • 関連付けられた open が発生している FILE_OBJECT oplock キーは、oplock を所有している FILE_OBJECT に 関連付けられている oplock キーと異なる。
	Oplock が破損している場合。 • [なし] に中断■ ◦ FILE_RESERVE_OPFILTER フラグが設定されま す。 OR ◦ 次のいずれかは、値が指定された処理を作成します。 ◦ FILE_SUPERSEDE ◦ FILE_OVERWRITE ◦ FILE_OVERWRITE_IF ■ ◦ 読み取り専用に中断します。 • 操作を続行する前に、受信確認を受信する必要があり ます。
書き込みハンドルの読み取り	Irp_mj_create 用に壊れている場合。 • 関連付けられた open が発生している FILE_OBJECT oplock キーは、oplock を所有している FILE_OBJECT に 関連付けられている oplock キーと異なる。

Oplock が破損している場合。

- [なし] に中断■
 - FILE_RESERVE_OPFILTER フラグが設定されません。
- OR
 - 次のいずれかは、値が指定された処理を作成します。
 - FILE_SUPERSEDE
 - FILE_OVERWRITE
 - FILE_OVERWRITE_IF
- - 共有違反が発生するように、現在のオープンが既存のオープンと競合する場合は、読み取り/書き込みを中断します。それ以外の場合、読み取りハンドルを中断します。
- 操作を続行する前に、受信確認を受信する必要があります。

ファイル システムは、バッチとフィルターの各 oplock (なく oplock パッケージ自体) の追加チェックを実行、ファイル システム oplock 中断処理を実行するための oplock のパッケージを確認するかどうかに影響を与える irp_mj_create 用の操作を処理するときに行います。これは、同じファイル (つまり、次の条件の一覧の最後の 2 つのリスト項目) の場合は、他のデータ ストリームの各 oplock を 1 つのデータ ストリームでの操作が影響する場合。1 つ以上の次の条件が満たされた場合、ファイル システムは、oplock 中断処理を実行するための oplock のパッケージに要求を送信します。

- これは、ネットワークのクエリを開くと、中断を要求 [KTM](#) トランザクションがある場合。それ以外の場合、区切りをネットワークのクエリを開く要求しないをしないでください。
- 優先的、上書きされたり OVERWRITE_IF 操作は、代替データ ストリームに対して FILE_SHARE_DELETE が指定されていないと、プライマリデータ ストリームにバッチまたはフィルター oplock が、プライマリデータ ストリームのバッチまたはフィルターの oplock の中断を要求します。
- プライマリデータ ストリームで優先的、上書きされたり OVERWRITE_IF 操作が実行され、削除のアクセス許可が要求されている任意の代替データ ストリームのバッチまたはフィルターの各 oplock がある、代替のすべてのデータのバッチまたはフィルターの各 oplock の中断を要求します。設定されているストリーム。

Oplock 中断処理を実行するための oplock のパッケージを確認するファイル システムが決定したら、上記の表にレイアウトされる規則が適用されます。

バッチとフィルターの oplock のチェックは、共有のアクセス チェックが行われる前に発生します。つまり、オープン要求は最終的には共有違反のため失敗した場合でも、バッチまたはフィルター oplock が壊れています。

IRP_MJ_READ 操作の Oplock 状態を確認しています

2020/04/24 • [Edit Online](#)

ストリームを読み取るときには、次の **oplock ブレーク** 条件が適用されます。TxF のトランザクションリーダーが読み取りを実行する場合、トランザクションリーダーがライターを除外するため、このチェックは行われません (つまり、oplock を保持するライターはまったく存在できません)。

レベル2、フィルター、読み取り、および読み取りハンドルの要求の種類の条件

- Oplock が破損していません。
- 確認は必要ありません。操作は直ちに続行されます。

レベル1、バッチ、読み取り/書き込み、および読み取り/書き込みハンドルの要求の種類の条件

- Oplock を所有する FILE_OBJECT のキーとは異なる oplock キーを持つ FILE_OBJECT で読み取り操作が発生した場合に IRP_MJ_READ を中断します。Oplock が解除された場合は、次のようになります。
 - レベル1とバッチ要求がレベル2に分割されます。
 - 読み取り/書き込み要求が読み取りに対して中断します。
 - 読み取り/書き込みハンドル要求が読み取りハンドルに分割されます。
- 操作を続行するには、受信確認を受信する必要があります。

IRP_MJ_WRITE 操作の Oplock 状態を確認しています

2020/04/24 • [Edit Online](#)

ストリームが書き込まれていて、書き込みがページング i/o ではない場合、次の **oplock ブレーク** 条件が適用されます。

レベル2の要求の種類の条件:

- 常に [なし] になります。
- 確認は必要ありません。操作は直ちに続行されます。

他のすべての要求の種類の条件:

- Oplock を所有する FILE_OBJECT のキーとは異なる oplock キーを持つ FILE_OBJECT で書き込み操作が行われたときに、IRP_MJ_WRITE を中断します。Oplock が解除されている場合は、None に中断します。
- 確認の要件は次のとおりです。
 - 読み取り要求: 確認は必要ありません。操作は直ちに続行されます。
 - 読み取りハンドル要求: 中断の確認が必要ですが、操作はすぐに続行されます (たとえば、受信確認を待たずに)。
 - レベル1、バッチ、フィルター、読み取り/書き込み、および読み取り/書き込みハンドル要求: 操作を続行する前に受信確認を受信する必要があります。

IRP_MJ_CLEANUP 操作の Oplock 状態を確認しています

2020/04/24 • [Edit Online](#)

次のoplock [ブレーク](#)条件は、ストリームが閉じられている場合にのみ適用されます。

レベル2および読み取り要求の種類の条件

- 常に [なし] になります。同じストリームに対する他のレベル2または読み取り oplock は影響を受けないことに注意してください。この FILE_OBJECT に関連付けられているレベル2または読み取り oplock のみが壊れています。
- 確認は必要ありません。操作は直ちに続行されます。

レベル1、バッチ、フィルター、読み取りハンドル、読み取り/書き込み、および読み取り/書き込み要求の種類の条件

- 常に [なし] になります。
- 確認は必要ありません。操作は直ちに続行されます。保留中の中断要求からの受信確認を待機している i/o 操作 (Irp) は、直ちに完了することに注意してください。

IRP_MJ_LOCK_CONTROL 操作の Oplock 状態を確認しています

2020/04/24 • [Edit Online](#)

次のoplock ブレーク条件は、指定されたストリームのすべてのバイト範囲ロック操作に適用されます。

レベル2の要求の種類の条件

- 常に [なし] になります。
- 確認は必要ありません。操作は直ちに続行されます。

フィルター要求の種類の条件

- Oplock が破損していません。
- 確認は必要ありません。操作は直ちに続行されます。

レベル1、バッチ、読み取り、読み取りハンドル、読み取り/書き込み、および読み取り/書き込みの各要求の種類の条件

- Oplock を所有する FILE_OBJECT のキーとは異なる oplock キーを持つ FILE_OBJECT でロック操作が発生したときに、IRP_MJ_LOCK_CONTROL を中断します。Oplock が解除されている場合は、None に中断します。
- 確認の要件は次のとおりです。
 - 読み取り要求: 確認は必要ありません。操作は直ちに続行されます。
 - 読み取りハンドルと読み取り/書き込みハンドルの要求: 中断の確認が必要ですが、操作はすぐに続行されます (たとえば、受信確認を待たずに)。
 - レベル1、バッチ、および読み取り/書き込み要求: 操作を続行する前に、受信確認を受信する必要があります。

IRP_MJ_SET_INFORMATION 操作の Oplock 状態を確認しています

2020/04/24 • [Edit Online](#)

次の IRP_MJ_SET_INFORMATION 操作の oplock の状態を確認します。

- FileEndOfFileInformation
- Fileの情報
- FileValidDataLengthInformation
- FileRenameInformation
- FileShortNameInformation
- FileLinkInformation
- FileDispositionInformation

FileEndOfFileInformation、FileValidDataLengthInformation、および操作の oplock 状態を確認しています

この情報は、ファイルまたはストリームに対して次の操作が実行されている場合に適用されます。

- 呼び出し元がストリームの論理サイズを変更しようとしています。キャッシュマネージャーのレイジーライタースレッドが新しいファイルの終わりを設定しようとする、oplock チェックは行われないことに注意してください。これは、実際の書き込み要求を受信したときにチェックが行われたためです。
- 呼び出し元は、割り当てられたストリームのサイズを変更しようとしています。

レベル2の要求の種類の条件

- 常に [なし] になります。
- 確認は必要ありません。操作は直ちに続行されます。

その他すべての要求の種類の条件

- Oplock を所有する FILE_OBJECT のキーとは異なる oplock キーを持つ FILE_OBJECT で操作が行われた場合、IRP_MJ_SET_INFORMATION (FileEndOfFileInformation、FileValidDataLengthInformation、および) で中断します。Oplock が解除されている場合は、None に中断します。
- 確認の要件は次のとおりです。
 - 読み取り要求: 確認は必要ありません。操作は直ちに続行されます。
 - 読み取りハンドル要求: 中断の確認が必要ですが、操作はすぐに (つまり、受信確認を待たずに) 続行されます。
 - レベル1、バッチ、フィルター、読み取り/書き込み、および読み取り/書き込みハンドル要求: 操作を続行する前に受信確認を受信する必要があります。

FileRenameInformation、FileShortNameInformation、および FileLinkInformation 操作の oplock 状態を確認しています

この情報は、ファイルまたはストリームに対して次の操作が実行されている場合に適用されます。

- ファイルまたはストリームの名前が変更されています。
- ファイルに短い名前が設定されています。

- ファイルのハードリンクが作成されています。これは、新しいハードリンクが別のファイルへの既存のリンクを置き換えるときに、置き換えられるリンクに oplock が存在する場合に、oplock 状態に影響します。
- Oplock が存在するストリームの先祖ディレクトリの名前が変更されたか、または先祖ディレクトリの短い名前が設定されています。

レベル1、レベル2、読み取り、および読み取り/書き込み操作の条件

- Oplock が破損していません。
- 確認は必要ありません。操作は直ちに続行されます。

バッチ、フィルター、読み取りハンドル、および読み取り/書き込みの各操作の条件

- Oplock を所有する FILE_OBJECT のキーとは異なる oplock キーを持つ FILE_OBJECT で操作が発生した場合に、IRP_MJ_SET_INFORMATION (FileRenameInformation、FileShortNameInformation、および FileLinkInformation) で中断します。Oplock が解除された場合は、次のようになります。
 - バッチおよびフィルター要求が None に分割されます。
 - 読み取りハンドル要求が読み取りを中断します。
 - 読み取り/書き込みハンドルの要求は、読み取り/書き込みを中断します。
- 操作を続行するには、受信確認を受信する必要があります。

FileDispositionInformation 操作の oplock 状態を確認しています

この情報は、呼び出し元がファイルを削除しようとしたときに適用されます。

- Oplock を所有している FILE_OBJECT のキー [FILE_DISPOSITION_INFORMATION](#)とは異なる oplock キーを持つ FILE_OBJECT で操作が発生した場合は、IRP_MJ_SET_INFORMATION (FileDispositionInformation) を中断します。DeleteFile は * * TRUE * * * * です。Oplock が解除された場合は、次のようになります。
 - 読み取りハンドル要求が読み取りを中断します。
 - 読み取り/書き込みハンドルの要求は、読み取り/書き込みを中断します。
- 操作を続行するには、受信確認を受信する必要があります。

IRP_MJ_FILE_SYSTEM_CONTROL の Oplock 状態を確認しています

2020/04/24 • [Edit Online](#)

次の IRP_MJ_FILE_SYSTEM_CONTROL 操作の oplock の状態を確認します。

- **FSCTL_SET_ZERO_DATA**

この情報は、呼び出し元が指定されたストリームの現在の内容をゼロにする必要がある場合に適用されます。

レベル2の要求の種類の条件:

- 常に [なし] になります。
- 確認は必要ありません。操作は直ちに続行されます。

他のすべての要求の種類の条件:

- Oplock を所有する FILE_OBJECT のキーとは異なる oplock キーを持つ FILE_OBJECT で操作が行われた場合に、IRP_MJ_FILE_SYSTEM_CONTROL (FSCTL_SET_ZERO_DATA) で中断します。Oplock が解除されている場合は、None に中断します。
- 確認の要件は次のとおりです。
 - 読み取り要求: 確認は必要ありません。操作は直ちに続行されます。
 - 読み取りハンドル要求: 中断の確認が必要ですが、操作はすぐに続行されます (たとえば、受信確認を待たずに)。
 - レベル1、バッチ、フィルター、読み取り/書き込み、および読み取り/書き込みハンドル要求: 操作を続行する前に受信確認を受信する必要があります。

Oplock の破損の確認

2020/04/24 • [Edit Online](#)

Oplock の所有者は、さまざまな種類の受信確認を返すことができます。[Grant 要求](#)と同様に、これらの受信確認はファイルシステム制御コード (つまりFSCTLs) として送信されます。それらは以下のとおりです。

- FSCTL_OPLOCK_中断_確認
 - この FSCTL は、oplock 所有者がストリーム同期を完了し、oplock が解除されたレベル (レベル2または None のいずれか) を受け入れることを示します。
- FSCTL_OPLOCK_中断_ACK_NO_2
 - この FSCTL は、oplock 所有者がストリーム同期を完了したが、レベル2の oplock を必要としないことを示します。代わりに、oplock は None に分割する必要があります (つまり、oplock は完全に放棄)。
- FSCTL_OPBATCH_ACK_終了_保留中
 - レベル1の oplock の場合、この FSCTL は oplock の所有者がストリームの同期を完了し、oplock が完全に保たままされていることを示します (この確認によってレベル2の oplock が生成されることはありません)。
 - パッチまたはフィルターの oplock の場合、この FSCTL は oplock 所有者が oplock が付与されたストリームハンドルを閉じることを示します。Oplock 解除の確認を待機している操作がブロックされた場合は、oplock 所有者のハンドルが閉じられるまで待機し続けます。
- FSCTL_要求_OPLOCK
 - 要求_OPLOCK を指定することによって、要求のFlagsメンバーで_入力_フラグ_ACKを指定することにより、[DeviceIoControl](#)のlpinbufferパラメーターとして渡される入力_バッファー構造体_この FSCTL を使用して、Windows 7 の oplock が解除されたことを確認します。_ 受信確認が必要なのは、要求の Flags メンバーで_出力_フラグ_ACK_必要フラグが設定されている場合にのみ、要求のフラグメンバー_OPLOCK_出力_バッファー構造として渡される [_DeviceIoControl](#)のlpoutbufferパラメーター。同様の方法で、[FltFscontrolfile](#)と[zwfscontrolfile](#)を使用して、カーネルモードからの Windows 7 の oplock を確認することができます。詳細については、「[FSCTL_REQUEST_OPLOCK](#)」を参照してください。

関連する FSCTL コードは FSCTL_OPLOCK_中断_通知します。このコードは、指定されたストリームで oplock の解除が完了したときに呼び出し元が通知を受け取る場合に使用されます。この呼び出しはブロックされる可能性があります。FSCTL_OPLOCK_中断_通知呼び出しが成功_ステータスを返した場合は、次のいずれかが返されます。

- Oplock は付与されません。
- 呼び出し時に oplock 解除が実行されていませんでした。
- 進行中の oplock の中断がすべて完了しました。

受信確認が想定されていないときに受信確認を送信すると、エラーが発生し、受信確認 FSCTL IRP が失敗し、状態_無効_OPLOCK_プロトコルになります。

Oplock 解除が要求されたファイルのハンドルを閉じると、暗黙的に中断が認識されます。Oplock が共有違反に対して中断された場合、oplock 所有者は oplock 解除を確認し、ファイルの他のユーザーの共有違反を防ぐファイルハンドルを閉じることができます。

RDBSS ドライバーとライブラリ

2020/04/24 • [Edit Online](#)

2 つの形式で、リダイレクトされたドライブのバッファリング サブシステム (RDBSS) に実装されます。

- ファイル システム ドライバー (*rdbss.sys*) オペレーティング システムに付属します。
- スタティック ライブラリ (*rdbsslib.lib*) と Windows Driver Kit (WDK) を指定します。

Rdbss.sys 任意の非モノリシック ネットワーク ミニ-リダイレクターがシステムに登録されている場合、ドライバが自動的に読み込まれます。サーバー メッセージ ブロック (SMB) リダイレクター (*mrxsmb.sys*) は現在非モノリシック ネットワーク ミニ リダイレクター ドライバーとしてビルドできる唯一のドライバです。

その他のネットワーク ミニ リダイレクター ドライバーなどの他の Microsoft ネットワーク ミニ-リダイレクター、オペレーティング システムに付属しているとリンクするモノリシック ドライバーとして実装する必要があります、*rdbsslib.lib* スタティック ライブラリ WDK に付属します。

RDBSS は、ネットワーク ミニ リダイレクター ドライバー、I/O マネージャー、キャッシュ マネージャー、メモリ マネージャー、およびその他のカーネルのシステムとの通信を適切に定義されたメカニズムを使用します。

RDBSS は、多数のネットワークのミニ リダイレクターとオプションを設定し、さまざまな操作を実行するには、その他のカーネル システムで呼び出すことができるルーチンをエクスポートします。RDBSS によってエクスポートされたルーチンを呼び出すには、ネットワーク ミニ リダイレクター ドライバー (またはその他のカーネル ドライバー) WDK の適切なヘッダー ファイルが含まれています、名前でエクスポートされた RDBSS ルーチンを呼び出すし、適切なリンク *rdbsslib.lib* ファイル WDK と共にインストールされます。異なる *rdbsslib.lib* ファイルは、WDK Window Vista、Windows Server 2003、Windows XP、および Windows 2000 に付属します。

RDBSS の WDK のヘッダー ファイルもいくつかの RDBSS ルーチンの一部を直接呼び出すのではなく、ネットワークのミニ リダイレクター ドライバーによって使用するために推奨されているマクロを定義します。

すべての定義し、RDBSS で使用されるデータ構造により、検証で広く使用されているデータ構造体の先頭に特別な 4 バイト署名があります。RDBSS データ構造体のこれらの署名の値は、WDK のヘッダー ファイルで定義されている *nodetype.h* します。これらのデータ構造の署名は、トラブルシューティングとデバッグ RDBSS が使用され、ネットワーク ミニ リダイレクター ドライバー。

次のセクションでは、ルーチン RDBSS によってエクスポートされたもので、これらのルーチンの呼び出しに定義されているマクロのカテゴリの詳細ごとについて説明します。まず RDBSS および RDBSS によって定義されているマクロのようなリストによって提供されるルーチンのすべての一覧。

- [RDBSS によって提供されるルーチン](#)
- [RDBSS によって定義されているマクロ](#)

RDBSS とこれらのルーチンの呼び出しに定義されている RDBSS マクロによってエクスポートされたルーチンは、数、以下のさまざまなカテゴリに整理できます。

- [ドライバの登録と開始/停止の制御](#)
- [プールの割り当てと解放ルーチン](#)
- [タイマーとワーカー スレッド管理](#)
- [作業のキュー メカニズムをディスパッチ](#)
- [診断およびデバッグ](#)
- [ログ記録ルーチンとマクロ](#)

- その他のルーチン
- RX_コンテキストと IRP の管理
- 接続およびファイル構造の管理
- FCB とリソースの同期
- バッファリング状態のコントロール
- 低い I/O ルーチン
- 名前のキャッシュ管理
- プレフィックスのテーブルの管理
- 削除と清掃のコントロール
- 多重化 ID の管理
- エンジンの接続の管理

RDBSS が提供するルーチン

2020/04/24 • [Edit Online](#)

次のルーチンは、RDBSS によってエクスポートされます。

名前	説明
RxAcquireExclusiveFcbResourceInMRx	このリソース取得ルーチンは、排他モードでファイル制御ブロック (FCB) リソースを取得します。このルーチンは、FCB リソースが解放されるのを待機します。そのため、このルーチンはリソースが取得されるまで制御を返しません。このようなルーチンは、この FCB に関連付けられている RX_CONTEXT がキャンセルされた場合でも、FCB リソースを取得します。
RxAcquireSharedFcbResourceInMRx	このリソース取得ルーチンは、共有モードで FCB リソースを取得します。このルーチンは、既に排他的に取得されている場合、FCB リソースが解放されるのを待機します。そのため、このルーチンはリソースが取得されるまで制御を返しません。このようなルーチンは、この FCB に関連付けられている RX_CONTEXT がキャンセルされた場合でも、FCB リソースを取得します。
RxAcquireSharedFcbResourceInMRxEx	このリソース取得ルーチンは、共有モードで FCB リソースを取得します。このルーチンは、以前に排他的に取得された FCB リソースが解放されるまで待機します。このルーチンは、リソースが取得されるまで制御を返しません。このようなルーチンは、この FCB に関連付けられている RX_CONTEXT がキャンセルされた場合でも、FCB リソースを取得します。 このルーチンは、Windows Server 2003 Service Pack 1 (SP1) 以降でのみ使用できます。
RxAssert	このルーチンは、RDBSS checked ビルドでアサート文字列をカーネルデバッガーに送信します (インストールされている場合)。
RxAssociateContextWithMid	このルーチンは、指定された不透明なコンテキストを、MID_ATLAS データ構造体の使用可能な多重 ID (MID) に関連付けます。
RxCancelTimerRequest	このルーチンは、タイマー要求をキャンセルします。キャンセルされる要求は、ルーチンとコンテキストによって識別されます。
RxCeAllocateIrpWithMDL	このルーチンは、接続エンジンによって使用される IRP を割り当て、MDL を IRP に関連付けます。 このルーチンは、Windows XP でのみ使用できます。
RxCeBuildAddress	このルーチンは、トランスポートアドレスをトランスポートバインドに関連付けます。

名前	説明
RxCeBuildConnection	このルーチンは、ローカルの RDBSS 接続アドレスと指定されたリモートアドレスとの間の接続を確立します。このルーチンは、システムワーカーズレッドのコンテキストで呼び出す必要があります。
RxCeBuildConnectionOverMultipleTransports	このルーチンは、ローカルの RDBSS 接続アドレスと指定されたリモートアドレスとの間の接続を確立し、複数のトランスポートをサポートします。ローカルアドレスのセットが指定され、このルーチンは、ローカルアドレスに関連付けられているすべてのトランスポートを使用して対象サーバーに接続しようとします。接続オプションに応じて、1 つの接続が勝者として選択されます。このルーチンは、システムワーカーズレッドのコンテキストで呼び出す必要があります。
RxCeBuildTransport	このルーチンは、指定されたトランスポート名に RDBSS transport をバインドします。
RxCeBuildVC	このルーチンは、指定された接続に仮想回線を追加します。
RxCeCancelConnectRequest	このルーチンは、以前に発行された接続要求をキャンセルします。 このルーチンは現在実装されていないことに注意してください。
RxCeFreeIrp	このルーチンは、接続エンジンによって使用される IRP を解放します。 このルーチンは、Windows XP でのみ使用できます。
RxCeInitiateVCDDisconnect	このルーチンは、仮想回線で切断を開始します。このルーチンは、システムワーカーズレッドのコンテキストで呼び出す必要があります。
RxCeQueryAdapterStatus	このルーチンは、指定されたトランスポートの ADAPTER_STATUS 構造体を返します。
RxCeQueryInformation	このルーチンは、特定の仮想回線に関する情報を照会します。
RxCeQueryTransportInformation	このルーチンは、特定のトランスポートに対して、サービスの接続数と品質に関する情報を照会します。
RxCeSend	このルーチンは、指定された接続に沿ったトランスポートサービスデータユニット (TSDU) を仮想回線に送信します。
RxCeSendDatagram	このルーチンは、指定されたトランスポートアドレスに TSDU を送信します。
RxCeTearDownAddress	このルーチンは、トランスポートバインドからトランスポートアドレスを削除します。

名前	説明
RxCeTearDownConnection	このルーチンは、特定の接続を破棄します。
RxCeTearDownTransport	このルーチンは、指定されたトランスポートからバインドを解除します。
RxCeTearDownVC	このルーチンは、仮想接続を破棄します。
RxChangeBufferingState	このルーチンは、バッファリング状態の変更要求を処理するために呼び出されます。
RxCompleteRequest	このルーチンは、RX_CONTEXT 構造に関連付けられている IRP を完了するために使用されます。このルーチンは、RDBSS によって内部的に使用されるため、ネットワークミニダイレクタードライバでは使用できません。
RxCompleteRequest_Real	このルーチンは、RX_CONTEXT 構造に関連付けられている IRP を完了するために使用されます。このルーチンは、ネットワークミニダイレクターでは使用しないでください。
RxCreateMidAtlas	このルーチンは、MID_ATLAS データ構造体の新しいインスタンスを割り当て、それを初期化します。RDBSS は、このデータ構造で定義されている多重 ID (MID) を使用します。これは、ネットワーククライアント (ミニダイレクター) とサーバーの両方が、任意の接続で同時にアクティブになっている要求を区別できるようにするためのものです。
RxCreateNetFcb	このルーチンは、この FCB が開かれている NET_ROOT 構造体のメモリ内データ構造に新しい FCB 構造体を割り当て、初期化し、挿入します。割り当てられた構造体には、SRV_OPEN および FOBX 構造体の領域があります。このルーチンは、RDBSS によって内部的に使用されるため、ネットワークミニダイレクタードライバでは使用できません。
RxCreateNetFobx	このルーチンは、新しいファイルオブジェクト拡張 (FOBX) 構造体を割り当て、初期化し、挿入します。ネットワークミニダイレクターは、このルーチンを呼び出して、作成操作が正常に終了した時点で FOBX を作成する必要があります。
RxCreateNetRoot	このルーチンは、NET_ROOT 構造体を表すノードを構築し、関連付けられているデバイスオブジェクトの NET name テーブルに名前を挿入します。このルーチンは、RDBSS によって内部的に使用されるため、ネットワークミニダイレクターでは使用できません。
RxCreateRxContext	このルーチンは、新しい RX_CONTEXT 構造体を割り当て、データ構造体を初期化します。

名前	説明
RxCreateSrvCall	このルーチンは、サーバー呼び出しコンテキストを表すノードを構築し、RDBSS によって管理される net name テーブルに名前を挿入します。このルーチンは、RDBSS によって内部的に使用されるため、ネットワークミニドライクタードライバーでは使用できません。
RxCreateSrvOpen	このルーチンは、RDBSS によって使用されるメモリ内データ構造に新しい SRV_OPEN 構造体を割り当て、初期化し、挿入します。新しい構造体を割り当てる必要がある場合は、FOBx 構造体の領域があります。このルーチンは、RDBSS によって内部的に使用されるため、ネットワークミニドライクタードライバーでは使用できません。
RxCreateVNetRoot	このルーチンは、V_NET_ROOT 構造体を表すノードを構築し、その名前を NET name テーブルに挿入します。このルーチンは、RDBSS によって内部的に使用されるため、ネットワークミニドライクタードライバーでは使用できません。
RxDbgBreakPoint	このルーチンは、カーネルデバッガーによって処理される例外を発生させます (カーネルデバッガーがインストールされている場合)。それ以外の場合は、デバッグシステムによって処理されます。
RxDereference	このルーチンは、RDBSS によって使用されるいくつかの参照カウントデータ構造体のインスタンスの参照カウントをデクリメントします。
RxDereferenceAndDeleteRxContext_Real	このルーチンは、RX_CONTEXT 構造体を逆参照し、参照カウントがゼロになると、指定された RX_CONTEXT 構造体を RDBSS のメモリ内データ構造から解放し、削除します。
RxDestroyMidAtlas	このルーチンは、MID_ATLAS データ構造体の既存のインスタンスを破棄し、割り当てられたメモリを解放します。
RxDispatchToWorkerThread	このルーチンは、ワーカースレッドのコンテキストでルーチンを呼び出します。
RxDriverEntry	このルーチンは、RDBSS を初期化するために、その Driverentry からモノリシックネットワークミニドライクタードライバーによって呼び出されます。 モノリシックでないドライバーの場合、この初期化ルーチンは、 <i>rdbss</i> デバイスドライバーの driverentry ルーチンに相当します。
RxFinalizeConnection	このルーチンは、共有への接続を削除します。接続で開いたファイルは、指定された force レベルに応じて閉じられます。ネットワークミニドライクターは、接続の終了を強制するオプションが指定されていない限り、トランスポート接続をパフォーマンス上の理由から開いたままにしておくことができます。

名前	説明
RxFinalizeNetFcb	このルーチンは、指定された FCB 構造体を終了します。呼び出し元には、FCB に関連付けられている NET_ROOT 構造体の排他ロックが必要です。このルーチンは、RDBSS によって内部的に使用されるため、ネットワークミニドライバードライバーでは使用できません。
RxFinalizeNetFobx	このルーチンは、指定された FOBX 構造体を終了します。呼び出し元は、この FOBX に関連付けられた FCB に排他ロックを保持している必要があります。このルーチンは、RDBSS によって内部的に使用されるため、ネットワークミニドライバードライバーでは使用できません。
RxFinalizeNetRoot	このルーチンは、指定された NET_ROOT 構造体を終了します。呼び出し元は、この NET_ROOT 構造体に関連付けられているデバイスオブジェクトの、SRV_CALL 構造体を介して、そのロックに排他的にアクセスできる必要があります。このルーチンは、RDBSS によって内部的に使用されるため、ネットワークミニドライバードライバーでは使用できません。
RxFinalizeSrvCall	このルーチンは、指定された SRV_CALL 構造体を終了します。呼び出し元には、この SRV_CALL 構造体に関連付けられているデバイスオブジェクトのネットテーブルのロックに対する排他アクセス権が必要です。このルーチンは、RDBSS によって内部的に使用されるため、ネットワークミニドライバードライバーでは使用できません。
RxFinalizeSrvOpen	このルーチンは、指定された SRV_OPEN 構造体を終了します。このルーチンは、RDBSS によって内部的に使用されるため、ネットワークミニドライバードライバーでは使用できません。
RxFinalizeVNetRoot	このルーチンは、指定された V_NET_ROOT 構造体を終了します。呼び出し元には、この V_NET_ROOT 構造体に関連付けられているデバイスオブジェクトのネットテーブルのロックに対する排他アクセスが必要です。このルーチンは、RDBSS によって内部的に使用されるため、ネットワークミニドライバードライバーでは使用できません。
RxFinishFcbInitialization	このルーチンは、ネットワークミニドライバードライバーによる作成操作が正常に完了した後に、FCB の初期化を完了するために使用されます。
RxForceFinalizeAllVNetRoots	このルーチンは、特定の NET_ROOT 構造体に関連付けられているすべての V_NET_ROOT 構造体を強制的に終了します。呼び出し元には、この V_NET_ROOT 構造体に関連付けられているデバイスオブジェクトのネットテーブルのロックに対する排他アクセスが必要です。このルーチンは、RDBSS によって内部的に使用されるため、ネットワークミニドライバードライバーでは使用できません。
RxFsdDispatch	このルーチンは、i/o 要求パケット (IRP) を処理するための RDBSS のファイルシステムドライバードライバー (FSD) ディスパッチを実装しています。このルーチンは、要求の RDBSS 処理を開始するために、ドライバードリスパッチルーチンのネットワークミニドライバードライバーによって呼び出されます。

名前	説明
RxFsdPostRequest	このルーチンは、RX_CONTEXT 構造体によって指定された i/o 要求パケット (IRP) を、ファイルシステムプロセス (FSP) による処理のためにワーカーキューにキューに置いています。
RxGetFileSizeWithLock	このルーチンは、64ビット値が一貫して読み取られるようにロックを使用して、FCB 構造体からファイルサイズを取得します。
RxGetRDBSSProcess	このルーチンは、RDBSS カーネルプロセスによって使用されるメインスレッドのプロセスへのポインターを返します。
RxIndicateChangeOfBufferingState	このルーチンは、後で処理するために、バッファリング状態の変更要求 (oplock の中断を示すなど) を登録するために呼び出されます。
RxIndicateChangeOfBufferingStateForSrvOpen	このルーチンは、後で処理するために、バッファリング状態の変更要求 (oplock の中断を示すなど) を登録するために呼び出されます。
RxInferFileType	このルーチンは、RX_CONTEXT 構造体の Createoptions (■■■) フィールドからファイルの種類 (ディレクトリまたはディレクトリ以外) を推論しようとします。
RxInitializeContext	このルーチンは、新しく割り当てられた RX_CONTEXT 構造体を初期化します。
RxIsThisACscAgentOpen	このルーチンは、ファイルがユーザーモードのクライアント側キャッシュエージェントによって作成されたかどうかを判断します。 このルーチンは、Windows Server 2003 でのみ使用できます。
RxLockEnumerator	このルーチンは、FCB のファイルロックを列挙するために、ネットワークミニリダイレクターから呼び出されます。
RxLogEventDirect	このルーチンは、i/o エラーログにエラーを記録するために呼び出されます。このルーチンを直接呼び出すのではなく、 RxLogEvent または RxLogFailure マクロを使用することをお勧めします。
RxLogEventWithAnnotation	このルーチンは、i/o エラーログ構造を割り当て、ログ構造を格納し、この構造を i/o エラーログに書き込みます。
RxLogEventWithBufferDirect	このルーチンは、i/o エラーログにエラーを記録するために呼び出されます。このルーチンは、行番号と状態を i/o エラーログ構造に格納されているデータバッファにエンコードします。

名前	説明
RxLowIoCompletion	このルーチンは、処理が完了したときに、最初に保留が返された場合に、ネットワークミニダイレクタードライバーの低 i/o ルーチンによって呼び出される必要があります。
RxLowIoGetBufferAddress	このルーチンは、RX_CONTEXT 構造体のLowiocontext構造から、MDL に対応するバッファを返します。
RxMakeLateDeviceAvailable	このルーチンは、デバイスオブジェクトを変更して "遅延デバイス" を使用できるようにします。遅延デバイスとは、ドライバーの読み込みルーチンで作成されたものではありません。
RxMapAndDissociateMidFromContext	このルーチンは、MID を MID_ATLAS データ構造内の関連付けられたコンテキストにマップし、その中間とコンテキストとの関連付けを解除します。
RxMapMidToContext	このルーチンは、MID_ATLAS データ構造内の関連付けられているコンテキストに MID をマップします。
RxMapSystemBuffer	このルーチンは、i/o 要求パケット (IRP) からのシステムバッファアドレスを返します。
RxNameCacheActivateEntry	このルーチンは、名前キャッシュエントリを受け取り、有効期限とネットワークミニダイレクターコンテキストを更新します。次に、アクティブなリストにエントリを配置します。
RxNameCacheCheckEntry	このルーチンは、NAME_CACHE エントリの有効性を確認します。
RxNameCacheCreateEntry	このルーチンは、指定された名前の文字列を使用して NAME_CACHE 構造体を割り当て、初期化します。呼び出し元は、name cache コンテキストの追加のネットワークミニダイレクター要素を初期化し、そのエントリを name cache active list に配置する必要があります。
RxNameCacheExpireEntry	このルーチンは、NAME_CACHE エントリをフリーリストに挿入します。
RxNameCacheExpireEntryWithShortName	このルーチンは、指定された短いファイル名と一致する名前プレフィックスを持つすべての NAME_CACHE エントリの有効期限が切れます。
RxNameCacheFetchEntry	このルーチンは、NAME_CACHE エントリに対して指定された名前の文字列との一致を検索します。

名前	説明
RxNameCacheFinalize	このルーチンは、NAME_CACHE_CONTROL 構造体に関連付けられているすべての NAME_CACHE エントリのストレージを解放します。
RxNameCacheFreeEntry	このルーチンは、NAME_CACHE エントリのストレージを解放し、NAME_CACHE_CONTROL 構造体に関連付けられている NAME_CACHE キャッシュエントリの数を減らします。
RxNameCacheInitialize	このルーチンは、NAME_CACHE 構造体を初期化し、NAME_CACHE_CONTROL 構造体に関連付けます。
RxNewMapUserBuffer	このルーチンは、低 i/o に使用されたユーザーバッファのアドレスを返します。 このルーチンは、Windows XP および Windows 2000 でのみ使用できます。
RxpAcquirePrefixTableLockExclusive	このルーチンは、SRV_CALL と NET_ROOT の名前をカタログ化するために使用されるプレフィックステーブルに対して排他ロックを取得します。 このルーチンは、Windows XP および Windows 2000 でのみ使用できます。このルーチンは、RDBSS によって内部的に使用されるため、ネットワークミニドライクターでは使用できません。
RxpAcquirePrefixTableLockShared	このルーチンは、SRV_CALL と NET_ROOT の名前のカタログ化に使用されるプレフィックステーブルに対して共有ロックを取得します。 このルーチンは、Windows XP および Windows 2000 でのみ使用できます。このルーチンは、RDBSS によって内部的に使用されるため、ネットワークミニドライクタードライバーでは使用できません。
RxpDereferenceAndFinalizeNetFcb	このルーチンは参照カウントを逆参照し、FCB を終了します。 このルーチンは、Windows Server 2003 Service Pack 1 (SP1) 以降でのみ使用できます。
RxpDereferenceNetFcb	このルーチンは、FCB の参照カウントをデクリメントします。
RxPostOneShotTimerRequest	このルーチンは、ドライバーがワンショットタイマー要求を初期化するために使用されます。タイマーの有効期限が切れると、このルーチンに渡されるワーカーレッドルーチンが1回呼び出されます。
RxPostRecurrentTimerRequest	このルーチンは、定期的なタイマー要求を初期化するために使用されます。このルーチンに渡されるワーカーレッドルーチンは、このルーチンへの入力パラメーターに基づいて定期的なタイマーが起動されると、一定の間隔で呼び出されます。
RxPostToWorkerThread	このルーチンは、ワーカーレッドのコンテキストでルーチンを呼び出します。

RxpReferenceNetFcb	このルーチンは、FCB の参照カウントをインクリメントします。
RxPrefixTableLookupName	このルーチンは、SRV_CALL と NET_ROOT の名前をカタログ化するために使用されるプレフィックステーブル内の名前を検索し、基になるポインターから含んでいる構造体に変換します。 このルーチンは、RDBSS によって内部的に使用されるため、ネットワークミニダイレクタードライバーでは使用できません。
RxpReleasePrefixTableLock	このルーチンは、SRV_CALL と NET_ROOT の名前をカタログ化するために使用されるプレフィックステーブルのロックを解放します。 このルーチンは、Windows XP および Windows 2000 でのみ使用できます。このルーチンは、RDBSS によって内部的に使用されるため、ネットワークミニダイレクタードライバーでは使用できません。
RxPrepareContextForReuse	このルーチンは、実行されたすべての操作固有の割り当てと取得をリセットすることによって、再利用できるように RX_CONTEXT 構造を準備します。IRP から取得したパラメーターは変更されません。このルーチンは、RDBSS によって内部的に使用されるため、ネットワークミニダイレクターでは使用できません。
RxPrepareToReparseSymbolicLink	このルーチンは、再解析を容易にするためにファイルオブジェクト名を設定します。このルーチンは、シンボリックリンクをスキャンするためにネットワークミニダイレクターによって使用されます。このルーチンは、RDBSS によって内部的に使用されるため、ネットワークミニダイレクターでは使用できません。
Rxp Track■	このルーチンは、チェックを行うビルドで、SRV_CALL、NET_ROOT、V_NET_ROOT、FOBX、FCB、SRV_OPEN の各構造を逆参照する要求を追跡するために使用されます。これらの逆参照要求のログは、ログ記録システムおよび WMI からアクセスできます。 リテールビルドの場合、このルーチンは何も行いません。
Rxp Trackreference	このルーチンは、チェックされたビルドで、SRV_CALL、NET_ROOT、V_NET_ROOT、FOBX、FCB、および SRV_OPEN 構造体を参照する要求を追跡するために使用されます。これらの参照要求のログは、ログ記録システムおよび WMI からアクセスできます。 リテールビルドの場合、このルーチンは何も行いません。
RxpUnregisterMinirdr	このルーチンは、RDBSS を使用してドライバーの登録を解除し、内部 RDBSS 登録テーブルから登録情報を削除するために、ネットワークミニダイレクタードライバーによって呼び出されます。
RxPurgeAllFobxs	このルーチンは、ネットワークミニダイレクターに関連付けられているすべての FOBX 構造を削除します。
RxPurgeRelatedFobxs	このルーチンは、NET_ROOT 構造体に関連付けられているすべての FOBX 構造体を削除します。

名前	説明
RxReassociateMid	このルーチンは、MID を別のコンテキストに再関連付けします。
RxReference	このルーチンは、RDBSS によって使用されるいくつかの参照カウントデータ構造体のインスタンスの参照カウントをインクリメントします。
RxRegisterMinirdr	このルーチンは、ネットワークミニダイレクタードライバーによって呼び出され、ドライバーを RDBSS に登録します。これにより、登録情報が内部登録テーブルに追加されます。RDBSS は、ネットワークミニダイレクター用のデバイスオブジェクトも構築します。
RxReleaseFcbResourceInMRx	このルーチンは、 RxAcquireExclusiveFcbResourceInMRx または RxAcquireSharedFcbResourceInMRx を使用して取得した FCB リソースを解放します。
RxReleaseFcbResourceForThreadInMRx	このルーチンは、 RxAcquireSharedFcbResourceInMRxEx を使用して取得した FCB リソースを解放します。 このルーチンは、Windows Server 2003 Service Pack 1 (SP1) 以降でのみ使用できます。
RxResumeBlockedOperations_Serially	このルーチンは、シリアル化されたブロッキング i/o キューで次の待機中のスレッド (存在する場合) をウェイクアップします。
RxScavengeAllFobxs	このルーチンは、特定のネットワークミニダイレクターデバイスオブジェクトに関連付けられているすべての FOBX 構造を清掃します。
RxScavengeFobxsForNetRoot	このルーチンは、指定された NET_ROOT 構造体に関連するすべての FOBX 構造体を清掃します。
RxSetDomainForMailslotBroadcast	このルーチンは、mailslots がドライバーでサポートされている場合に、メールスロットブロードキャストに使用されるドメインを設定するために、ネットワークミニダイレクタードライバーによって呼び出されます。
RxSetMinirdrCancelRoutine	このルーチンは、RX_CONTEXT 構造体に対してネットワークミニダイレクターのキャンセルルーチンを設定します。
RxSetSrvCallDomainName	このルーチンは、特定のサーバー (SRV_CALL 構造) に関連付けられているドメイン名を設定します。
RxSpinDownMRxDispatcher	このルーチンは、ネットワークミニダイレクターのディスパッチャーコンテキストを破棄します。 このルーチンは、Windows XP 以降でのみ使用できます。

名前	説明
RxStartMinirdr	このルーチンは、を呼び出して自身を登録するネットワークミニダイレクターを起動します。また、ドライバが UNC 名をサポートしている場合は、RDBSS (MUP) と共 Multiple UNC Provider にネットワークミニダイレクタードライバが UNC (汎用名前付け規則) プロバイダーとして登録されます。
RxStopMinirdr	このルーチンは、ネットワークミニダイレクタードライバを停止します。停止したドライバは、新しいコマンドを受け付けなくなります。
RxUnregisterMinirdr	このルーチンは、 <i>rxstruc</i> で定義されたインライン関数で、RDBSS でドライバの登録を解除し、内部 RDBSS 登録テーブルから登録情報を削除するために、ネットワークミニダイレクタードライバによって呼び出されます。 RxUnregisterMinirdr inline 関数は、内部的に RxpUnregisterMinirdr を呼び出します。
_RxAllocatePoolWithTag	このルーチンは、メモリの問題のインスタンスをキャッチするために使用できる、ブロックの先頭に4バイトのタグを持つプールからメモリを割り当てます。 このルーチンを直接呼び出すのではなく、RxAllocatePoolWithTagマクロを使用することをお勧めします。
_RxCheckMemoryBlock	このルーチンは、メモリブロックに特別な RX_POOL_HEADER HEADER 署名があるかどうかをチェックします。ネットワークミニダイレクタードライバは、ルーチンを使用するために割り当てられたメモリに、この特殊な署名ブロックを追加する必要があることに注意してください。 この特別なヘッダーブロックが実装されていないため、このルーチンは使用しないでください。
_RxFreePool	このルーチンは、メモリプールを解放します。 このルーチンを直接呼び出すのではなく、RxFreePoolマクロを使用することをお勧めします。
_RxLog	このルーチンは、書式指定文字列と可変数のパラメーターを受け取り、ログ記録が有効になっている場合に i/o エラーログエントリとして構造体を作成するための出力文字列を書式設定します。 このルーチンを直接呼び出すのではなく、RxLogマクロを使用することをお勧めします。 このルーチンは、Windows Server 2003、Windows XP、および Windows 2000 の RDBSS のチェックされたビルドでのみ使用できます。
__RxFillAndInstallFastIoDispatch	このルーチンは、通常のディスパッチ i/o ベクターと同一の高速 i/o ディスパッチベクターを入力し、渡されたデバイスオブジェクトに関連付けられているドライバオブジェクトにインストールします。 このルーチンは、モノリシックでないドライバに対してのみ実装され、モノリシックドライバでは何も行いません。

名前	説明
__RxSynchronizeBlockingOperations	このルーチンは、ブロッキング i/o を同じ作業キューに同期するために使用されます。このルーチンは、名前付きパイプ操作を同期するために RDBSS によって内部的に使用されます。ネットワークミニリダイレクターはこのルーチンを使用して、ネットワークミニリダイレクターによって管理されている別のキューで操作を同期します。 このルーチンは、Windows Server 2003 でのみ使用できます。
__RxSynchronizeBlockingOperationsMaybeDroppingFileLock	このルーチンは、ブロッキング i/o を同じ作業キューに同期するために使用されます。このルーチンは、名前付きパイプ操作を同期するために RDBSS によって内部的に使用されます。ネットワークミニリダイレクターはこのルーチンを使用して、ネットワークミニリダイレクターによって管理されている別のキューで操作を同期します。 このルーチンは、Windows XP および Windows 2000 でのみ使用できます。

RDBSS で定義されたマクロ

2020/04/24 • [Edit Online](#)

便利なマクロの多くは、これらの RDBSS ルーチンやその他のカーネルルーチンを呼び出す Windows Driver Kit (WDK) ヘッダーファイルで定義されています。これらのマクロの一部は、通常、RDBSS ルーチンを直接呼び出す代わりに使用されます。これらのマクロの一部は、便利なルーチンとして使用されます。

次のマクロは、RDBSS で定義されています。

III	II
RxAcquirePrefixTableLockExclusive (テーブル、待機)	このマクロは、変更操作のために、排他モードでプレフィックステーブルロックを取得します。
RxAcquirePrefixTableLockShared (テーブル、待機)	このマクロは、参照操作のために、共有モードでプレフィックステーブルロックを取得します。
RxAllocatePoolWithTag (<i>type</i> , <i>size</i> , <i>tag</i>)	Checked ビルドでは、このマクロは、メモリ trashing のインスタンスをキャッチするために使用できる、ブロックの先頭に4バイトのタグがあるプールからメモリを割り当てます。 リテールビルドでは、このマクロは ExAllocatePoolWithTag への直接の呼び出しになります。
RxCheckMemoryBlock (<i>ptr</i>)	Checked ビルドでは、このマクロはメモリブロックに特別な RX_POOL_HEADER HEADER 署名があるかどうかをチェックします。 リテールビルドでは、このマクロは何も行いません。
RxDereferenceAndFinalizeNetFcb (<i>Fcb</i> , <i>RxContext</i> , <i>RecursiveFinalize</i> , <i>forcefinalize</i>)	このマクロは、FCB 構造体での逆参照操作を追跡するために使用されます。 このマクロは参照カウントを操作し、最後の逆参照呼び出しの状態も返すことに注意してください。
RxDereferenceNetFcb (<i>Fcb</i>)	このマクロは、FCB 構造体での逆参照操作を追跡するために使用されます。 このマクロは参照カウントを操作し、最後の逆参照呼び出しの状態も返すことに注意してください。
RxDereferenceNetFobx (<i>fobx</i> , <i>LockHoldingState</i>)	このマクロは、FOBx 構造体での逆参照操作を追跡するために使用されます。
RxDereferenceNetRoot (<i>netroot</i> , <i>LockHoldingState</i>)	このマクロは、NET_ROOT 構造体の逆参照操作を追跡するために使用されます。

III	II
RxDereferenceSrvCall (<i>srvcall</i> , <i>LockHoldingState</i>)	このマクロは、SRV_CALL 構造体の逆参照操作を追跡するために使用されます。
RxDereferenceSrvOpen (<i>srvopen</i> , <i>LockHoldingState</i>)	このマクロは、SRV_OPEN 構造体の逆参照操作を追跡するために使用されます。
RxDereferenceVNetRoot (<i>vnetroot</i> , <i>LockHoldingState</i>)	このマクロは、V_NET_ROOT 構造体の逆参照操作を追跡するために使用されます。
RxFcbAcquiredShared (<i>RXCONTEXT</i> , <i>FCB</i>)	このマクロは、現在のスレッドが共有モードの通常のリソースにアクセスできるかどうかを確認します。このマクロは、 ExIsResourceAcquiredSharedLite ルーチンを呼び出します。
RxFillAndInstallFastIoDispatch(__ <i>devobj</i> , __ <i>fa</i>)	このマクロは、 __RxFillAndInstallFastIoDispatch を呼び出して、通常のディスパッチ i/o ベクターと同一の高速 i/o ディスパッチベクターを入力し、渡されたデバイスオブジェクトに関連付けられた driver オブジェクトにインストールします。
RxFreePool (<i>ptr</i>)	チェックされたビルドでは、このマクロはメモリプールを解放します。 リテールビルドでは、このマクロは Exfreepool への直接の呼び出しになります。
RxIsFcbAcquiredShared (<i>FCB</i>)	このマクロは、現在のスレッドが共有モードの通常のリソースにアクセスできるかどうかを確認します。このマクロは、 ExIsResourceAcquiredSharedLite ルーチンを呼び出します。
RxIsFcbAcquiredExclusive (<i>FCB</i>)	このマクロは、現在のスレッドが排他モードで通常のリソースにアクセスできるかどうかを確認します。このマクロは、 ExIsResourceAcquiredExclusiveLite ルーチンを呼び出します。
RxIsFcbAcquired (<i>FCB</i>)	このマクロは、現在のスレッドが共有モードまたは排他モードで通常のリソースにアクセスできるかどうかを確認します。このマクロは、 ExIsResourceAcquiredSharedLite ルーチンと ExIsResourceAcquiredExclusiveLite ルーチンを呼び出します。
RxIsPrefixTableLockAcquired (テーブル)	このマクロは、プレフィックステーブルロックが排他的モードまたは共有モードのいずれかで取得されたかどうかを示します。
RxIsPrefixTableLockExclusive (テーブル)	このマクロは、プレフィックステーブルロックが排他モードで取得されたかどうかを示します。

III	II
RxLog(<i>Args</i>)	チェックされたビルドでは、このマクロは RxLog ルーチンを呼び出します。 リテールビルドでは、このマクロは何も行いません。 RxLog への引数は、ログ記録をオフにする必要がある場合に null 呼び出しに変換できるようにするために、追加の <code>かつ</code> このペアで囲む必要があることに注意してください。
RxLogEvent (<i>_DeviceObject</i> , <i>id</i> , <i>EventId</i> , <i>Status</i>)	このマクロは、 RxLogEventDirect ルーチンを呼び出します。
RxLogFailure (<i>_DeviceObject</i> , <i>id</i> , <i>EventId</i> , <i>Status</i>)	このマクロは、 RxLogEventDirect ルーチンを呼び出します。
RxLogFailureWithBuffer (<i>_DeviceObject</i> , " <i> "</i> ", " <i> "</i> ", " <i>状態</i> ", " <i>バッファの長さ</i> ", " <i>長さなし</i> ")	このマクロは、 RxLogEventWithBufferDirect ルーチンを呼び出します。
RxLogRetail(<i>Args</i>)	チェックされたビルドでは、このマクロは RxLog ルーチンを呼び出します。 リテールビルドでは、このマクロは何も行いません。 RxLogRetail への引数は、ログ記録をオフにする必要がある場合に null 呼び出しに変換できるようにするために、追加の <code>かつ</code> このペアで囲む必要があることに注意してください。
RxReferenceNetFcb (<i>Fcb</i>)	このマクロは、FCB 構造体に対する参照操作を追跡するために使用されます。
RxReferenceNetFobx (<i>fobx</i>)	このマクロは、FOBx 構造体の参照操作を追跡するために使用されます。これらの参照操作のログは、ログ記録システムおよび WMI からアクセスできます。
RxReferenceNetRoot (<i>netroot</i>)	このマクロは、NET_ROOT 構造体の参照操作を追跡するために使用されます。これらの参照操作のログは、ログ記録システムおよび Windows Management Instrumentation (WMI) からアクセスできます。
RxReferenceSrvCall (<i>srvcall</i>)	このマクロは、遅延プロシージャ呼び出し (DPC) レベルではない SRV_CALL 構造体に対する参照操作を追跡するために使用されます。
RxReferenceSrvCallAtDpc (<i>srvcall</i>)	このマクロは、DPC レベルで SRV_CALL 構造体に対する参照操作を追跡するために使用されます。
RxReferenceSrvOpen (<i>srvopen</i>)	このマクロは、SRV_OPEN 構造体の参照操作を追跡するために使用されます。

III	II
RxReferenceVNetRoot (<i>vnetroot</i>)	このマクロは、V_NET_ROOT 構造体の参照操作を追跡するために使用されます。
RxReleasePrefixTableLock (テーブル)	このマクロは、プレフィックステーブルロックを解放します。
RxSynchronizeBlockingOperations (<i>RXCONTEXT</i> 、 <i>FCB</i> 、 <i>ioqueue</i>)	このマクロは、ブロッキング i/o 要求を同じ作業キューに同期します。Windows Server 2003 では、このマクロは <i>Dropfcblock</i> パラメーターを FALSE に設定して __RxSynchronizeBlockingOperations ルーチン呼び出します。 Windows XP および Windows 2000 では、このマクロは <i>Dropfcblock</i> パラメーターを FALSE に設定して __RxSynchronizeBlockingOperationsMaybeDroppingFcbLock ルーチン呼び出します。
RxSynchronizeBlockingOperations (<i>RXCONTEXT</i> 、 <i>FCB</i> 、 <i>ioqueue</i>)	このマクロは、ブロッキング i/o 要求を同じ作業キューに同期します。Windows Server 2003 では、このマクロは <i>Dropfcblock</i> パラメーターを TRUE に設定して __RxSynchronizeBlockingOperations ルーチン呼び出します。 Windows XP および Windows 2000 では、このマクロは <i>Dropfcblock</i> パラメーターを TRUE に設定して __RxSynchronizeBlockingOperationsMaybeDroppingFcbLock ルーチン呼び出します。

ドライバーの登録と開始/停止制御

2020/04/24 • [Edit Online](#)

オペレーティングシステムが起動すると、Windows によって、レジストリの設定に基づいて、RDBSS およびすべてのネットワークミニリダイレクタードライバーが読み込まれます。Rdbsslib を使用して静的にリンクされているモノリシックネットワークミニリダイレクタードライバーの場合、ドライバーは **RxDriverEntry** ルーチンを **driverentry** ルーチンから呼び出して、ネットワークにリンクされている rdbsslib ライブラリのコピーを初期化する必要があります。driver。この場合、他の RDBSS ルーチンが呼び出されて使用される前に、**RxDriverEntry** ルーチンを呼び出す必要があります。モノリシックではないネットワークミニリダイレクタードライバー (Microsoft SMB リダイレクター) の場合、rdbss デバイスドライバーは、読み込まれると、独自の **Driverentry** ルーチンで初期化されます。

ドライバーがカーネルによって読み込まれ、ドライバーがアンロードされたときに RDBSS で登録解除されると、ネットワークミニリダイレクターは RDBSS に登録します。ネットワークミニリダイレクターは、RDBSS からエクスポートされた登録ルーチン **RxRegisterMinirdr** を呼び出すことによって読み込まれたことを RDBSS に通知します。この登録プロセスの一環として、ネットワークミニリダイレクターは **RxRegisterMinirdr** にパラメーターを渡します。このパラメーターは、大きな構造体である **minirdr_DISPATCH** へのポインターです。この構造体には、ネットワークミニリダイレクターの構成情報と、ネットワークミニリダイレクターカーネルドライバーによって実装されるコールバックルーチンへのポインターのディスパッチテーブルが含まれています。RDBSS は、このコールバックルーチンの一覧を使用してネットワークミニリダイレクタードライバーを呼び出します。

RxRegisterMinirdr ルーチンは、最上位レベルの RDBSS ディスパッチルーチン **RxFsdDispatch** を指すように、ネットワークミニリダイレクタードライバーのすべてのドライバーディスパッチルーチンを設定します。ネットワークミニリダイレクターは、独自のエントリポイントを保存し、**RxRegisterMinirdr** への呼び出しが返された後、またはを呼び出すときに特別なパラメーターを設定することにより、この動作をオーバーライドできます。**RxRegisterMinirdr**。

ネットワークミニリダイレクタードライバーは、**MRxStart** ルーチンへの呼び出しを受け取るまで、実際には操作を開始しません。これは、**minirdr_DISPATCH** 構造体で渡されたコールバックルーチンの1つです。**MrxStart** コールバックルーチンは、ネットワークミニリダイレクターが独自のドライバーディスパッチエントリポイントを保持しない限り、操作のコールバックルーチンを受信する場合は、ネットワークミニリダイレクタードライバーによって実装される必要があります。それ以外の場合、RDBSS は、**MrxStart** が正常に返されるまで、次の i/o 要求パケットだけをドライバーに送信できます。

- デバイスに対する IRP 要求では、IRPSP の FileObject へのファイル名の長さが 0 で、FileObject への関連性のある Fileobject が NULL のデバイス操作が作成されます。

その他の IRP 要求では、RDBSS ディスパッチルーチン **RxFsdDispatch** は、_開始されていない状態_リダイレクター_状態の状態を返します。

RDBSS ディスパッチルーチンは、次の i/o 要求パケットに対するすべての要求も失敗します。

- _メールスロットを作成_IRP_MJ
- IRP_MJ_パイプ_名前を作成します

ネットワークミニリダイレクターによって実装される **MrxStart** コールバックルーチンは、**RxStartMinirdr** ルーチンが呼び出されたときに RDBSS によって呼び出されます。RDBSS **RxStartMinirdr** ルーチンは、通常、ネットワークミニリダイレクターを開始するために、ユーザーモードのアプリケーションまたはサービスからのファイルシステム制御コード (FSCTL) または i/o 制御コード (IOCTL) 要求の結果として呼び出されます。**RxRegisterMinirdr** への呼び出しが成功した後、ネットワークミニリダイレクターの **driverentry** ルーチンから **RxStartMinirdr** を呼び出すことはできません。一部の開始処理では、ドライバーの初期化が必要になるためです。完了。 **RxStartMinirdr** 呼び出しが受信されると、RDBSS はネットワークミニリダイレクターの **MrxStart** ルーチンを呼び出すことによって開始プロセスを完了します。**MrxStart** を呼び出すと成功が返された場合、RDBSS は RDBSS 内のミニリダイレクターの内部状態を RDBSS_開始に設定します。

RDBSS は、**RxSetDomainForMailslotBroadcast** のルーチンをエクスポートして、メールスロットブロードキャスト用にドメインを設定します。このルーチンは、ネットワークミニリダイレクターが mailslots をサポートしている場合に、登録時に使用されます。

RDBSS によってエクスポートされた便宜的なルーチン [__RxFillAndInstallFastIoDispatch](#) を使用して、i/o 要求処理を同等の高速 i/o ディスパッチに処理するための、すべての IRP_MJ_XXX ドライバルーチンポインターをコピーできます。ベクターですが、このルーチンはモノリシックでないドライバーに対してのみ機能します。

また、RDBSS は、ネットワークミニリダイレクターが開始または停止していることを RDBSS に通知するルーチンもエクスポートします。これらの呼び出しは、ネットワークミニリダイレクターに、リダイレクターを開始および停止するユーザーモード管理サービスまたはユーティリティアプリケーションが含まれている場合に使用されます。このユーザーモードサービスまたはアプリケーションは、ネットワークミニリダイレクタードライバーにカスタム FSCTL または IOCTL 要求を送信して、開始または停止する必要があることを示すことができます。リダイレクターは、RDBSS [RxStartMinirdr](#) または [RxStopMinirdr](#) ルーチンを呼び出して、このネットワークミニリダイレクターを開始または停止するように RDBSS に通知できます。

次の表は、RDBSS ドライバーの登録と開始/停止の制御ルーチンを示しています。

名前	説明
RxDriverEntry	このルーチンは、RDBSS を初期化するために、 Driverentry ルーチンからモノリシックネットワークミニリダイレクタードライバーによって呼び出されます。 モノリシックでないドライバーの場合、この初期化ルーチンは、rbss デバイスドライバーの Driverentry ルーチンに相当します。
RxRegisterMinirdr	このルーチンは、ネットワークミニリダイレクタードライバーによって呼び出され、ドライバーを RDBSS に登録します。これにより、登録情報が内部登録テーブルに追加されます。RDBSS は、ネットワークミニリダイレクター用のデバイスオブジェクトも構築します。
RxSetDomainForMailslotBroadcast	このルーチンは、mailslots がドライバーでサポートされている場合に、メールスロットブロードキャストに使用されるドメインを設定するために、ネットワークミニリダイレクタードライバーによって呼び出されます。
RxStartMinirdr	このルーチンは、を呼び出して自身を登録するネットワークミニリダイレクターを起動します。また、ドライバーが UNC 名のサポートを示す場合は、RDBSS はネットワークミニリダイレクタードライバーを、MUP と共に UNC プロバイダーとして登録します。
RxStopMinirdr	このルーチンは、ネットワークミニリダイレクタードライバーを停止します。停止したドライバーは、IOCTL または FSCTL 要求を除く新しいコマンドを受信しなくなります。
RxpUnregisterMinirdr	このルーチンは、ネットワークミニリダイレクタードライバーによって呼び出され、ドライバーを RDBSS に登録解除し、内部 RDBSS 登録テーブルから登録情報を削除します。
RxUnregisterMinirdr	このルーチンは、rxstruc で定義されたインライン関数で、RDBSS にドライバーを登録解除し、内部 RDBSS 登録テーブルから登録情報を削除するために、ネットワークミニリダイレクタードライバーによって呼び出されます。 RxUnregisterMinirdr inline 関数は、内部的に RxpUnregisterMinirdr を呼び出します。

'''	''
__RxFillAndInstallFastIoDispatch	このルーチンは、通常のディスパッチ i/o ベクターと同一の高速 i/o ディスパッチベクターを入力し、渡されたデバイスオブジェクトに関連付けられた driver オブジェクトにインストールします。

次のマクロは、これらのルーチンの1つを呼び出す `mrxd.h` ヘッダーファイルで定義されています。通常、このマクロは、[__RxFillAndInstallFastIoDispatch](#)ルーチンを直接呼び出す代わりに使用されます。

'''	''
<code>RxFillAndInstallFastIoDispatch(__ devobj __ fa)</code>	このマクロは、 __RxFillAndInstallFastIoDispatch を呼び出して、通常のディスパッチ i/o ベクターと同一の高速 i/o ディスパッチベクターを入力し、渡されたデバイスオブジェクトに関連付けられた driver オブジェクトにインストールします。

プールの割り当てとフリールーチン

2020/04/24 • [Edit Online](#)

RDBSS には、プール割り当てに使用する多くのルーチンが用意されています。通常、これらのルーチンは、これらのルーチンを直接呼び出すのではなく、マクロを使用して呼び出されます。これらのマクロは、リテールビルドとチェックビルドの違いを自動的に処理します。

チェックを行ったビルドでは、これらのルーチンは、通常のカーネル割り当ておよびフリールーチンのラッパーを追加するように設計されています。プールの割り当てと解放のルーチンのラッパーは、追加のデバッグ情報を提供し、カーネルプールの割り当てと解放のルーチンを呼び出す前に、さまざまな種類のチェックと保護を実行するルーチンのセットを呼び出します。ただし、これらの機能は、これらの割り当ておよびフリールーチンには現在実装されていませんが、今後のリリースで追加される可能性があります。

無料のビルドでは、これらのルーチンは、カーネルの割り当てとフリールーチン、[ExAllocatePoolWithTag](#)、および[exfreepool](#)への直接呼び出しになります。

次の表に、RDBSS プールの割り当てとフリールーチンを示します。

名前	説明
_RxAllocatePoolWithTag	このルーチンは、メモリの問題をキャッチするために、ブロックの先頭に 4 バイトのタグがあるプールからメモリを割り当てます。 このルーチンを直接使用するのではなく、 RxAllocatePoolWithTag マクロを呼び出すことをお勧めします。
_RxCheckMemoryBlock	このルーチンは、メモリブロックに特別な <code>RX_POOL_HEADER</code> HEADER 署名があるかどうかをチェックします。ネットワークミニドライバードライバーは、ルーチンを使用するために割り当てられたメモリに、この特殊な署名ブロックを追加する必要があることに注意してください。 この特別なヘッダーブロックが実装されていないため、このルーチンは使用しないでください。
_RxFreePool	このルーチンは、メモリプールを解放します。 このルーチンを直接使用するのではなく、 RxFreePool マクロを呼び出すことをお勧めします。

`Ntrxdef.h` ヘッダーファイルで定義されている多くのマクロは、これらのルーチンを呼び出します。前の表に示したルーチンを直接呼び出すのではなく、通常は次のマクロを使用します。

名前	説明
<code>RxAllocatePoolWithTag (type, size, tag)</code>	Checked ビルドでは、このマクロは、メモリ trashing のインスタンスをキャッチするのに役立つ 4 バイトのタグを持つプールからメモリを割り当てます。 リテールビルドでは、このマクロは ExAllocatePoolWithTag への直接の呼び出しになります。

Ⅲ	Ⅱ
RxCheckMemoryBlock (<i>ptr</i>)	Checked ビルドでは、このマクロはメモリブロックに特別な RX_POOL_HEADER HEADER 署名があるかどうかをチェックしま す。 リテールビルドでは、このマクロは何も行いません。
RxFreePool (<i>ptr</i>)	チェックされたビルドでは、このマクロはメモリプールを解放します。 リテールビルドでは、このマクロはExfreepoolへの直接の呼び出し になります。

タイマーとワーカー スレッド管理

2020/04/24 • [Edit Online](#)

RDBSS には、ワーカー スレッド管理用のタイマー ルーチンがいくつか用意されています。これらのサービスは、すべてのネットワークミニリダイレクター ドライバーに提供されます。次の種類のタイマー ルーチンを使用できます。

- 定期的なトリガー
- ワンショット通知

タイマーは、デバイスオブジェクトとワーカー スレッド ルーチンに関連付けられています。タイマーが期限切れになると、最初の `RxPostOneShotTimerRequest` または `RxPostRecurrentTimerRequest` ルーチンに入力パラメーターとして渡されたワーカー スレッド ルーチンが呼び出されます。

次の RDBSS timer ルーチンが含まれています。

名前	説明
<code>RxCancelTimerRequest</code>	このルーチンは、タイマー要求をキャンセルします。キャンセルされる要求は、ルーチンへのポインターとコンテキストパラメーターによって識別されます。
<code>RxPostOneShotTimerRequest</code>	このルーチンは、ドライバーがワンショットタイマー要求を初期化するために使用されます。タイマーの有効期限が切れると、このルーチンに渡されるワーカー スレッド ルーチンが1回呼び出されます。
<code>RxPostRecurrentTimerRequest</code>	このルーチンは、定期的なタイマー要求を初期化します。このルーチンに渡されるワーカー スレッド ルーチンは、このルーチンへの入力パラメーターに基づいて定期的なタイマーが起動されると、一定の間隔で呼び出されます。

ワークキューのディスパッチメカニズム

2020/04/24 • [Edit Online](#)

RDBSS は、Windows カーネルワークキューを使用して、後で実行するために複数のスレッドで操作をディスパッチします。ネットワークミニドライバードライバーは、RDBSS によって管理される作業キューを使用して、後で実行するためのディスパッチ操作を行うことができます。

RDBSS には、RDBSS で使用されるディスパッチ機構を実装するいくつかのルーチンが用意されています。これらのルーチンは、ネットワークミニドライバードライバーでも使用できます。

RDBSS は、作業項目をデバイス単位で追跡します。これにより、RDBSS はネットワークミニドライバークの読み込みとアンロードに関連する競合状態を処理できます。これは、すべてのリソースを使用して起因が1つのネットワークミニドライバークを防ぐことができるように、RDBSS のメカニズムを提供します。

作業項目のディスパッチが避けられない特定のシナリオがあります。このようなシナリオでは、メモリの割り当てが頻繁に行われず、操作が解放されるのを防ぐために、WORK_QUEUE_ITEM が別のデータの一部として割り当てられます。ディスパッチがあまり行われない他のシナリオでは、必要になるまでメモリの割り当てが行われないようにします。RDBSS work queue の実装では、これらの両方のシナリオを、作業キュー要求のディスパッチとポストイングの形式で提供します。[RxDispatchToWorkerThread](#) ルーチンを使用してディスパッチする場合は、作業_キュー_項目のメモリが、呼び出し元によって割り当てられる必要があります。[RxPostToWorkerThread](#) ルーチンを使用して投稿する場合、WORK_QUEUE_ITEM のメモリは、呼び出し元によって割り当てられる必要があります。

ワーカースレッドへの操作のディスパッチには、次の2つの一般的なケースがあります。

- 非常に頻度の低い操作の場合は、[RxDispatchToWorkerThread](#) ルーチンを使用して、必要に応じて作業キュー項目のメモリを動的に割り当てたり解放したりすることによって、メモリの使用量を節約します。
- 操作を繰り返しディスパッチする場合は、[RxPostToWorkerThread](#) ルーチンを使用して、ディスパッチされるデータ構造の一部として作業_キュー_項目を事前に割り当てて、時間を節約します。

2つのディスパッチ操作の間のトレードオフは、時間と領域 (メモリ使用) です。

RDBSS のディスパッチメカニズムは、プロセスごとに複数のレベルの作業キューを提供します。現在サポートされている作業キューのレベルは次のとおりです。

- 重大
- 遅延
- ハイパークリティカル

Critical と遅延の区別は、優先度の1つです。ハイパークリティカルレベルは他の2つとは異なり、ルーチンはブロックしないでください (任意のリソースを待機する必要があります)。この要件を強制することはできないため、ディスパッチメカニズムの有効性は、クライアントの暗黙的な連携に依存します。

RDBSS でのワークキューの実装は、KQUEUE 実装に基づいて構築されています。追加のサポートには、作業項目をアクティブに待機している多数のスレッドの規則が含まれます。各ワークキューのデータ構造は、非ページプールのメモリから割り当てられ、独自の同期機構 (スピンロック) を持ちます。

RDBSS では、ブックの情報 (キューの状態と種類など) だけでなく、ワークキューの有効期間に収集される統計も保持されます。これにより、ワークキューのチューニングに関する有益な情報を得ることができます。処理された項目の数、処理する必要がある項目の数、および累積キューの長さは構造になります。累積キューの長さは重要なメトリックであり、追加の作業項目がキューに登録されるたびに処理を待機している項目の数の合計を表します。累積キューの長さは、処理された項目の合計数と処理される項目の数の合計で割った値によって、キューの長さの平均が示されます。値が1以上の場合は、ワークキューに関連付けられているワー

カーズレッドの最小数を増やすことができます。1未満の値は、キューに関連付けられている作業スレッドの最大数を減らすことができることを示します。

ワークキューは通常、アクティブな状態で開始し、回復不可能な状況 (システムリソースの不足など) が発生するか、非アクティブな状態に遷移するまで続行されます。ランダウンが開始されると、進行状況のランダウン状態に遷移します。

スレッドがスピンドウンされた場合、作業キューのランダウンは完了しません。データ構造を破棄する前に、スレッドの終了を確認する必要があります。RDBSS の作業キューの実装は、スピンドウンする各スレッドがランダウンコンテキストでスレッドオブジェクトへの参照を保存するプロトコルに従います。ランダウン実行スレッド (ワークキューに属していない) は、データ構造を細分化する前に、すべてのスレッドの完了を待機します。

RxDispatchToWorkerThreadおよびRxPostToWorkerThreadキューの現在の実装は、呼び出し元と同じプロセッサ上で動作します。

作業キューのディスパッチに関する次の RDBSS ルーチンには、が含まれます。

RxDispatchToWorkerThread	このルーチンは、ワーカースレッドのコンテキストでルーチンを呼び出します。WORK_QUEUE_ITEM のメモリは、このルーチンによって割り当てられます。
RxPostToWorkerThread	このルーチンは、ワーカースレッドのコンテキストでルーチンを呼び出します。WORK_QUEUE_ITEM のメモリは、呼び出し元によって割り当てられる必要があります。
RxSpinDownMRxDispatcher	このルーチンは、ネットワークミニリダイレクターのディスパッチャーコンテキストを破棄します。 このルーチンは、Windows Server 2003 および Windows XP でのみ使用できます。

診断とデバッグ

2020/04/24 • [Edit Online](#)

RDBSS には、診断およびデバッグのための多くのルーチンが用意されています。これらのルーチンは、次の2つのカテゴリに分類されます。

- アサートルーチンとデバッグルーチン
- 参照と逆参照の追跡ルーチン

これらのルーチンには、次の表に示す項目が含まれます。

ⅢⅢ	Ⅱ
RxAssert	このルーチンは、RDBSS のチェックされたビルドでアサート文字列をカーネルデバッガーに送信します (インストールされている場合)。 RxAssert インクルードファイルを使用すると、このrxAssertルーチンも呼び出されるように Windows カーネルのrtlassert呼び出しが再定義されます。 リテールビルドの場合、このルーチンの呼び出しによってバグチェックが行われます。
RxDbgBreakPoint	このルーチンは、カーネルデバッガーによって処理される例外を発生させます (カーネルデバッガーがインストールされている場合)。それ以外の場合は、デバッグシステムによって処理されます。
Rxp Track ■	このルーチンは、チェックされたビルドで、SRV_CALL、NET_ROOT、V_NET_ROOT、FOB_X、FCB、SRV_OPEN の各構造体を参照する要求を追跡するために使用されます。これらの参照要求のログは、ログ記録システムおよび WMI からアクセスできます。このルーチンは、逆参照操作を実行しません。 リテールビルドの場合、このルーチンは何も行いません。
Rxp Trackreference	このルーチンは、チェックを行うビルドで、SRV_CALL、NET_ROOT、V_NET_ROOT、FOB_X、FCB、SRV_OPEN の各構造体を逆参照する要求を追跡するために使用されます。これらの逆参照要求のログは、ログ記録システムおよび WMI からアクセスできます。このルーチンでは、参照操作は実行されません。 リテールビルドの場合、このルーチンは何も行いません。

前の表に示したルーチンに加えて、これらのルーチンを呼び出す多くのマクロがデバッグ用に定義されています。次の表に記載されているこれらのマクロは、[RxReference](#)または[RxDeference](#)ルーチンのラッパーを提供しています。これらのルーチンは、SRV_呼び出し、Net_ROOT、V_net_root でのファイル構造管理操作に使用されます。FOB_X、FCB、および SRV_開いている構造体。これらのマクロは、対応するRxReferenceまたはRxDeferenceルーチンを呼び出す前に、対応する[Rxp Trackreference](#)または[rxp track逆参照](#)ルーチンを呼び出して診断情報をログに記録します。参照要求と逆参照要求のログには、RDBSS logging システムと WMI からアクセスできます。

Ⅲ	Ⅱ
RxDereferenceAndFinalizeNetFcb (<i>Fcb, RxContext, RecursiveFinalize, forcefinalize</i>)	このマクロは、FCB 構造体での逆参照操作を追跡するために使用されます。 このマクロは参照カウントを操作し、finalize 呼び出しの状態も返すことに注意してください。
RxDereferenceNetFcb (<i>Fcb</i>)	このマクロは、FCB 構造体での逆参照操作を追跡するために使用されます。 このマクロは参照カウントを操作し、最後の逆参照呼び出しの状態も返すことに注意してください。
RxDereferenceNetFobx (<i>fobx, LockHoldingState</i>)	このマクロは、FOBx 構造体での逆参照操作を追跡するために使用されます。
RxDereferenceNetRoot (<i>netroot, LockHoldingState</i>)	このマクロは、NET_ROOT 構造体の逆参照操作を追跡するために使用されます。
RxDereferenceSrvCall (<i>srvcall, LockHoldingState</i>)	このマクロは、SRV_CALL 構造体の逆参照操作を追跡するために使用されます。
RxDereferenceSrvOpen (<i>srvopen, LockHoldingState</i>)	このマクロは、SRV_OPEN 構造体の逆参照操作を追跡するために使用されます。
RxDereferenceVNetRoot (<i>vnetroot, LockHoldingState</i>)	このマクロは、NET_ROOT 構造体の逆参照操作を追跡するために使用されます。
RxReferenceNetFcb (<i>Fcb</i>)	このマクロは、FCB 構造体に対する参照操作を追跡するために使用されます。
RxReferenceNetFobx (<i>fobx</i>)	このマクロは、FOBx 構造体の参照操作を追跡するために使用されます。
RxReferenceNetRoot (<i>netroot</i>)	このマクロは、NET_ROOT 構造体の参照操作を追跡するために使用されます。
RxReferenceSrvCall (<i>srvcall</i>)	このマクロは、DPC レベルではない SRV_CALL 構造体に対する参照操作を追跡するために使用されます。
RxReferenceSrvCallAtDpc (<i>srvcall</i>)	このマクロは、DPC レベルで SRV_CALL 構造体に対する参照操作を追跡するために使用されます。
RxReferenceSrvOpen (<i>srvopen</i>)	このマクロは、SRV_OPEN 構造体の参照操作を追跡するために使用されます。

III	II
RxReferenceVNetRoot (<i>vnetroot</i>)	このマクロは、V_NET_ROOT 構造体の参照操作を追跡するために使用されます。

ログ記録用ルーチンとマクロ

2020/04/24 • [Edit Online](#)

RDBSS には、ログ記録用のルーチンが多数用意されています。これらのログ記録機能は常に存在します。RDBSSLOG マクロが定義されている場合は、チェックされたビルドでのログ呼び出しの生成が有効になります。_RDBSSLOG が設定されていない場合、ログ記録の呼び出しは無効になります。

ログルーチンは、循環バッファに格納されているログレコードを作成します。各レコードは、レコード記述子によって左右に境界されます。このレコード記述子の長さは4バイトです。

次の表には、ログ記録ルーチンが含まれています。

名前	説明
RxLogEventDirect	このルーチンは、i/o エラーログにエラーを記録するために呼び出されます。 このルーチンを直接呼び出すのではなく、 RxLogFailure または RxLogEvent マクロを使用することをお勧めします。
RxLogEventWithAnnotation	このルーチンは、i/o エラーログレコードを割り当て、ログレコードを入力して、このレコードを i/o エラーログに書き込みます。
RxLogEventWithBufferDirect	このルーチンは、i/o エラーログレコードを割り当て、ログレコードを入力して、このレコードを i/o エラーログに書き込みます。このルーチンは、行番号と状態を、i/o エラーログレコードに格納されている生データバッファにエンコードします。 このルーチンを直接呼び出すのではなく、 RxLogFailureWithBuffer マクロを使用することをお勧めします。
_RxLog	このルーチンは、書式指定文字列と可変数のパラメーターを受け取り、ログ記録が有効になっている場合、i/o エラーログエントリとして記録するための出力文字列を書式設定します。 このルーチンを直接呼び出すのではなく、 RxLog マクロを使用することをお勧めします。 このルーチンは、Windows Server 2003、Windows XP、および Windows 2000 の RDBSS のチェックされたビルドでのみ使用できます。

次のマクロは、前の表に示したルーチンを呼び出す rxlog と rxprocs のヘッダーファイルで定義されています。これらのマクロは、通常、これらのルーチンを直接呼び出す代わりに使用されます。

名前	説明

III	II
RxLog (<i>Args</i>)	チェックされたビルドでは、このマクロは_RxLogルーチンを呼び出します。 リテールビルドでは、このマクロは何も行いません。 RxLogへの引数は、ログ記録をオフにする必要がある場合に null 呼び出しに変換できるようにするために、追加のかつこのペアで囲む必要があることに注意してください。
RxLogEvent (<i>_DeviceObject</i> , <i>id</i> , <i>EventId</i> , <i>Status</i>)	このマクロは、 RxLogEventDirect ルーチンを呼び出します。
RxLogFailure (<i>_DeviceObject</i> , <i>id</i> , <i>EventId</i> , <i>Status</i>)	このマクロは、 RxLogEventDirect ルーチンを呼び出します。
RxLogFailureWithBuffer (<i>_DeviceObject</i> , " <i> "</i> ", " <i> "</i> ", " <i>状態</i> ", " <i>バッファの長さ</i> ", " <i>長さなし</i> ")	このマクロは、 RxLogEventWithBufferDirect ルーチンを呼び出します。
RxLogRetail (<i>Args</i>)	チェックされたビルドでは、このマクロは_RxLogルーチンを呼び出します。 リテールビルドでは、このマクロは何も行いません。 RxLogRetailへの引数は、ログ記録をオフにする必要がある場合に null 呼び出しに変換できるようにするために、追加のかつこのペアで囲む必要があることに注意してください。

その他のルーチン

2020/04/24 • [Edit Online](#)

RDBSS には、特定のカテゴリに分類されない多くのユーティリティルーチンが含まれています。

RDBSS その他のルーチンには、次のようなものがあります。

名前	説明
RxFsdDispatch	このルーチンは、i/o 要求パケット (IRP) を処理するための RDBSS のファイルシステムドライバー (FSD) ディスパッチを実装しています。このルーチンは、要求の RDBSS 処理を開始するために、ドライバー ディスパッチルーチンのネットワークミニリダイレクターによって呼び出されます。
RxFsdPostRequest	このルーチンは、RX_CONTEXT 構造体によって指定された IRP を、ファイルシステムプロセス (FSP) による処理のためにワーカー キューにキューします。
RxGetRDBSSProcess	このルーチンは、RDBSS カーネルプロセスによって使用されるメインスレッドのプロセスへのポインターを返します。
RxIsThisACscAgentOpen	このルーチンは、ファイルオープン要求がユーザーモードのクライアント側キャッシュエージェントによって行われたかどうかを判断します。 このルーチンは、Windows Server 2003 でのみ使用できます。
RxMakeLateDeviceAvailable	このルーチンは、デバイスオブジェクトを変更して "遅延デバイス" を使用できるようにします。遅延デバイスとは、ドライバーの読み込みルーチンで作成されたものではありません。
RxPrepareToReparseSymbolicLink	このルーチンは、再解析を容易にするためにファイルオブジェクト名を設定します。このルーチンは、シンボリックリンクをスキャンするためにネットワークミニリダイレクターによって使用されます。このルーチンは、ネットワークミニリダイレクターでは使用しないでください。

RX_コンテキストと IRP 管理

2020/04/24 • [Edit Online](#)

RX_コンテキスト構造は、i/o 要求パケット (IRP) を管理するために RDBSS とネットワークミニダイレクターによって使用される基本的なデータ構造の1つです。RX_コンテキスト構造は、処理中の IRP を記述し、IRP の完了時にグローバルリソースを解放するための状態情報を格納します。RX_CONTEXT データ構造体は、RDBSS、ネットワークミニダイレクター、およびファイルシステムで使用される IRP をカプセル化します。RX_コンテキスト構造体には、単一の IRP へのポインターと、IRP を処理するために必要なすべてのコンテキストが含まれます。

RX_コンテキスト構造は、Windows Driver Kit (WDK) のヘッダーファイルおよびネットワークミニダイレクタードライバの開発に使用されるその他のリソースで、IRP コンテキストまたは RxContext と呼ばれることがあります。

RX_コンテキストは、さまざまなネットワークミニダイレクターによって提供される追加情報が接続されるデータ構造です。設計の観点からは、この追加情報は次のいずれかの方法で処理できます。

- コンテンツを格納するためにネットワークミニダイレクターが使用する、RX_コンテキストの一部としてコンテキストポインターを定義できるようにします。これは、RX_コンテキスト構造が割り当てられ破棄されるたびに、ネットワークミニダイレクタードライバは、追加のネットワークミニダイレクターを含むメモリブロックの割り当てまたは破棄を個別に実行する必要があることを意味します。参照: RX_コンテキスト構造は作成され、大きな数値で破棄されるため、パフォーマンスの観点からは許容できるソリューションではありません。
- もう1つの方法は、各 RX_コンテキスト構造のサイズを、各ネットワークミニダイレクターの事前に指定された量だけ割り当てることで構成されます。これは、ミニダイレクターで使用するために予約されます。このような方法を採用すると、追加の割り当てと破棄が回避されますが、RDBSS の RX_コンテキスト管理コードが複雑になります。
- 3番目の方法では、事前に指定された領域を割り当てます。これは、各 RX_コンテキストの一部として、すべてのネットワークミニダイレクターで同じです。これは、さまざまなネットワークミニダイレクターによって必要な構造を設定できる、フォーマットされていない領域です。このようなアプローチでは、以前のアプローチに関連する欠点を克服できます。これは、RDBSS で現在実装されているアプローチです。

3番目の方法は、RDBSS によって使用されるスキームです。そのため、ネットワークミニダイレクタードライバの開発者は、RX_コンテキストデータ構造で定義されているこの事前指定領域に適合するように、関連付けられているプライベートコンテキストを試して定義する必要があります。この規則に違反するネットワークミニダイレクタードライバは、大幅なパフォーマンスの低下を伴います。

ネットワークミニダイレクターによってエクスポートされる多くの RDBSS ルーチンおよびルーチンは、受信側のスレッドまたはルーチンによって使用されるその他のスレッドで、RX_のコンテキスト構造を参照します。したがって、RX_のコンテキスト構造は参照カウントされ、非同期操作の使用を管理します。参照カウントがゼロになると、RX_コンテキスト構造を終了し、最後の逆参照操作で解放できます。

RDBSS は、RX_コンテキスト構造および関連する IRP を操作するために使用される多くのルーチンを提供します。これらのルーチンは、RX_コンテキスト構造の割り当て、初期化、および削除に使用されます。これらのルーチンは、RX_コンテキストに関連付けられている IRP を完了し、RX_コンテキストのキャンセルルーチンを設定するためにも使用されます。

次のルーチンは、RX_コンテキスト構造体进行操作します。

名前	説明
RxCompleteRequest	このルーチンは、RX_CONTEXT 構造に関連付けられている IRP を完了するために使用されます。このルーチンは、RDBSS によって内部的に使用されるため、ネットワークミニダイレクターでは使用できません。

名前	説明
RxCompleteRequest_Real	このルーチンは、RX_CONTEXT 構造に関連付けられている IRP を完了するために使用されます。このルーチンは、RDBSS によって内部的に使用されるため、ネットワークミニドライクターでは使用できません。
RxCreateRxContext	このルーチンは、新しい RX_CONTEXT 構造体を割り当て、データ構造体を初期化します。
RxDereferenceAndDeleteRxContext_Real	このルーチンは、RX_CONTEXT 構造体を逆参照し、参照カウントがゼロになると、指定された RX_CONTEXT 構造体を RDBSS のメモリ内データ構造から解放し、削除します。
RxInitializeContext	このルーチンは、新しく割り当てられた RX_CONTEXT 構造体を初期化します。
RxPrepareContextForReuse	このルーチンは、以前に実行されたすべての操作固有の割り当てと買収をリセットすることによって、再利用できるように RX_CONTEXT 構造を準備します。IRP から取得したパラメータは変更されません。このルーチンは、RDBSS によって内部的に使用されるため、ネットワークミニドライクターでは使用できません。
RxResumeBlockedOperations_Serially	このルーチンは、シリアル化されたブロッキング i/o キューで次の待機中のスレッド (存在する場合) をウェイクアップします。
RxSetMinirdrCancelRoutine	ルーチンは、RX_CONTEXT 構造体のネットワークミニドライクターのキャンセルルーチンを設定します。
__RxSynchronizeBlockingOperations	このルーチンは、ブロッキング i/o を同じ作業キューに同期するために使用されます。このルーチンは、名前付きパイプ操作を同期するために RDBSS によって内部的に使用されます。ネットワークミニドライクターはこのルーチンを使用して、ネットワークミニドライクターによって管理されている別のキューで操作を同期します。 このルーチンは、Windows Server 2003 でのみ使用できます。
__RxSynchronizeBlockingOperationsMaybeDroppingFileLock	このルーチンは、ブロッキング i/o を同じ作業キューに同期するために使用されます。このルーチンは、名前付きパイプ操作を同期するために RDBSS によって内部的に使用されます。ネットワークミニドライクターはこのルーチンを使用して、ネットワークミニドライクターによって管理されている別のキューで操作を同期します。 このルーチンは、Windows XP および Windows 2000 でのみ使用できます。

次のマクロは、前の表に示したルーチンを呼び出す *rxcontx* ヘッダーファイルで定義されています。これらのマクロは、通常、これらのルーチンを直接呼び出す代わりに使用されます。

III	II
RxSynchronizeBlockingOperations (<i>RXCONTEXT</i> , <i>FCB</i> <i>, ioqueue</i>)	このマクロは、ブロッキング i/o 要求を同じ作業キューに同期します。Windows Server 2003 では、このマクロは <i>Dropfcblock</i> パラメーターを FALSE に設定して __RxSynchronizeBlockingOperations ルーチン呼び出します。 Windows XP および Windows 2000 では、このマクロは <i>Dropfcblock</i> パラメーターを FALSE に設定して __RxSynchronizeBlockingOperationsMaybeDroppingFcbLock ルーチン呼び出します。
RxSynchronizeBlockingOperations (<i>RXCONTEXT</i> , <i>FCB</i> <i>, ioqueue</i>)	このマクロは、ブロッキング i/o 要求を同じ作業キューに同期します。Windows Server 2003 では、このマクロは <i>Dropfcblock</i> パラメーターを TRUE に設定して __RxSynchronizeBlockingOperations ルーチン呼び出します。 Windows XP および Windows 2000 では、このマクロは <i>Dropfcblock</i> パラメーターを TRUE に設定して __RxSynchronizeBlockingOperationsMaybeDroppingFcbLock ルーチン呼び出します。

接続とファイル構造管理

2020/04/24 • [Edit Online](#)

接続とファイル構造を管理するために RDBSS で使用される基本的なデータ構造は6つあります。これらのデータ構造は、RDBSS とさまざまなネットワークミニダイレクターによって内部的に使用されます。これらのデータ構造には、2つのバージョンがあります。ネットワークミニダイレクターのバージョンには、ネットワークミニダイレクタードライバによって操作できるフィールドが含まれています。これらのデータ構造のネットワークミニダイレクターバージョンは、MRX_ プレフィックスで始まります。RDBSS バージョンには、RDBSS によってのみ操作できる追加フィールドが含まれています。

これらの6つの基本的なデータ構造は次のとおりです。

- SRV_呼び出し--サーバー呼び出しコンテキスト。この構造は、リモートサーバーの抽象化を提供します。
- NET_ROOT--net root。この構造は、共有への接続を抽象化します。
- V_NET_ROOT--net ルートのビュー (仮想 netroots とも呼ばれます)。
- FCB--ファイル制御ブロック。この構造体は、共有上の開いているファイルを表します。
- SRV_開いているサーバー側のコンテキストを開きます。この構造体は、サーバー上の開いているハンドルをカプセル化します。
- FOBX--ファイルオブジェクト拡張子。この構造体は、[ファイル_オブジェクト](#)構造の RDBSS 拡張です。

これらのデータ構造は、次の階層で構成されています。

```
          SRV_CALL
FCB  <-----> NET_ROOT
      SRV_OPEN  <----> V_NET_ROOT
      FOBX
          FILE_OBJECT
```

カーネルファイルシステムの呼び出しに応答して、RDBSS は、通常は FOBX 構造体を除き、前述のすべての構造を使用してネットワークミニダイレクタードライバを作成し、終了します。そのため、ネットワークミニダイレクタードライバは通常、接続とファイル構造の管理に使用されるいくつかの RDBSS ルーチンのみを呼び出します。これらのルーチンのほとんどは、RDBSS によって内部的に呼び出されます。

これらのすべてのデータ構造は参照カウントされます。データ構造の参照カウントは次のとおりです。

SRV_CALL	SRV_CALL を指す NET_ROOT エントリの数と、いくつかの動的な値。
NET_ROOT	NET_ROOT をポイントする FCB エントリと V_NET_ROOT エントリ の数と、いくつかの動的な値。
V_NET_ROOT	V_NET_ROOT を指す SRV_OPEN エントリの数と、いくつかの動的な値。
FCB	FCB を指す SRV_OPEN エントリの数と、いくつかの動的な値。

=====	=====
SRV_OPEN	SRV_OPEN を指す FOBX エントリの数と、いくつかの動的な値。
FOBX	いくつかの動的な値。

どちらの場合も、動的な値は、その構造体を逆参照せずに参照した呼び出し元の数参照します。参照カウンタの静的部分は、ルーチン自体によって管理されます。たとえば、[RxCreateNetRoot](#)は、関連付けられている SRV_呼び出し構造の参照カウンタをインクリメントします。

参照呼び出しと成功した参照は、参照カウンタをインクリメントします。逆参照呼び出しでは、カウンタをデクリメントします。ルーチン呼び出しの作成では、構造体を割り当て、参照カウンタを1に設定します。

任意のデータ構造に関連付けられている参照カウンタは、少なくとも1と、それに関連付けられている次の下位レベルのデータ構造のインスタンス数を加算した値になります。たとえば、2つの NET_ルートが関連付けられている SRV_呼び出しに関連付けられている参照カウンタは、少なくとも3です。RDBSS internal **NameTable**構造体と、次の下位レベルのデータ構造によって保持されている参照に加えて、追加の参照が取得されている可能性があります。

これらの制限により、次のレベルにあるすべてのデータ構造が完了するか、その参照が解放されるまで、特定のレベルのデータ構造を完了 (解放され、関連するメモリブロックが解放される) にすることはできません。たとえば、FCB への参照が保持されている場合、その参照に関連付けられている V_NET_ルート、NET_ROOT、および SRV_呼び出し構造に安全にアクセスできます。

ネットワークミニダイレクターと RDBSS の間のインターフェイスで使用する2つの重要な抽象化は、SRV_呼び出しと NET_ルート構造です。SRV_呼び出し構造は、接続が確立されているサーバーに関連付けられたコンテキストに対応します。また、NET_ルート構造は、サーバー上の共有に対応します (これは、ネットワークミニダイレクターによって要求された名前空間の一部としても表示される可能性があります)。

SRV_呼び出しと NET_ルート構造の作成には、通常、少なくとも1つのネットワークラウンドトリップが必要です。非同期操作を続行するために、これらの操作は2フェーズのアクティビティとしてモデル化されています。SRV_呼び出しと NET_ルート構造を作成するためにネットワークミニダイレクターにコールダウンするたびに、ネットワークミニダイレクターから RDBSS への呼び出しが行われ、要求の完了ステータスが通知されます。現時点では、これらは同期です。

SRV_呼び出し構造の作成は、RDBSS がサーバーとの接続を確立するために多数のネットワークミニダイレクターから選択する必要があるという事実により、さらに複雑になります。RDBSS を展開するネットワークミニダイレクターを選択する際の柔軟性を最大限に高めるために、SRV_呼び出し構造の作成には3番目のフェーズがあります。このフェーズでは、RDBSS がネットワークミニダイレクターに勝者を通知します。失われたすべてのネットワークミニダイレクターは、関連付けられているコンテキストを破棄します。

このセクションでは、次のトピックについて説明します。

[SRV_呼び出しの構造体](#)

[NET_のルート構造](#)

[V_NET_ルート構造](#)

[FCB 構造体](#)

[SRV_オープン構造](#)

[FOBX 構造体](#)

[接続とファイル構造のロック](#)

[接続およびファイル制御ブロックの管理ルーチン](#)

SRV_呼び出し構造

2020/04/24 • [Edit Online](#)

サーバーの呼び出しコンテキストの構造、SRV_を呼び出すと、ネットワークのミニ リダイレクターによって管理される各特定のネットワーク サーバーの接続に関する情報を保持します。

グローバル リスト、SRV の_RDBSS により、呼び出し構造がグローバル データに格納されます。各 SRV_構造体が、SRV に固有の要素と共に、他の RDBSS 構造と共通の要素をいくつかの呼び出し_呼び出し構造体。SRV を管理する RDBSS ルーチン_構造体は、次の要素を変更するだけの呼び出し。

- 署名と参照カウント
- テーブルの名前と関連付けられている情報
- 関連付けられているネットワークの一覧_ルート エントリ
- ネットワークのミニ リダイレクターが RDBSS によって異なる状況で呼び出される必要があるどのくらいの頻度を制御するタイミング パラメーターのセット (アイドル タイムアウトなど)
- 関連付けられているネットワークのミニ リダイレクター ドライバー ID
- どのような追加のストレージは、ネットワークのミニ リダイレクターの要求 (または、SRV の作成者_呼び出しデータ構造体)

SRV の Unicode 名前_呼び出し構造が最後に構造体自体に格納されます。既知の SRV の最後にネットワークのミニ リダイレクターで使用を開始するために予約された余分なスペース_ネットワークミニ リダイレクターはインクルード ファイルからコンテキストのフィールドを使用してこの余分なスペースを単に参照できるように、呼び出しのデータが構造体します。

SRV のファイナライズ_呼び出し構造が 2 つの部分で構成されます。

1. すべてのネットワークとの関連付けを破棄する_ルート
2. メモリの解放

これら 2 つのアクションと、SRV 内のフィールド間に遅延があります_呼び出し構造が複製されてから最初の手順を防止します。

NET_ROOT 構造

2020/04/24 • [Edit Online](#)

NET_ROOT、net ルート構造体には、特定のネットワークの各サーバーの情報が含まれています。ネットワーク ミニリダイレクターによって維持されている接続を共有します。

NET_ROOT はどのような RDBSS とを扱うネットワーク ミニ リダイレクター ドライバーするサーバーではありません。したがって、RDBSS 通常作成し NET_ROOT 構造が開き、呼び出しますネットワーク ミニ リダイレクター ドライバー、サーバーを開く責任を負います。ネットワークのミニ リダイレクター ドライバーは、渡される適切なフィールドに入力が必要です。NET_ROOT 構造にします。

NET_ROOT 構造体のリストは、各 SRV_CALL の RDBSS によって維持されます。各 NET_ROOT 構造体には、NET_ROOT 構造体に固有の要素と共に、他の RDBSS 構造と一般的ないくつかの要素があります。NET_ROOT 構造を管理する RDBSS ルーチンは、次の要素のみを変更します。

- 署名と参照カウント
- テーブルの名前と関連付けられている情報
- 関連付けられている SRV_CALL 構造へのポインターを戻します
- 各種のサブ構造体のサイズ情報
- 関連付けられている FCB の構造体のルックアップ テーブル
- どのような追加のストレージは、ネットワークのミニ リダイレクター (や NET_ROOT データ構造体の作成者) からの要求

NET_ROOT 構造体には、NET_ROOT 遷移 IRP の処理の再開する前に完了するを待機している RX_CONTEXT 構造の一覧も含まれています。これは通常、同時実行の要求は、サーバーに転送するときに発生します。これらの要求の 1 つは、他の要求がキューに置かれたときに開始されます。ネットワークのミニ リダイレクターはインクルード ファイルからコンテキストのフィールドを使用してこの余分なスペースを単に参照できるように既知 NET_ROOT データ構造体の末尾に余分なスペースがネットワーク ミニリダイレクター用に予約を開始します。

NET_ROOT 構造の最終処理では、2 つの部分で構成されます。

1. すべての V_NET_ROOTS との関連付けを破棄します。
2. メモリの解放

これら 2 つのアクション間で遅延が発生できるし、NET_ROOT 構造内のフィールドから重複する最初の手順をできないようにします。

V_NET_ルート構造体

2020/04/24 • [Edit Online](#)

V_NET_ルート構造体は、共有 (たとえば、ユーザー ドライブ マッピングを関連付けられている共有ポイントのルートの下に示す) にマッピングするためのメカニズムを提供します。V_NET_ルート名は、次の形式のいずれかで指定できます。

```
\server\share\d1\d2  
\;m:\server\share\d1\d2
```

名前の形式に依存この V に関連付けられているローカル デバイス (「x:」、たとえば) があるかどうか_NET_ルート構造体。ローカルドライブのマッピングの場合 (d1\例については、d2)、ローカル ドライブ マッピングが各にプレフィックスを取得 [CreateFile](#) この V の開いている_NET_ルート構造体。

V_NET_資格情報を指定するルート構造体にも使用します。このような V の目的は、_NET_ルート構造体は、NET に代替の資格情報を伝達するルートの既定値。このため、必要がありませんいその他の参照。

V の一覧_NET_RDBSS によって各ネットワークのルート構造体が保持される_ルート。各 V_NET_ルート構造が V に固有の要素と共に、他の RDBSS 構造と一般的ないくつかの要素を持つ_NET_ルート構造体。V を管理する RDBSS ルーチン_NET_ルート構造体は、次の要素のみを変更します。

- 署名と参照カウント
- 関連付けられているネットワークへのポインター_ルート構造体とのリンク
- テーブルの検索 (プレフィックス) の名前の情報
- 名前、ユーザーを追加するプレフィックスの名前が表示されます (これは、NET をシミュレートするため_実際 NET のルートにマップされていないルート構造_ルート構造体)

V のファイナライズ_NET_ルート構造は、2 つの部分で構成されます。

1. すべての SRV との関連付けを破棄する_オープン構造体
2. メモリの解放

これら 2 つのアクションと V 内のフィールド間に遅延があります_NET_ルート構造体から重複する最初の手順をできないようにします。

FCB 構造

2020/04/24 • [Edit Online](#)

ファイル制御ブロック (FCB) 構造が指す、*FsContext*フィールドに、ファイル オブジェクト。すべての FCB を共有する操作は、同じファイルを参照してください。残念ながら、2 つの異なる名前には、同じファイルが可能性がありますように、名前は、エイリアスであることに SMB サーバーは今すぐ実装されます。FCB は、ファイル操作の中心点です。同じ FCB での操作は、同じファイルを実際には、ために、同期は、上位レベルのオブジェクトではなく、FCB に基づいています。

作成されるたびに、FCB 構造は、対応する SRV_もオープンと FOBX 構造が作成されます。1 つ以上の SRV_オープン構造体は、指定 FCB 構造体を関連付けることにより、FOBX 構造体の 1 つ以上の特定の SRV に関連付けられて_オープン構造体。ほとんどの場合、1 つの SRV_オープン構造体は、FCB と関連付けられ、FOBX 構造体の数が指定された SRV に関連付けられている_オープン構造は 1 です。空間的局所性と、このような場合、ページング動作を向上させるのに FCB の構造体の割り当ても含まれます割り当て SRV 関連付けられている 1 つの_開き FOBX 構造体。

RDBSS が関連付けられている FCB、SRV の割り当てを試みます_オープン、およびページング動作を向上させるためにメモリ内でまとめて FOBX 構造体。FCB と NET RDBSS は割り当てられません_ルートのため、構造化、NET_ルート構造体はページングされないが FCB の構造体は、通常がページング (ある場合を除いてページング ファイル)。

FCB 構造体は、すべての開いているファイルとディレクトリに対応します。FCB の構造は次の 2 つの部分に分割します。

- 非ページ プールに割り当てられた非ページの一部
- ページの一部

前者は、非_ページ_FCB および、それ以降と呼びます FCB します。

FCB には、対応する非へのポインターが含まれています。_ページ_FCB の一部です。戻りポインターは、非管理_ページ_デバッグのための FCB を FCB チェックビルドします。

非_ページ_FCB にはセクション オブジェクトを操作するメモリ マネージャーとキャッシュ マネージャーで使用される特殊なポインターの構造体が含まれています。これらのポインターの値が、ファイル システムの外部で通常設定することに注意してください。

FCB の構造体には、次のものが含まれています。

- FSRTL_共通_ヘッダー構造
- 署名と参照カウント
- テーブルの名前と関連付けられている情報
- 関連付けられている NET 戻りポインター_ルート構造体
- 一連の関連付けられている SRV_オープン構造体
- デバイス オブジェクト
- ネットワークのミニ リダイレクターまたは FCB 構造体の作成者によって要求された追加のストレージ

SRV_オープン構造体

2020/04/24 • [Edit Online](#)

SRV_オープン構造体は、サーバー上、開いているファイルについて説明します。複数のファイル オブジェクトとファイル オブジェクトの拡張機能 (FOBXs) を共有する同じ SRV_アクセス権が一致する場合、開いている構造体。たとえば、ファイル ID が格納 Smb 向けです。ファイルの Id の一覧は、FCB に関連付けられます。同様に、同じサーバー側のオープン共有するすべてのファイル オブジェクト拡張子にまとめて示します。また、FCB の新しいオープンが、サーバー側の開いているコンテキストを共有するかどうかに関する情報が格納されます。

SRV に影響するフラグ値_オープン操作は、2 つのグループに分割されます。

- ネットワークのミニ リダイレクターに表示されるフラグ
- RDBSS とネットワークのミニ リダイレクターを非表示を内部的に使用されるプライベートフラグ

ネットワークのミニ リダイレクターに表示されるフラグは、可能な SRV の下位 16 ビットで構成されている_フラグ。上位 16 ビットは、RDBSS によって内部的に使用するため予約されています。

SRV_構造体開くにはには、次が含まれています。

- 署名と参照カウント
- FCB 構造体への戻りポインター
- V に戻りポインター_NET_ルート構造体 (通常は)
- FOBX 構造の一覧
- アクセス権と collapsibility 状態
- ネットワークのミニ リダイレクターまたは SRV の作成者によって要求された追加のストレージ_オープン構造体

FOBX 構造

2020/04/24 • [Edit Online](#)

ファイルオブジェクト拡張機能 (FOBX) 構造体は、[ファイル_オブジェクト](#)構造の RDBSS 拡張です。FOBX 構造体は、ファイルオブジェクトの `Fileobjectextension` フィールドによってポイントされます。FOBX 構造体には、次のものが含まれます。

- 署名と参照カウント
- 関連付けられた FCB 構造体への戻りポインター
- 関連付けられた SRV_オープン構造体へのバックポインター
- このオープン構造体に関するコンテキスト情報
- ネットワークミニリダイレクターまたは FOBX 構造の作成者によって要求された追加のストレージ

FOBX 構造体には、ファイルオブジェクトごとに必要なすべての情報が含まれています。これは通常、i/o システムによって格納されるわけではありません。ファイルオブジェクトに関する情報は、固定サイズのファイルシステムオブジェクトの i/o システムによって格納されます。FOBX 構造体は、ネットワークミニリダイレクターによってファイルオブジェクトに必要なその他の情報を処理します。

File オブジェクトの FOBX 構造体は、file オブジェクトの `FsContext2` フィールドによって参照されます。FOBX 構造体は、通常、RDBSS 構造体では無視されますが、FOBX 構造体は現在参照カウントされています。

FOBX フラグは、次の2つのグループに分割されます。

- ネットワークミニリダイレクターに表示されるフラグ
- RDBSS によって内部的に使用され、ネットワークミニリダイレクターからは見えないフラグ

ネットワークミニリダイレクターに表示されるフラグは、可能な FOBX フラグの下位16ビットで構成されます。上位16ビットは、RDBSS によって内部で使用するために予約されています。

接続とファイル構造のロック

2020/04/24 • [Edit Online](#)

ロックのためでは、使用される参照テーブルの 2 つのレベルです。

- SRV のデバイスごとのオブジェクト テーブル_呼び出しと NET_ルート構造体 (プレフィックス テーブル)
- NET ごとのテーブル_FCB 構造 (FCB テーブル) のルート構造体

これらの個別のテーブルが別のネットワーク上のディレクトリ操作を許可する_をほぼ完全に非-妨げられること、接続が確立されると、ルート構造体。同じネットワーク上のディレクトリ操作_ルート構造体がただし干渉、少しの操作を行います。次の表では、特定の操作に必要なロックについて説明します。

II	IIII	IIIIII
作成または最終処理	SRV_CALL NET_ROOT V_NET_ROOT	NetName テーブルに対する排他ロック (RxContext - の TableLock フィールド > RxDeviceObject-> pRxNetNameTable)。
参照を逆参照、または参照	SRV_CALL NET_ROOT V_NET_ROOT	NetName テーブルを共有または排他ロック (RxContext - の TableLock フィールド > RxDeviceObject-> pRxNetNameTable)。
作成または最終処理	FCB SRV_OPEN FOBX	FCB テーブルに対する排他ロック (NET_ROOT - の TableLock フィールド > FcbTable)。
参照を逆参照、または参照	FCB SRV_OPEN FOBX	FCB テーブルを共有または排他ロック (NET_ROOT - の TableLock フィールド > FcbTable)。

注その操作の SRV_現在、オープンと FOBX データ構造体には FCB データ構造の操作に必要な同じロックが必要です。これは、アイデアを保存するメモリだけです。Windows の将来のバージョンは、この制限を削除して、一連の共有リソースは比較的安価なレベルの衝突の可能性を減らすにされる可能性があります FCB レベルで別のリソースを追加することがあります。

行う必要があります、ロック NetName テーブルの最初のロック (FinalizeNetFcb、たとえば)、両方 FCB テーブルをロックしが必要な場合。逆の順序でロックを解除する必要があります。

SRV_呼び出し、NET_ルート、および V_NET_ルートの作成と終了のプロセスは、取得と RDBSS NetName テーブル ロックの解放によって管理されます。

FCB の作成と終了のプロセスは、取得と NET に関連付けられている NetName テーブルのロックの解放によって管理されます_ルート構造体。

FOBX SRVOPEN の作成とファイナライズ プロセスは、取得と FCB テーブルのロックの解放によって制御されます。

次の表は、ロックとロックの作成と、さまざまなデータ構造の最終処理を取得する必要がモードをまとめたものです。

操作	SRV_CALL	NET_ROOT	FCB	SRV_OPEN	FOB_X
作成	NetName テーブルに排他ロック	NetName テーブルに排他ロック	FCB のテーブルに排他ロック	FCB のテーブルに排他ロック	FCB のテーブルに排他ロック
最終処理します。	NetName テーブルに排他ロック	NetName テーブルに排他ロック	FCB のテーブルに排他ロック	FCB のテーブルに排他ロック	FCB のテーブルに排他ロック

参照して、これらのデータ構造体を逆参照も特定の規則に従う必要があります。

参照カウントに関連付けられているデータ構造体のいずれか 1 (ほとんどの場合、名前のテーブルによって保持されている唯一のリファレンス) を削除、ときに、データ構造体、終了処理の潜在的な候補です。データ構造体かを確定できます直ちにまたは清掃を指定できます。これら両方のメソッドは、RDBSS で実装されます。逆参照時にロックの要件が満たされると、データ構造がすぐに完了しました。1 つの例外は、遅延と (たとえば FCB 構造体の逆参照) 操作の最適化を実装します。それ以外の場合、データ構造は、清掃マークされます。

ネットワークのミニリダイレクターは、終了処理ルーチンを呼び出すために NetName テーブルに排他ロックが必要です。

をこれらのデータ構造のいずれかで、作成を実行するには、ネットワークミニリダイレクター ドライバーは、次のようなものを実行する必要があります。

```
getshared();lookup();
if (failed) {
    release(); getexclusive(); lookup();
    if ((failed) { create(); }
}
deref();
release();
```

ロックが正常に取得されるとは、テーブルに、ノードを挿入し、ロックを解放し、サーバーが使用可能なかどうかを参照してください。他の情報を設定し、ブロックを解除するが、同じサーバー上で待機しているすべてのユーザー、サーバーが使用可能な場合は、(、SRV_呼び出しまたは NET_ルート構造体)。

接続とファイル制御ブロック管理ルーチン

2020/04/24 • [Edit Online](#)

接続とファイル制御ブロックの管理ルーチンは、接続とファイル制御ブロックを表すために使用される構造を管理するために RDBSS によって使用されます。

RDBSS は、ネットワークミニダイレクタードライバーで使用できる接続とファイル制御ブロックの管理について、次のルーチンを提供します。

名前	説明
RxCreateNetFcb	このルーチンは、この FCB が開かれている NET_ROOT 構造体のメモリ内データ構造に新しい FCB 構造体を割り当て、初期化し、挿入します。割り当てられた構造体には、SRV_OPEN および FOBX 構造体の領域があります。このルーチンは、RDBSS によって内部的に使用されるため、ネットワークミニダイレクターでは使用できません。
RxCreateNetFobx	このルーチンは、新しいファイルオブジェクト拡張 (FOBX) 構造体を割り当て、初期化し、挿入します。ネットワークミニダイレクターは、このルーチンを呼び出して、作成操作が正常に終了した時点で FOBX を作成する必要があります。
RxCreateNetRoot	このルーチンは、NET_ROOT 構造体を表すノードを構築し、関連付けられているデバイスオブジェクトの NET name テーブルに名前を挿入します。このルーチンは、RDBSS によって内部的に使用されるため、ネットワークミニダイレクターでは使用できません。
RxCreateSrvCall	このルーチンは、サーバー呼び出しコンテキストを表すノードを構築し、RDBSS によって管理される net name テーブルに名前を挿入します。このルーチンは、RDBSS によって内部的に使用されるため、ネットワークミニダイレクターでは使用できません。
RxCreateSrvOpen	このルーチンは、RDBSS によって使用されるメモリ内データ構造に新しい SRV_OPEN 構造体を割り当て、初期化し、挿入します。新しい構造体を割り当てる必要がある場合は、FOBX 構造体の領域があります。このルーチンは、RDBSS によって内部的に使用されるため、ネットワークミニダイレクターでは使用できません。
RxCreateVNetRoot	このルーチンは、V_NET_ROOT 構造体を表すノードを構築し、その名前を NET name テーブルに挿入します。このルーチンは、RDBSS によって内部的に使用されるため、ネットワークミニダイレクターでは使用できません。
RxDereference	このルーチンは、RDBSS によって使用されるいくつかの参照カウントデータ構造体のインスタンスの参照カウントをデクリメントします。

名前	説明
RxFinalizeConnection	このルーチンは、共有への接続を削除します。接続で開いたファイルは、指定された force レベルに応じて閉じられます。ネットワークミニダイレクターは、接続の終了を強制するオプションが指定されていない限り、トランスポート接続をパフォーマンス上の理由から開いたままにしておくことができます。
RxFinalizeNetFcb	このルーチンは、指定された FCB 構造体を終了します。呼び出し元には、この FCB に関連付けられている NET_ROOT 構造体の排他ロックが必要です。このルーチンは、RDBSS によって内部的に使用されるため、ネットワークミニダイレクターでは使用できません。
RxFinalizeNetFobx	このルーチンは、指定された FOBX 構造体を終了します。呼び出し元は、この FOBX に関連付けられた FCB に排他ロックを保持している必要があります。このルーチンは、RDBSS によって内部的に使用されるため、ネットワークミニダイレクターでは使用できません。
RxFinalizeNetRoot	このルーチンは、指定された NET_ROOT 構造体を終了します。呼び出し元には、この NET_ROOT 構造体に関連付けられているデバイスオブジェクトのネットテーブル (SRV_CALL 構造体を通じて) に排他ロックが設定されている必要があります。このルーチンは、RDBSS によって内部的に使用されるため、ネットワークミニダイレクターでは使用できません。
RxFinalizeSrvCall	このルーチンは、指定された SRV_CALL 構造体を終了します。呼び出し元には、この SRV_CALL 構造体に関連付けられているデバイスオブジェクトのネットテーブルのロックに対する排他アクセス権が必要です。このルーチンは、RDBSS によって内部的に使用されるため、ネットワークミニダイレクターでは使用できません。
RxFinalizeSrvOpen	このルーチンは、指定された SRV_OPEN 構造体を終了します。このルーチンは、RDBSS によって内部的に使用されるため、ネットワークミニダイレクターでは使用できません。
RxFinalizeVNetRoot	このルーチンは、指定された V_NET_ROOT 構造体を終了します。呼び出し元には、この V_NET_ROOT 構造体に関連付けられているデバイスオブジェクトのネットテーブルのロックに対する排他アクセスが必要です。このルーチンは、RDBSS によって内部的に使用されるため、ネットワークミニダイレクターでは使用できません。
RxFinishFcbInitialization	このルーチンは、ネットワークミニダイレクターによる作成操作が正常に完了した後に、FCB の初期化を完了するために使用されます。
RxForceFinalizeAllVNetRoots	このルーチンは、特定の NET_ROOT 構造体に関連付けられているすべての V_NET_ROOT 構造体を強制的に終了します。呼び出し元には、この V_NET_ROOT 構造体に関連付けられているデバイスオブジェクトのネットテーブルのロックに対する排他アクセスが必要です。このルーチンは、RDBSS によって内部的に使用されるため、ネットワークミニダイレクターでは使用できません。

名前	説明
RxGetFileSizeWithLock	このルーチンは、64ビット値が一貫して読み取られるようにロックを使用して、FCB ヘッダー内のファイルサイズを取得します。
RxInferFileType	このルーチンは、RX_CONTEXT 構造体のフィールドからファイルの種類 (ディレクトリまたはディレクトリ以外) を推論しようとします。
RxLockEnumerator	このルーチンは、FCB のファイルロックを列挙するために、ネットワーク ミニリダイレクターから呼び出されます。
RxpDereferenceAndFinalizeNetFcb	このルーチンは、参照カウントをデクリメントし、FCB を終了します。 このルーチンは、Windows Server 2003 Service Pack 1 (SP1) 以降でのみ使用できます。
RxpDereferenceNetFcb	このルーチンは、FCB の参照カウントをデクリメントします。
RxpReferenceNetFcb	このルーチンは、FCB の参照カウントをインクリメントします。
RxReference	このルーチンは、RDBSS によって使用されるいくつかの参照カウント データ構造体のインスタンスの参照カウントをインクリメントします。
RxSetSrvCallDomainName	このルーチンは、特定のサーバー (SRV_CALL 構造) に関連付けられているドメイン名を設定します。

デバッグのために[RxReference](#)ルーチンと[RxDeference](#)ルーチンのラッパーを提供する多くのマクロも定義されていることに注意してください。これらのマクロの詳細については、「[診断とデバッグ](#)」を参照してください。

FCB リソース同期

2020/04/24 • [Edit Online](#)

ミニリダイレクタードライバーにとって重要な同期リソースは、主に FCB に関連付けられています。ページング i/o リソースと通常のリソースがあります。ページング i/o リソースは、RDBSS によって内部的に管理されます。ミニリダイレクタードライバーからアクセスできる唯一のリソースは通常のリソースです。このリソースには、次の指定されたルーチンを使用してアクセスする必要があります。

名前	説明
RxAcquireExclusiveFcbResourceInMRx	このルーチンは、排他モードで FCB リソースを取得します。このルーチンは、既に取得されている場合、FCB リソースが解放されるまで待機します。このルーチンは、排他的なリソースが取得されるまで制御を返しません。このようなルーチンは、この FCB に関連付けられている RX_CONTEXT 構造体がキャンセルされている場合でも、FCB リソースを取得します。
RxAcquireSharedFcbResourceInMRx	このルーチンは、共有モードで FCB リソースを取得します。このルーチンは、以前に排他的に取得された FCB リソースが解放されるまで待機します。このルーチンは、共有リソースが取得されるまで制御を返しません。このようなルーチンは、この FCB に関連付けられている RX_CONTEXT 構造体がキャンセルされている場合でも、FCB リソースを取得します。
RxAcquireSharedFcbResourceInMRxEx	このルーチンは、共有モードで FCB リソースを取得します。このルーチンは、以前に排他的に取得された FCB リソースが解放されるまで待機します。このルーチンは、共有リソースが取得されるまで制御を返しません。このようなルーチンは、この FCB に関連付けられている RX_CONTEXT 構造体がキャンセルされている場合でも、FCB リソースを取得します。 このルーチンは、Windows Server 2003 Service Pack 1 (SP1) 以降でのみ使用できます。
RxReleaseFcbResourceForThreadInMRx	このルーチンは、RxAcquireSharedFcbResourceInMRxExを使用して以前に取得した FCB リソースを解放します。 このルーチンは、Windows Server 2003 Service Pack 1 以降でのみ使用できます。
RxReleaseFcbResourceInMRx	このルーチンは、 RxAcquireExclusiveFcbResourceInMRx または RxAcquireSharedFcbResourceInMRx を使用して以前に取得した FCB リソースを解放します。

次のマクロは、現在のスレッドが FCB 通常のリソースにアクセスできるかどうかを判断するために、rxprocs ヘッダーファイルで定義されています。

III	II
RxFcbAcquiredShared (<i>RXCONTEXT</i> , <i>FCB</i>)	このマクロは、現在のスレッドが共有モードの通常のリソースにアクセスできるかどうかを確認します。このマクロは、 ExIsResourceAcquiredSharedLite ルーチンを呼び出します。
RxIsFcbAcquiredShared (<i>FCB</i>)	このマクロは、現在のスレッドが共有モードの通常のリソースにアクセスできるかどうかを確認します。このマクロは、 ExIsResourceAcquiredSharedLite ルーチンを呼び出します。
RxIsFcbAcquiredExclusive (<i>FCB</i>)	このマクロは、現在のスレッドが排他モードで通常のリソースにアクセスできるかどうかを確認します。このマクロは、 ExIsResourceAcquiredExclusiveLite ルーチンを呼び出します。
RxIsFcbAcquired (<i>FCB</i>)	このマクロは、現在のスレッドが共有モードまたは排他モードで通常のリソースにアクセスできるかどうかを確認します。このマクロは、 ExIsResourceAcquiredSharedLite および ExIsResourceAcquiredExclusiveLite ルーチンを呼び出します。

バッファ処理状態の制御

2020/04/24 • [Edit Online](#)

RDBSS には、分散キャッシュの一貫性をさまざまなネットワークミニダイレクターと組み合わせて提供するためのメカニズムであるバッファリングマネージャーが用意されています。このサービスは、バッファリング状態の変更要求を処理する RDBSS のバッファリングマネージャーにカプセル化されています。RDBSS のバッファリングマネージャーは、さまざまなネットワークミニダイレクターおよび RDBSS によって生成されたすべての変更バッファリング状態要求に対するアクションを追跡し、開始します。

一般的なネットワークミニダイレクターでのキャッシュの一貫性の実装には、いくつかの一般的なコンポーネントがあります。

- ファイルルーチンを作成して開くための変更。

このパスでは、バッファリング要求の種類が決定され、適切な要求がサーバーに対して行われます。ネットワークミニダイレクターおよび場合によっては、サーバーから戻ると、作成またはオープン呼び出しの結果に基づいて、FCB に関連付けられているバッファリング状態が更新されます。

- サーバーからの変更バッファリング状態通知を処理するための指示コードを受け取るように変更します。

このような要求が検出された場合は、バッファリング状態を調整するローカル機構をトリガーする必要があります。

- RDBSS の一部として実装されるバッファリング状態を変更するための機構。変更バッファリング状態要求では、要求が適用されるオープン構造の SRV_ を識別する必要があります。

SRV_ のオープン構造を識別するために必要な計算作業の量は、ネットワークミニダイレクターのプロトコルと詳細によって異なります。SMB プロトコルでは、便宜的ロック (oplock) によって、キャッシュの一貫性を確保するために必要なインフラストラクチャが提供されます。Windows での SMB プロトコルの実装では、RDBSS によって提供される多重 ID Api が使用されます。サーバーでは、サーバーで開かれたファイルを識別するために使用される多重 ID を選択します。多重 Id は、それらが開かれている NET_ROOT (共有) に関連しています。したがって、すべての変更バッファリング状態要求は、NetRootKey と SrvOpenKey の2つのキーによって識別されます。このキーは、それぞれ適切な NET_ROOT および SRV_OPEN 構造体に変換する必要があります。リソースの取得/解放メカニズムとの統合を強化し、さまざまなネットワークミニダイレクターでのこの作業の重複を回避するために、RDBSS はこのサービスを提供します。

RDBSS で提供されるルーチンは2つあります。これは、SRV_開いている構造体に対するバッファリング状態の変化を示すためのものです。

- 要求を登録するための [RxIndicateChangeOfBufferingState](#)
- SRV_OPEN 構造体をキーと関連付けるための [RxIndicateChangeOfBufferingStateForSrvOpen](#)

キーの関連付けは元に戻すことができないため、関連付けられた SRV_オープン構造の有効期間が終了します。

ネットワークミニダイレクターは、多重 Id から SRV_ のオープン構造へのマッピングを確立するための補助機構を必要としますが、[RxIndicateChangeOfBufferingState](#)を使用できますが、これは不要なネットワークミニダイレクターです。[RxIndicateChangeOfBufferingStateForSrvOpen](#)を使用することができます。

RDBSS のバッファリングマネージャーは、これらの要求をさまざまな段階で処理します。さまざまな基盤となるネットワークミニダイレクターから受信した要求を、複数のリストのいずれかに保持します。

- ディスパッチャーの一覧には、SRV_OPEN 構造体に対する適切なマッピングが確立されていないすべての要求が含まれます。
- ハンドラーの一覧には、適切なマッピングが確立されているがまだ処理されていないすべての要求が含まれています。
- LastChanceHandlerList には、初期処理が失敗したすべての要求が含まれています。これは通常、変更バッファリング状態要求の受信時に FCB が共有モードで取得された場合に発生します。このような場合、oplock 解除要求は遅延ワー

カーズレッドでのみ処理できます。

ネットワークミニリダイレクタードライバーでの変更バッファリング状態要求の処理は、FCB の取得と解放のプロトコルに関係します。これにより、ターンアラウンド時間が短縮されます。

RDBSS には、ネットワークミニリダイレクタードライバーで使えるバッファリング状態を管理するための次のルーチンが用意されています。

RxChangeBufferingState	このルーチンは、バッファリング状態の変更要求を処理するために呼び出されます。
RxIndicateChangeOfBufferingState	このルーチンは、後で処理するために、バッファリング状態の変更要求 (oplock の中断を示すなど) を登録するために呼び出されます。
RxIndicateChangeOfBufferingStateForSrvOpen	このルーチンは、後で処理するために、バッファリング状態の変更要求 (oplock の中断を示すなど) を登録するために呼び出されます。

I/O ルーチンが不足しています

2020/04/24 • [Edit Online](#)

低 i/o ルーチンは、ファイルオブジェクトに対する基本的な IRP_MJ_XXX 非同期操作 (オープン、閉じる、読み取り、書き込みなど) を表します。RDBSS には、ネットワークミニリダイレクターによる低 i/o 操作で使用される便利なルーチンがいくつか用意されています。RDBSS 低 i/o ルーチンには、次のものが含まれます。

名前	説明
RxLowIoCompletion	このルーチンは、ルーチンが最初に pending として返された場合に、処理が完了すると、ネットワークミニリダイレクタードライバーの低 i/o ルーチンによって呼び出される必要があります。
RxLowIoGetBufferAddress	このルーチンは、RX_CONTEXT 構造体の <code>LowIocontext</code> 構造体から、MDL に対応するバッファを返します。
RxMapSystemBuffer	このルーチンは、i/o 要求パケット (IRP) からのシステムバッファアドレスを返します。
RxNewMapUserBuffer	このルーチンは、低 i/o に使用されたユーザーバッファのアドレスを返します。このルーチンは、Windows XP と Windows 2000 でのみ使用できます。

名前キャッシュ管理

2020/04/24 • [Edit Online](#)

キャッシュ構造_名前を指定すると、サーバーで実行された最近の操作の名前文字列がキャッシュされるため、クライアントは冗長な要求を抑制できます。たとえば、開いている要求が最近失敗し、"ファイルが見つかりません"というメッセージが表示され、クライアントアプリケーションが大文字の文字列を使用して開いている要求を再度試行し、ネットワークミニダイレクターが大文字と小文字を区別した名前をサポートしていない場合、RDBSS は要求を失敗させることができます。サーバーに到達することなくすぐに使用できます。

一般に、このアルゴリズムでは、名前_キャッシュエントリに時間枠と操作数の制限を設定します。通常、時間枠は2秒です。そのため、NAME_CACHE entry が2秒を超える場合、一致は失敗し、要求はサーバーに送られます。サーバーで要求が再度失敗した場合は、_キャッシュエントリの名前が、別の2秒のウィンドウで更新されます。要求操作数が一致しない場合は、1つ以上の要求がサーバーに送信されています。これにより、この名前_キャッシュエントリが無効になる可能性があります。ここでも、この操作はサーバーに送信されます。

キャッシュ構造_名前には、ネットワークミニダイレクター、MRX_NAME_CACHE に公開されている部分と、RDBSS でのみ使用するプライベートセクションがあります。ミニダイレクターの部分には、この名前エントリに対する以前のサーバー操作の結果、および名前_キャッシュと共同で割り当てることができる追加のミニダイレクター固有のストレージ用のコンテキスト拡張機能ポインターのコンテキストフィールド NTSTATUS があります。データ 詳細については、「[RxNameCacheInitialize](#)」を参照してください。

Windows ネットワークの場合、SMB 操作数はミニダイレクター固有の状態の例です。これは、MRX_NAME_CACHE のコンテキストフィールドに保存できます。[RxNameCacheCheckEntry](#)が呼び出されると、名前キャッシュ内の一致の検索の一部として、コンテキストフィールドと指定されたパラメーターの間で等価性のチェックが実行されます。名前_キャッシュエントリが作成または更新されると、このフィールドに適切な値を指定するためのネットワークミニダイレクターのジョブと、NAME_CACHE エントリの有効期間(秒単位) が示されます。

名前_キャッシュ構造体の private RDBSS 部分には、Unicode 文字列としての名前、高速参照を行う名前のハッシュ値、エントリの有効期限、およびサーバーが大文字と小文字を区別する名前をサポートするかどうかを示すフラグが含まれています。

_制御構造の名前_キャッシュは、特定の名前キャッシュを管理します。これには、空きリスト、アクティブリスト、および更新を同期するためのロックがあります。_制御構造の名前_キャッシュには、現在割り当てられている名前_キャッシュエントリの数、割り当てられるエントリの最大数の値、使用される追加のネットワークミニダイレクターストレージのサイズを格納するフィールドもあります。各名前_キャッシュエントリ、統計の値(キャッシュが更新された回数、確認された、有効な一致が返された回数、およびネットワークミニダイレクターがネットワーク操作を保存した時刻)。Maximumentriesフィールドは、不適切なプログラムが大量のメモリを消費している不適切なファイル名を持つ多数のオープン要求を生成する場合に、作成されるキャッシュエントリ_名前の数を制限します。

現在、オブジェクト_名に対して RDBSS によって管理されている名前キャッシュが見つかりません__。この名前キャッシュでは、2秒のウィンドウが保持されます。これは、いずれかの操作がサーバーに送信されると無効になります。これは、サーバー上のアプリケーションがサーバー上で別のファイル(サンプル 2)の作成を通知するために使用するファイル(サンプル 1)が開かれている場合に発生する可能性があります。クライアントが最初のファイル(サンプル 1)を読み取って、2番目のファイル(サンプル 2)がサーバーに作成されたことを認識すると、2番目のファイル(サンプル 2)に一致する名前キャッシュのヒットがエラーを返すことはできません。この最適化によって処理されるのは、まだ存在しない同じファイルに対して、連続したファイルのオープン操作が行われた場合のみです。このシナリオは、Microsoft Word を使用して行われます。

RDBSS name キャッシュ管理ルーチンには、次のものが含まれます。

||||

||

名前	概要
RxNameCacheActivateEntry	このルーチンは、名前キャッシュエントリを受け取り、有効期限とネットワークミニダイレクターコンテキストを更新します。次に、アクティブなリストにエントリを配置します。
RxNameCacheCheckEntry	このルーチンは、NAME_CACHE エントリの有効性を確認します。
RxNameCacheCreateEntry	このルーチンは、指定された名前の文字列を使用して NAME_CACHE 構造体を割り当て、初期化します。呼び出し元は、name cache コンテキストの追加のネットワークミニダイレクター要素を初期化し、そのエントリを name cache active list に配置する必要があります。
RxNameCacheExpireEntry	このルーチンは、NAME_CACHE エントリをフリーリストに挿入します。
RxNameCacheExpireEntryWithShortName	このルーチンは、指定された短いファイル名と一致する名前プレフィックスを持つすべての NAME_CACHE エントリの有効期限が切れます。
RxNameCacheFetchEntry	このルーチンは、NAME_CACHE エントリに対して指定された名前の文字列との一致を検索します。
RxNameCacheFinalize	このルーチンは、NAME_CACHE_CONTROL 構造体に関連付けられているすべての NAME_CACHE エントリのストレージを解放します。
RxNameCacheFreeEntry	このルーチンは、NAME_CACHE エントリのストレージを解放し、NAME_CACHE_CONTROL 構造体に関連付けられている NAME_CACHE キャッシュエントリの数を減らします。
RxNameCacheInitialize	このルーチンは、名前キャッシュ (NAME_CACHE_CONTROL 構造体) を初期化します。

プレフィックステーブル管理

2020/04/24 • [Edit Online](#)

RDBSS では、プレフィックステーブルの使用を有効にするデータ構造を定義しています。これは、SRV_呼び出し、NET_ROOT、および V_NET_ のルート名に使用できるようにします。

RDBSS での name management の現在の実装では、次のコンポーネントを含むテーブルが使用されます。

- 挿入された名前のキュー
- バージョンスタンプ
- テーブルへのアクセスを制御するテーブルロックリソース
- 名前の一致で大文字と小文字を区別するかどうかを示す値
- このプレフィックステーブルのハッシュ値エントリのバケット

テーブルロックリソースは通常の方法で使用されます。これは、参照操作のための共有で、変更操作専用です。

各変更に合わせてバージョンスタンプが変更されます。キューの理由は、プレフィックステーブルパッケージによって、複数の呼び出し元が一度に列挙できることです。挿入された名前とバージョンスタンプのキューを使用すると、複数の呼び出し元を同時に列挙できます。キューはファイル名の高速な参照として使用できますが、プレフィックステーブルは、NET_ のルート構造の正しいアプローチであることが確実です。

これらのプレフィックステーブル管理ルーチンは、MUP からの呼び出しに応答して RDBSS によって内部的に使用され、名前を要求したり、NET_ のルート構造の作成パスを形成したりします。これらの RDBSS プレフィックステーブル管理ルーチンは、テーブルにアクセスする前に適切なロックが取得され、作業が完了したときにロックが解放される限り、ネットワークミニリダイレクターでも使用できます。ドライバーによって通常使用されるのは、次のようになります。

- `RxAcquirePrefixTableLockShared` を呼び出して、共有ロックを取得します。
- `RxPrefixTableLookupName` を呼び出して名前を検索します。
- `RxReleasePrefixTableLock` を呼び出して、共有ロックを解除します。

一部のルーチンは、Windows XP およびそれ以前のバージョンの Windows でのみ実装されていることに注意してください。

`RxPrefixTableLookupName` は、すべてのバージョンの Windows に実装されている唯一のプレフィックステーブル管理ルーチンです。

RDBSS prefix テーブル管理ルーチンには、次のものが含まれます。

名前	説明
<code>RxAcquirePrefixTableLockExclusive</code>	このルーチンは、SRV_CALL と NET_ROOT の名前をカタログ化するために使用されるプレフィックステーブルに対して排他ロックを取得します。 このルーチンは、Windows XP および Windows 2000 でのみ使用できます。このルーチンは、RDBSS によって内部的に使用されるため、ネットワークミニリダイレクターでは使用できません。

名前	説明
RxpAcquirePrefixTableLockShared	このルーチンは、SRV_CALL と NET_ROOT の名前のカタログ化に使用されるプレフィックステーブルに対して共有ロックを取得します。 このルーチンは、Windows XP および Windows 2000 でのみ使用できます。このルーチンは、RDBSS によって内部的に使用されるため、ネットワークミニドライクターでは使用できません。
RxPrefixTableLookupName	このルーチンは、SRV_CALL と NET_ROOT の名前をカタログ化するために使用されるプレフィックステーブル内の名前を検索し、基になるポインターから含んでいる構造体に変換します。
RxpReleasePrefixTableLock	このルーチンは、SRV_CALL と NET_ROOT の名前をカタログ化するために使用されるプレフィックステーブルのロックを解放します。 このルーチンは、Windows XP および Windows 2000 でのみ使用できます。このルーチンは、RDBSS によって内部的に使用されるため、ネットワークミニドライクターでは使用できません。

Windows Server 2003 以降では、前の表に示されているルーチン ([RxPrefixTableLookupName](#) を除く) がマクロに置き換えられています。次のマクロは、より少数のパラメーターでプレフィックステーブルルーチンを呼び出すように定義されています。

名前	説明
RxAcquirePrefixTableLockExclusive (テーブル、待機)	このマクロは、変更操作のために、排他モードでプレフィックステーブルロックを取得します。
RxAcquirePrefixTableLockShared (テーブル、待機)	このマクロは、参照操作のために、共有モードでプレフィックステーブルロックを取得します。
RxIsPrefixTableLockAcquired (テーブル)	このマクロは、プレフィックステーブルロックが排他的モードまたは共有モードのいずれかで取得されたかどうかを示します。
RxIsPrefixTableLockExclusive (テーブル)	このマクロは、プレフィックステーブルロックが排他モードで取得されたかどうかを示します。
RxReleasePrefixTableLock (テーブル)	このマクロは、プレフィックステーブルロックを解放します。

削除と清掃の制御

2020/04/24 • [Edit Online](#)

RDBSS には、不要になったときに FOBX 構造を消去して清掃するための多くのルーチンが用意されています。

クリーンアップ時には、ファイルオブジェクトに関連付けられたユーザーハンドルはこれ以上ありません。このような場合、終了とクリーンアップの間の時間枠は、Memory Manager と Cache Manager によって管理される追加の参照によって決まります。RDBSS は、別のスレッドで実行されるスカベンジャープロセスを使用して、不要な FOBX およびその他の構造を清掃し、削除します。

現時点では、SRV_呼び出し、NET_ROOT および V_NET_ルート構造に清掃が実装されています。FCB 清掃は個別に処理されます。FOBX は、常に同期的に完了する必要があります。清掃を終了するために有効にする必要があるデータ構造は、SRV_オープン構造のみです。

現在、RDBSS に実装されているスカベンジャープロセスでは、ファイナライズの清掃が必要になるまで、システムリソースは消費されません。清掃を終了するようにマークされる最初のエントリは、そのスカベンジャーに対してタイマー要求がポストされます。現在の実装では、タイマー要求はワンショットタイマー要求としてポストされます。これは、エントリが完了するまでの時間間隔に関して保証がないことを意味します。スカベンジャーアクティブ化メカニズムは、後の段階での微調整の候補となる可能性があります。

RDBSS の削除と清掃のルーチンには、次のものが含まれます。

名前	説明
RxPurgeAllFobxs	このルーチンは、ネットワークミニリダイレクターに関連付けられているすべての FOBX 構造を削除します。
RxPurgeRelatedFobxs	このルーチンは、NET_ROOT 構造体に関連付けられているすべての FOBX 構造体を削除します。
RxScavengeAllFobxs	このルーチンは、特定のネットワークミニリダイレクターデバイスオブジェクトに関連付けられているすべての FOBX 構造を清掃します。
RxScavengeFobxsForNetRoot	このルーチンは、特定の NET_ROOT 構造体に関連付けられているすべての FOBX 構造体を清掃します。

多重 ID 管理

2020/04/24 • [Edit Online](#)

RDBSS は、16ビット値の多重化 ID を定義します。この ID は、ネットワーククライアント (ミニリダイレクター) とサーバーの両方で使用でき、任意の接続で同時にアクティブになっている要求を区別するために使用されます。ネットワークリダイレクターは、MID を、それが使用する任意のコンテキストまたは内部データ構造に関連付けることができます。MIDs が割り当てられて使用されているかどうかにかかわらず、ネットワークリダイレクターのオプションで完全に設定されています。

RDBSS によって定義されている MID は、いくつかの条件を満たすように設計された、1つの_アトラスデータ構造の一部です。_アトラスデータ構造に関連付けられている1つ以上の MID_マップデータ構造体は、関連付けられたコンテキストに MIDs をマップするために使用されます。

_アトラスデータ構造体、MID_マップ構造体、および MIDs は、さまざまなリモートサーバーのさまざまな機能进行处理するために適切にスケーリングする必要があります。たとえば、Windows の一般的な LAN Manager サーバーでは、任意の接続で50の未処理の要求を許可しています。サーバーの種類によっては、未処理の要求が1つしかサポートされない場合がありますが、ゲートウェイサーバーではこの数が非常に高くなることが求められる場合があります (多数の未処理の接続の順序で)。

適切に処理する必要がある2つの主な操作は次のとおりです。

- MID をそれに関連付けられているコンテキストにマップする。このルーチンは、クライアントとサーバーの両方で、任意の接続で受信したすべてのパケット进行处理するために呼び出されます (サーバーで MIDs が使用されていることを前提としています)。
- サーバーに要求を送信するための新しい MID を生成しています。このルーチンは、最大接続数の制限を適用するためだけでなく、それぞれの同時要求に一意の ID をタグ付けするために、クライアントで使用されます。

MID は、65536値の可能な組み合わせから、一意のタグ付けと多数の MIDs (通常は 50) の識別を効率的に管理する必要があります。場合によっては、小さな MID_アトラス構造を作成して、MID_マップ構造によって使用されるカーネルメモリを節約し、必要に応じて MID_アトラス構造のサイズを拡大して、より多くの使用量を効率的に処理することが理にかなっています。適切な時間領域のトレードオフを保証するために、参照は3レベルの階層として編成されます。MID を表すために使用される16ビットは、3つのビットフィールドに分割されます。右端のフィールドの長さ (最も重要ではない) は、最初のアトラスで許可されている MIDs の最大数によって決定されます。この最大値は、_アトラスデータ構造体の最初の作成時に [RxCreateMidAtlas](#) ルーチンに渡されるパラメーターです。この最大値によって、作成される MID_アトラスデータ構造の初期サイズと、対応可能な MID_マップデータ構造の数が決まります。残りの長さは、次の2つのフィールド間で均等に分割されます。これにより、既存のミッド_を拡大して拡張するために定義できるアトラス構造_アトラス構造の最大サイズが中間の3レベルの階層になります。_マップのデータ構造体。そのため、各 MID_アトラスデータ構造体には、_マップ構造体の最大数、または1つの下位中央_アトラスと中間_マップ構造体へのポインターを含めることができます。

たとえば、作成時に最大 50 MIDs が割り当てられている場合、最初のフィールドの長さは 6 (64 (2 ** 6) が50を超える) になります。残りの長さは、2番目と3番目の階層レベルでそれぞれ5ビットの2つのフィールドに分割されます。そのため、既存の MID_アトラスデータ構造を拡張して、より多くの MID_マップエントリに対応することができます。

RDBSS は、_アトラスデータ構造体、関連する_マップデータ構造体、および多重 Id を作成して操作するための次のルーチンを提供します。

IIII	II
RxAssociateContextWithMid	このルーチンは、指定された不透明なコンテキストを、MID_ATLAS 構造体の使用可能な中間に関連付けます。

名前	説明
RxCreateMidAtlas	このルーチンは、MID_ATLAS データ構造体の新しいインスタンスを割り当て、それを初期化します。
RxDestroyMidAtlas	このルーチンは、MID_ATLAS データ構造体の既存のインスタンスを破棄し、割り当てられたメモリを解放します。副作用として、MID_ATLAS 構造体のすべての有効なコンテキストで、渡されたコンテキストデストラクターを呼び出します。
RxMapMidToContext	このルーチンは、MID を MID_ATLAS 構造体内の関連付けられたコンテキストにマップします。
RxMapAndDissociateMidFromContext	このルーチンは、MID を MID_ATLAS 構造体内の関連付けられたコンテキストにマップし、その中間とコンテキストとの関連付けを解除します。
RxReassociateMid	このルーチンは、MID を別のコンテキストに再関連付けします。

接続エンジン管理

2020/04/24 • [Edit Online](#)

RDBSS では、接続エンジンは、可能な限り TDI 仕様をマップおよびエミュレートするように設計されています。これにより、ネットワークミニダイレクターで使用するために、基になる TDI 実装を完全に悪用する効率的なメカニズムが提供されます。

RDBSS 接続エンジンは、TDI を抽象化しますが、ネットワークリダイレクターは、これらの RDBSS 接続エンジンルーチンを使用する代わりに、TDI と直接通信することもできます。TDI のラッパーを提供する既存の RDBSS 接続エンジンルーチンは、Microsoft ネットワークをサポートするために開発されたものであるため、Windows 中心であり、他のネットワークディレクターには適していない可能性があります。また、RDBSS の接続エンジンルーチンは、Windows Server 2003 以降にリリースされた Windows オペレーティングシステムから削除される予定です。今後、各ネットワークリダイレクターは、必要な接続エンジンルーチン (TDI またはその他のトランスポート) の開発を担当します。たとえば、WebDAV リダイレクターは、TDI ではなく HTTP パケット (標準 TCP/IP) を送信するために、一部のユーザーモードリフレクタープロセスと通信できます。

RDBSS 接続エンジンルーチンは、次のエンティティに対処します。

- プロトコル
- トランスポートアドレス
- トランスポート接続
- 接続上の仮想回線

トランスポートは、システム上のさまざまなトランスポートサービスプロバイダーへのバインドです。トランスポートアドレスは、ローカル接続エンドポイントです。接続は、エンドポイント間のトランスポート接続です。各接続は、多数の仮想回線 (通常は1つ) をカプセル化します。

次の重要なデータ構造は、RDBSS に関連付けられたさまざまな接続エンジンルーチンによって作成および操作されます。

- RXCE_TRANSPORT--トランスポートのすべてのパラメーターをカプセル化します。
- RXCE_ADDRESS--トランスポートアドレスのすべてのパラメーターをカプセル化します。
- RXCE_接続--トランスポート接続のすべてのパラメーターをカプセル化します。
- RXCE_VC--トランスポート接続上の仮想回線のすべてのパラメーターをカプセル化します。

ネットワークミニダイレクタードライバでは、これらのデータ構造を使用して、各型に対して提供されるルーチン呼び出して、接続エンジンの部分を構築および破棄することができます。これらのルーチンは、これらの構造体に関連付けられているメモリの割り当てや解放を行いません。これにより、このような接続エンジンのデータ構造のインスタンスを管理するための柔軟な機能がミニダイレクタードライバに提供されます。

上記で説明した4種類の接続エンジンは、各データ構造体の先頭に、検証に RDBSS が広く使用する特殊な RXCE_署名署名を使用してタグ付けされています。

RDBSS には、ネットワークミニダイレクタードライバで利用できる次の接続エンジンルーチンが用意されています。

RxCAllocateIrpWithMDL	このルーチンは、接続エンジンによって使用される IRP を割り当て、MDL を IRP に関連付けます。 このルーチンは、Windows XP でのみ使用できます。

名前	説明
RxCeBuildAddress	このルーチンは、トランスポートアドレスをトランスポートバインドに関連付けます。
RxCeBuildConnection	このルーチンは、ローカルの RDBSS 接続アドレスと指定されたリモートアドレスとの間の接続を確立します。このルーチンは、システムワーカーサレッドのコンテキストで呼び出す必要があります。
RxCeBuildConnectionOverMultipleTransports	このルーチンは、ローカルの RDBSS 接続アドレスと指定されたリモートアドレスとの間の接続を確立し、複数のトランスポートをサポートします。ローカルアドレスのセットが指定され、このルーチンは、ローカルアドレスに関連付けられているすべてのトランスポートを介して対象サーバーへの接続を試みます。接続オプションに応じて、1つの接続が勝者として選択されます。このルーチンは、システムワーカーサレッドのコンテキストで呼び出す必要があります。
RxCeBuildTransport	このルーチンは、指定されたトランスポート名に RDBSS transport をバインドします。
RxCeBuildVC	このルーチンは、指定された接続に仮想回線を追加します。
RxCeCancelConnectRequest	このルーチンは、以前に発行された接続要求をキャンセルします。 このルーチンは現在実装されていないことに注意してください。
RxCeFreeIrp	このルーチンは、接続エンジンによって使用される IRP を解放します。 このルーチンは、Windows XP でのみ使用できます。
RxCeInitiateVCDDisconnect	このルーチンは、仮想回線で切断を開始します。このルーチンは、システムワーカーサレッドのコンテキストで呼び出す必要があります。
RxCeQueryAdapterStatus	このルーチンは、指定されたトランスポートの ADAPTER_STATUS 構造体を返します。
RxCeQueryInformation	このルーチンは、接続に関連する情報を照会します。
RxCeQueryTransportInformation	このルーチンは、指定されたトランスポートの接続数とサービスの品質に関するトランスポート情報を返します。
RxCeSend	このルーチンは、仮想回線上の指定された接続に沿って TSDU を送信します。
RxCeSendDatagram	このルーチンは、指定されたトランスポートアドレスに TSDU を送信します。

名前	説明
RxCeTearDownAddress	このルーチンは、トランスポートバインドからトランスポートアドレスを削除します。
RxCeTearDownConnection	このルーチンは、特定の接続を破棄します。
RxCeTearDownTransport	このルーチンは、指定されたトランスポートからバインドを解除します。
RxCeTearDownVC	このルーチンは、仮想接続を破棄します。

Windows Vista 以降では、Microsoft Windows のバージョンでは TDI はサポートされません。代わりに、[Windows フィルタリングプラットフォーム](#)または[Winsock カーネル](#)を使用してください。

カーネル ネットワーク ミニリダイレクター ドライバー

2020/04/24 • [Edit Online](#)

カーネルネットワークミニリダイレクタードライバーは、リダイレクトされたドライババッファリングサブシステム (RDBSS) がドライバーと通信するために使用する多数のコールバックルーチンを実装します。このドキュメントの残りの部分では、カーネルネットワークミニリダイレクタードライバーをネットワークミニリダイレクタードライバーと呼びます。

ネットワークミニリダイレクタードライバーが最初に起動したとき (そのDriverentryルーチン内)、ドライバーは RDBSS [RxRegisterMinirdr](#) ルーチン呼び出して、ネットワークミニリダイレクタードライバーを RDBSS に登録します。ネットワークミニリダイレクタードライバーは、MINIRDR_ディスパッチ構造体を渡します。これには、構成データと、ネットワークミニリダイレクタードライバーによって実装されるルーチンへのポインター (ディスパッチテーブル) が含まれます。

ネットワークミニリダイレクターは、これらのルーチンの一部のみを実装することを選択できます。ネットワークミニリダイレクターによって実装されていないルーチンは、[RxRegisterMinirdr](#)に渡される minirdr_ディスパッチ構造体内のNULLポインターに設定する必要があります。RDBSS は、ネットワークミニリダイレクターによって実装されたルーチンのみを呼び出します。

ネットワークミニリダイレクターによって実装されるルーチンの特別なカテゴリの1つに、読み取り、書き込み、およびその他のファイル操作のための従来のファイル i/o 呼び出しを表す低 i/o 操作があります。すべての低 i/o ルーチンは、RDBSS によって非同期的に呼び出すことができます。ネットワークミニリダイレクター用のカーネルドライバーは、実装されているすべての低 i/o ルーチンを安全に非同期的に呼び出すことができるようにする必要があります。Low i/o ルーチンは、[Driverentry](#) ルーチンからの minirdr_ディスパッチ構造体の一部として、ルーチンポインターの配列として渡されます。配列エントリの値は、実行する低 i/o 操作です。すべての低 i/o ルーチンは、RX_コンテキスト構造体へのポインターをパラメーターとして渡すことを想定しています。RX_コンテキストデータ構造体には、実行する低 i/o 操作を指定するLowiocontext (操作) メンバーがあります。このLowiocontextは、要求された低 i/o 操作を指定するために使用できる、低 i/o ルーチンのいくつか、ネットワークミニリダイレクタードライバーで同じルーチンを指している可能性があります。たとえば、ファイルロックに関連するすべての低 i/o 呼び出しで、ネットワークミニリダイレクターで同じ低 i/o ルーチンを呼び出すことができます。また、このルーチンでは、要求されたロックまたはロック解除操作を指定するためにLowiocontext. operationメンバーを使用できます。

また、RDBSS は、ネットワークミニリダイレクターによって実装される他のいくつかのルーチンの非同期操作も想定しています。これらのルーチンは、リモートリソースとの接続を確立するために使用されます。接続操作の完了にはかなりの時間がかかるため、RDBSS はこれらのルーチンが非同期操作として実装されていると想定しています。

RDBSS は、低 i/o と接続関連のルーチン以外のネットワークミニリダイレクターによって実装されたすべてのルーチンが、同期呼び出しに基づいていることを前提としています。ただし、これは Windows オペレーティングシステムの将来のリリースで変更される可能性があります。

ネットワークミニリダイレクターによって実装されるすべてのルーチンは、完了時に NTSTATUS 値を返します。ほとんどのルーチンは、正常に完了したか、適切な NTSTATUS 値を_状態を返します。特定のルーチンに固有の戻り値に加えて、ほとんどのルーチンで返すことができるエラーの汎用カテゴリが2つあります。

- ネットワーク エラー
- 認証エラー

次のようなネットワークエラーが考えられます。

状態_IO_タイムアウト

リモートサーバーへの i/o 要求がタイムアウトしました。

状態_無効_ネットワーク_パス

I/O 要求が、存在しないネットワークパスに対して行われました。このエラーは、ディレクトリの名前が変更されたか、削除された場

合に発生する可能性があります。

ネットワーク_到達できない状態_

クライアントからネットワークに到達できません。

リモート_リスンしていない状態_

リモートサーバーが接続をリスンしていません。

_ユーザー_セッション_削除された状態

サーバー上のユーザーセッションが削除されました。セッションがタイムアウトしたか、サーバーが再起動されたため、既存のすべてのユーザーセッションが削除されたか、またはサーバーの管理者がユーザーセッションを強制的に削除した可能性があります。

状態_接続_切断

リモートサーバーへの接続が切断されました。

状態_ネットワーク_名_削除されました

I/O 要求は、削除されたネットワーク名を対象としています。

次のような認証エラーが発生する可能性があります。

ステータス_ログオン_失敗

リモートサーバーへのログイン要求が失敗しました。

状態_ネットワーク_の資格情報_競合

提示されたネットワーク資格情報と競合が発生しました。

ステータス_ダウングレード_検出されました

サーバーとの通信にクライアントが使用するネットワークプロトコルの変更がサーバーによって検出され、変更が古いバージョンのプロトコルに変更されました。

ステータス_ログイン_WKSTA_制限

サーバーへのワークステーションログインには制限があります。

次のセクションでは、ネットワークミニリダイレクターで実装できるルーチンの詳細について説明します。

[カーネルネットワークミニリダイレクターによって実装されるルーチン](#)

[RDBSS によって使用されないルーチン](#)

ネットワークミニリダイレクターによって実装されるルーチンは、機能に基づいて次のカテゴリに分類できます。

[接続と名前解決](#)

[ドライバの開始、停止、デバイス制御](#)

[ファイルシステムオブジェクトの作成と削除](#)

[ファイルシステムオブジェクトの i/o ルーチン](#)

[ファイルシステムオブジェクトのクエリと Set ルーチン](#)

[その他のネットワークミニリダイレクタールーチン](#)

カーネル ネットワーク ミニリダイレクターによって実装されるルーチン

2020/04/24 • [Edit Online](#)

次のルーチンは、ネットワークミニリダイレクターで実装できます。

名前	説明
MRxAreFilesAliased	2つのファイル制御ブロック (FCB) 構造体が同じファイルを表している場合、RDBSS はこのルーチンを呼び出してネットワークミニリダイレクターを照会します。RDBSS は、同じように見える2つのファイルを処理するときにこのルーチンを呼び出しますが、名前は異なります (MS-DOS の短い名前と長い名前など)。
MRxCleanupFobx	RDBSS は、このルーチンを呼び出して、ネットワークミニリダイレクターがファイルシステムオブジェクトの拡張構造を閉じるように要求します。RDBSS は、ファイルオブジェクトの IRP_MJ_CLEANUP の受信に応答して、この呼び出しを発行します。
MRxCloseSrvOpen	RDBSS は、このルーチンを呼び出して、ネットワークミニリダイレクターが <code>SRV_OPEN</code> 構造を閉じるように要求します。
MRxCollapseOpen	RDBSS は、このルーチンを呼び出して、ネットワークミニリダイレクターが開いているファイルシステム要求を既存の <code>SRV_OPEN</code> に折りたたんでいることを要求します。
MRxCompleteBufferingStateChangeRequest	RDBSS は、このルーチンを呼び出して、バッファリング状態の変更要求が完了したことをネットワークミニリダイレクターに通知します。たとえば、SMB リダイレクターは、このルーチンを使用して oplock 解除応答を送信するか、またはファイルが使用されなくなった場合に oplock 中断のハンドルを閉じます。サーバーにフラッシュする必要があるバイト範囲ロックは、 <code>RX_CONTEXT</code> 構造体のネットワークミニリダイレクターに、 <code>Lowiocontext.locks.LockList</code> メンバーとして渡されます。
MRxComputeNewBufferingState	RDBSS はこのルーチンを呼び出して、ネットワークミニリダイレクターが新しいバッファリング状態の変化を計算するように要求します。
MRxCreate	RDBSS は、このルーチンを呼び出して、ネットワークミニリダイレクターがファイルシステムオブジェクトを作成するように要求します。RDBSS は、 IRP_MJ_CREATE の受信に応答してこの呼び出しを発行します。
MRxCreateSrvCall	RDBSS は、このルーチンを呼び出して、ネットワークミニリダイレクターが <code>SRV_CALL</code> 構造を作成し、サーバーとの接続を確立するように要求します。

名前	説明
MRxCreateVNetRoot	RDBSS は、このルーチンを呼び出して、ネットワークミニダイレクターが V_NET_ROOT 構造を作成するように要求します。
MRxDeallocateForFcb	RDBSS は、このルーチンを呼び出して、ネットワークミニダイレクターによって FCB が割り当て解除されるように要求します。この呼び出しは、ファイルシステムオブジェクトを閉じる要求に応答しています。
MRxDeallocateForFobx	RDBSS は、このルーチンを呼び出して、ネットワークミニダイレクターに FOBX 構造体の割り当てを解除するよう要求します。この呼び出しは、ファイルシステムオブジェクトを閉じる要求に応答しています。
MRxDevFcbXXXControlFile	RDBSS はこのルーチンを呼び出して、デバイス FCB コントロール要求をネットワークミニダイレクターに渡します。RDBSS は、デバイス FCB で IRP_MJ_DEVICE_CONTROL 、 IRP_MJ_FILE_SYSTEM_CONTROL 、または IRP_MJ_INTERNAL_DEVICE_CONTROL を受信する応答として、この呼び出しを発行します。
MRxExtendForCache	RDBSS は、このルーチンを呼び出して、ファイルがキャッシュマネージャーによってキャッシュされるときに、ネットワークミニダイレクターがファイルを拡張することを要求します。
MRxExtendForNonCache	RDBSS は、このルーチンを呼び出して、ファイルがキャッシュマネージャーによってキャッシュされていない場合に、ネットワークミニダイレクターがファイルを拡張することを要求します。
MRxExtractNetRootName	RDBSS は、このルーチンを呼び出して、ネットワークミニダイレクターが、指定されたパス名から NET_ROOT の名前を抽出するように要求します。
MRxFinalizeNetRoot	RDBSS は、このルーチンを呼び出して、ネットワークミニダイレクターが NET_ROOT オブジェクトを完了するように要求します。
MRxFinalizeSrvCall	RDBSS は、このルーチンを呼び出して、サーバーとの接続を確立するために使用される SRV_CALL 構造の最終処理をネットワークミニダイレクターに要求します。
MRxFinalizeVNetRoot	RDBSS は、このルーチンを呼び出して、ネットワークミニダイレクターが V_NET_ROOT オブジェクトを完了するように要求します。
MRxFlush	RDBSS は、このルーチンを呼び出して、ネットワークミニダイレクターがファイルシステムオブジェクトをフラッシュすることを要求します。RDBSS は、 IRP_MJ_FLUSH_BUFFERS の受信に応答してこの呼び出しを発行します。

名前	説明
MRxForceClosed	RDBSS はこのルーチン呼び出して、ネットワークミニダイレクターが強制的に閉じることを要求します。このルーチンは、SRV_OPEN 構造体の条件が正しくない場合、または SRV_OPEN 構造体が closed とマークされている場合に呼び出されます。
MRxGetConnectionId	RDBSS は、このルーチン呼び出して、ネットワークミニダイレクターが、複数のセッションを処理するために使用できる接続の接続 ID を返すように要求します。ネットワークミニダイレクターで接続 Id がサポートされている場合、返された接続 ID は、name テーブルに格納されている接続構造に追加されます。RDBSS は接続 ID を不透明な blob と見なし、指定された名前の net name テーブルを参照している間に、接続 ID blob をバイト単位で比較します。
MRxIsLockRealizable	RDBSS は、このルーチン呼び出して、特定の NET_ROOT 構造でバイト範囲ロックがサポートされているかどうかをネットワークミニダイレクターが示すことを要求します。
MRxIsValidDirectory	RDBSS は、このルーチン呼び出して、パスが有効なディレクトリであるかどうかをネットワークミニダイレクターが示すことを要求します。
MRxLowIOSubmit [LOWIO_OP_EXCLUSIVELOCK]	RDBSS は、このルーチン呼び出して、ネットワークミニダイレクターがファイルオブジェクトの排他ロックを開くように要求します。 RDBSS は、IRP_MN_LOCK のマイナーコードを持つ IRP_MJ_LOCK_CONTROL の受信にตอบสนองして、irps->■に SL_EXCLUSIVE_LOCK ビットが設定されている場合に、この呼び出しを発行します。
MRxLowIOSubmit [LOWIO_OP_FSCTL]	RDBSS は、このルーチン呼び出して、ファイルシステムコントロール要求をネットワークミニダイレクターに渡します。RDBSS は、 IRP_MJ_FILE_SYSTEM_CONTROL の受信にตอบสนองしてこの呼び出しを発行します。
MRxLowIOSubmit [LOWIO_OP_IOCTL]	RDBSS はこのルーチン呼び出して、ネットワークミニダイレクターに i/o システム制御要求を渡します。RDBSS は、 IRP_MJ_DEVICE_CONTROL または IRP_MJ_INTERNAL_DEVICE_CONTROL の受信にตอบสนองして、この呼び出しを発行します。
MRxLowIOSubmit [LOWIO_OP_NOTIFY_CHANGE_DIRECTORY]	RDBSS は、このルーチン呼び出して、ディレクトリ変更通知操作のためにネットワークミニダイレクターに要求を発行します。RDBSS は、 IRP_MJ_DIRECTORY_CONTROL の受信にตอบสนองしてこの呼び出しを発行します。
MRxLowIOSubmit [LOWIO_OP_READ]	RDBSS は、このルーチン呼び出して、ネットワークミニダイレクターに読み取り要求を発行します。RDBSS は、 IRP_MJ_READ の受信にตอบสนองしてこの呼び出しを発行します。

名前	説明
MRxLowIOSubmit [LOWIO_OP_SHAREDLOCK]	RDBSS は、このルーチン呼び出しで、ネットワークドライクターがファイルオブジェクトに対して共有ロックを開くことを要求します。RDBSS は、IRP_MN_LOCK のマイナーコードを持つ IRP_MJ_LOCK_CONTROL の受信に回答して、irps->■に SL_EXCLUSIVE_LOCK ビットが設定されている場合に、この呼び出しを発行します。
MRxLowIOSubmit [LOWIO_OP_UNLOCK]	RDBSS は、このルーチン呼び出しで、ネットワークミニドライクターがファイルオブジェクトの1つのロックを解除するように要求します。RDBSS は、IRP_MN_UNLOCK_SINGLE のマイナーコードを使用して IRP_MJ_LOCK_CONTROL を受け取ることに回答して、この呼び出しを発行します。
MRxLowIOSubmit [LOWIO_OP_UNLOCK_MULTIPLE]	RDBSS は、このルーチン呼び出しで、ファイルオブジェクトに保持されている複数のロックがネットワークミニドライクターによって削除されるように要求します。RDBSS は、IRP_MN_UNLOCK_ALL または IRP_MN_UNLOCK_ALL_BY_KEY のマイナーコードを使用して IRP_MJ_LOCK_CONTROL を受け取ることに回答して、この呼び出しを発行します。ロックを解除する範囲は、RX_CONTEXT のLowiocontextに指定されています。
MRxLowIOSubmit [LOWIO_OP_WRITE]	RDBSS はこのルーチン呼び出しで、ネットワークミニドライクターに書き込み要求を発行します。RDBSS は、 IRP_MJ_WRITE の受信に回答してこの呼び出しを発行します。
MRxPreparseName	RDBSS はこのルーチン呼び出しで、ネットワークミニドライクターに名前を preparse する機会を与えます。
MRxQueryDirectory	RDBSS は、このルーチン呼び出しで、ネットワークミニドライクターがファイルディレクトリシステムオブジェクトの情報を照会するように要求します。
MRxQueryEaInfo	RDBSS は、このルーチン呼び出しで、ネットワークミニドライクターがファイルシステムオブジェクトの拡張属性情報を照会するように要求します。RDBSS は、 IRP_MJ_QUERY_EA の受信に回答してこの呼び出しを発行します。
MRxQueryFileInfo	RDBSS は、このルーチン呼び出しで、ネットワークミニドライクターがファイルシステムオブジェクトのファイル情報を照会するように要求します。RDBSS は、 IRP_MJ_QUERY_INFORMATION の受信に回答してこの呼び出しを発行します。
MRxQueryQuotaInfo	RDBSS は、このルーチン呼び出しで、ネットワークミニドライクターがファイルシステムオブジェクトのクォータ情報を照会するように要求します。RDBSS は、 IRP_MJ_QUERY_QUOTA の受信に回答してこの呼び出しを発行します。

名前	説明
MRxQuerySdInfo	RDBSS は、このルーチン呼び出して、ネットワークミニダイレクターがファイルシステムオブジェクトのセキュリティ記述子情報を照会するように要求します。RDBSS は、 IRP_MJ_QUERY_SECURITY の受信に回答してこの呼び出しを発行します。
MRxQueryVolumeInfo	RDBSS は、このルーチン呼び出して、ネットワークミニダイレクターがボリューム情報を照会するように要求します。RDBSS は、 IRP_MJ_QUERY_VOLUME_INFORMATION の受信に回答してこの呼び出しを発行します。
MRxSetEaInfo	RDBSS は、このルーチン呼び出して、ネットワークミニダイレクターがファイルシステムオブジェクトに拡張属性情報を設定するように要求します。RDBSS は、 IRP_MJ_SET_EA の受信に回答してこの呼び出しを発行します。
MRxSetFileInfo	RDBSS は、このルーチン呼び出して、ネットワークミニダイレクターがファイルシステムオブジェクトのファイル情報を設定するように要求します。RDBSS は、 IRP_MJ_SET_INFORMATION の受信に回答してこの呼び出しを発行します。
MRxSetFileInfoAtCleanup	RDBSS は、このルーチン呼び出して、クリーンアップ時にネットワークミニダイレクターがファイルシステムオブジェクトにファイル情報を設定するように要求します。RDBSS は、アプリケーションがハンドルを閉じるときに、ファイルオブジェクトが i/o マネージャーによって閉じられる前に、この呼び出しをクリーンアップ中に発行します。
MRxSetQuotaInfo	RDBSS は、このルーチン呼び出して、ネットワークミニダイレクターがファイルシステムオブジェクトのクォータ情報を設定するように要求します。RDBSS は、 IRP_MJ_SET_QUOTA の受信に回答してこの呼び出しを発行します。
MRxSetSdInfo	RDBSS は、このルーチン呼び出して、ネットワークミニダイレクターがファイルシステムオブジェクトのセキュリティ記述子情報を設定するように要求します。RDBSS は、 IRP_MJ_SET_SECURITY の受信に回答してこの呼び出しを発行します。
MRxSetVolumeInfo	RDBSS は、このルーチン呼び出して、ネットワークミニダイレクターがボリューム情報を設定するように要求します。RDBSS は、 IRP_MJ_SET_VOLUME_INFORMATION の受信に回答してこの呼び出しを発行します。
MRxShouldTryToCollapseThisOpen	RDBSS は、このルーチン呼び出して、RDBSS が開いている要求を既存のファイルシステムオブジェクトに対して試行するかどうかを指定します。

名前	説明
MRxSrvCallWinnerNotify	このルーチンは、複数のリダイレクターが要求を満たすことができたことをネットワークミニダイレクターに通知するために、当初は RDBSS によって呼び出されるように設計されました。優先されるネットワークミニダイレクターは、SRV_CALL 構造を作成し、サーバーとの接続を確立する必要があります。 RDBSS の現在の実装では、各ネットワークミニダイレクターは RDBSS の独自のコピーを持っているため、RDBSS 層で競合するネットワークリダイレクターは存在しません。このルーチンは、SRV_CALL 構造体を作成するすべての要求の後に呼び出されます。 同じ汎用名前付け規則 (UNC) 名前空間を処理するために複数のリダイレクターがインストールされている場合、要求を処理するためのリダイレクターは、レジストリに指定されているリダイレクターの順序に基づいて、Multiple UNC Provider (MUP) によって選択されます。
MRxStart	RDBSS は、このルーチンを呼び出してネットワークミニダイレクターを開始します。
MRxStop	RDBSS は、このルーチンを呼び出してネットワークミニダイレクターを停止します。
MRxTruncate	RDBSS は、このルーチンを呼び出して、ネットワークミニダイレクターがファイルシステムオブジェクトを切り捨てることを要求します。
MRxZeroExtend	RDBSS は、このルーチンを呼び出して、ファイルサイズが FCB の有効なデータ長よりも大きい場合に、ファイルがクリーンアップ時にいっぱいになるファイルシステムオブジェクトをネットワークミニダイレクターが拡張するように要求します。

RDBSS が使用しないルーチン

2020/04/24 • [Edit Online](#)

MINIRDR_ディスパッチ構造体に表示されている多くのルーチンが、RDBSS で呼び出されていないか、使用されていません。ネットワークミニリダイレクターは呼び出されないため、これらのルーチンを実装する必要はありません。ネットワークミニリダイレクターは、その **Driverentry** ルーチンから **RxRegisterMinirdr** に渡された minirdr_ディスパッチ構造に含まれるすべてのルーチンに **NULL** ポインターを設定する必要があります。

RDBSS によって使用されていないルーチンの完全な一覧を次に示します。

- MRxCancel
- MRxCancelCreateSrvCall
- MRxClosedFcbTimeOut
- MRxClosedSrvOpenTimeOut
- MRxClosePrintFile
- MRxEnumeratePrintQueue
- MRxLowIOSubmit[LOWIO_OP_CLEAROUT]
- MRxOpenPrintFile
- MRxUpdateNetRootState
- MRxWritePrintFile

接続と名前の解決

2020/04/24 • [Edit Online](#)

ネットワークミニダイレクターは、リモートサーバーへの接続を確立し、いくつかのルーチンを使用して名前解決を処理します。RDBSS は、このプロセスをいくつかの構造に抽象化します。これには、SRV_呼び出し、NET_ROOT、V_NET_ルート構造がカウントされます。ネットワークミニダイレクターの用語では、接続の確立が "ツリー接続" と呼ばれることがよくあります。この接続を確立するには、SRV_呼び出し、ネット_ルート、および V_NET_ルート構造を作成する必要があります。そのため、通常の手順は、ネットワークミニダイレクター **MRxCreateSrvCall** ルーチンの呼び出しの後に、**MRxSrvCallWinnerNotify** ルーチンと **MRxCreateVNetRoot** ルーチンを呼び出すことです。これらの呼び出しは、通常、ユーザーモードのアプリケーションまたはサービスからの要求に応じて発行され、ドライブをマウントします (net use x: \\サーバー\パブリックなど)。これらの呼び出しは、UNC ファイルオブジェクトの要求 (メモ帳 \\server\public\readme.txt など) によって発生することもあります。RDBSS は、これらの両方のケースをネットワークミニダイレクターの内部で処理し、MRxCreateSrvCall シーケンスを開始します。

接続が削除されると、同様の finalize 呼び出しが行われ、SRV_呼び出しと、NET_ルート構造が破棄され、使用されているすべてのメモリが解放されます。ネットワークミニダイレクターに対するこれらの呼び出しには、**MRxFinalizeVNetRoot**、**MRxFinalizeNetRoot**、および **MRxFinalizeSrvCall** が含まれます。

RDBSS の元の設計では、さまざまなネットワークミニダイレクターが RDBSS の1つのコピーを共有していましたが、これは実装されていませんでした。この当初の設計の holdover は、さまざまなネットワークミニダイレクターが、ネットワーク接続 (\\サーバー\共有など) の要求を満たすために競合していることを示しています。**MRxSrvCallWinnerNotify** ルーチンは、複数のリダイレクターが要求を満たすことができた場合に、RDBSS がネットワークミニダイレクターに通知するために使用されます。優先順位の高いネットワークミニダイレクターは、SRV_呼び出しを作成し、サーバーとの接続を確立する必要があります。

Windows Server 2003、Windows XP、および Windows 2000 の RDBSS の実装では、各ネットワークミニダイレクターに独自の RDBSS のコピーがあるため、RDBSS レイヤーに競合するネットワークリダイレクターは存在しません。各ネットワークミニダイレクター (ネットワークプロバイダー) と RDBSS のローカルコピーは、レジストリ設定の順序に基づいて順番に呼び出されます。プロバイダーが照会される順序は、次のレジストリ値によって制御されます。

```
ProviderOrder
```

このレジストリ値は、次のレジストリキーの下にあります。

```
HKLM\CurrentControlSet\Control\NetworkProvider\Order
```

MRxSrvCallWinnerNotify ルーチンは、SRV_呼び出し構造を作成するすべての要求の後に呼び出されます。

次の表に、接続と名前解決操作のためにネットワークミニダイレクターで実装できるルーチンを示します。

名前	説明
MRxCreateSrvCall	RDBSS はこのルーチンを呼び出して、ネットワークミニダイレクターが SRV_CALL 構造を作成し、サーバーとの接続を確立するように要求します。
MRxCreateVNetRoot	RDBSS は、このルーチンを呼び出して、ネットワークミニダイレクターが V_NET_ROOT 構造体を作成するように要求します。

名前	説明
MRxExtractNetRootName	RDBSS は、このルーチンを呼び出して、ネットワークミニダイレクターが、指定されたパス名の NET_ROOT 構造から名前を抽出するように要求します。
MRxFinalizeNetRoot	RDBSS は、このルーチンを呼び出して、ネットワークミニダイレクターが NET_ROOT オブジェクトを完了するように要求します。
MRxFinalizeSrvCall	RDBSS は、このルーチンを呼び出して、サーバーとの接続確立に使用される SRV_CALL 構造をネットワークミニダイレクターが完了するように要求します。
MRxFinalizeVNetRoot	RDBSS は、このルーチンを呼び出して、ネットワークミニダイレクターが V_NET_ROOT 構造を完了するように要求します。
MRxPreparseName	RDBSS はこのルーチンを呼び出して、ネットワークミニダイレクターに名前を preparse する機会を与えます。
MRxSrvCallWinnerNotify	このルーチンは、RDBSS によって呼び出されるように設計されており、複数のリダイレクターが要求を満たすことができるようになったことをネットワークミニダイレクターに通知します。優先されるネットワークミニダイレクターは、SRV_CALL 構造を作成し、サーバーとの接続を確立する必要があります。 RDBSS の現在の実装では、各ネットワークミニダイレクターは RDBSS の独自のコピーを持っているため、RDBSS 層で競合するネットワークリダイレクターは存在しません。このルーチンは、SRV_CALL 構造体を作成するすべての要求の前に呼び出されます。 同じ UNC 名前空間を処理するために複数のリダイレクターがインストールされている場合、要求を処理するリダイレクターは、レジストリに指定されているリダイレクターの順序に基づいて、MUP によって選択されます。

ドライバの開始、停止、デバイス制御

2020/04/24 • [Edit Online](#)

ドライバの登録は、ネットワークミニドライバードライバのDriverentryルーチンで処理されます。ネットワークミニドライバードライバが最初に起動したとき (そのDriverentryルーチン内)、ドライバは RDBSS [RxRegisterMinirdr](#)ルーチン呼び出して、ネットワークミニドライバードライバを RDBSS に登録する必要があります。ネットワークミニドライバードライバは、構成データと、ネットワークミニドライバードライバが実装するルーチンへのルーチンポインタ (ディスパッチテーブル) のテーブルを含む MINIRDR_ディスパッチ構造体を渡します。

ドライバを開始および停止できるようにするには、[MRxStart](#)ルーチンと[MRxStop](#)ルーチンをネットワークミニドライバードライバで実装する必要があります。

ネットワークミニドライバードライバを開始または停止する順序は複雑です。通常、このシーケンスは、管理と管理の目的でドライバを制御するために、ネットワークミニドライバードライバで提供されるユーザーモードアプリケーションまたはサービスによって開始されます。ネットワークミニドライバードライバは、オペレーティングシステムの起動時に自動的に開始するように構成されているサービスを使用できます。このサービスは、オペレーティングシステムが起動するたびにネットワークミニドライバードライバを開始するように要求できます。

[MRxStart](#)は、[RxStartMinirdr](#)ルーチンが呼び出されたときに RDBSS によって呼び出されます。RxStartMinirdrルーチンは、通常、ユーザーモードのアプリケーションまたはサービスからの FSCTL または IOCTL 要求の結果として、ネットワークミニドライバードライバを開始するために呼び出されます。[RxRegisterMinirdr](#)への呼び出しが成功した後、ネットワークミニドライバードライバのdriverentryルーチンからRxStartMinirdrの呼び出しを行うことはできません。一部の開始処理では、ドライバの初期化が完了する必要があるためです。RxStartMinirdr呼び出しが受信されると、RDBSS はネットワークミニドライバードライバのMRxStartルーチン呼び出すことによって開始プロセスを完了します。MRxStartを呼び出すと成功が返された場合、RDBSS は RDBSS 内のミニドライバードライバの内部状態を RDBSS_開始に設定します。

[MRxStop](#)は、[RxStopMinirdr](#)ルーチンが呼び出されたときに RDBSS によって呼び出されます。RDBSS RxStopMinirdrルーチンは、通常、ユーザーモードのアプリケーションまたはサービスからの FSCTL または IOCTL 要求の結果として、ネットワークミニドライバードライバを停止するために呼び出されます。この呼び出しは、ネットワークミニドライバードライバから行うことも、オペレーティングシステムによるシャットダウンプロセスの一部として行うこともできます。RxStopMinirdr呼び出しが受信されると、RDBSS はネットワークミニドライバードライバのMRxStopルーチン呼び出してプロセスを完了します。

[MRxDevFcbXXXControlFile](#)ルーチンは、デバイス FCB で IOCTL または FSCTL 呼び出しを行うことにより、ユーザーモードのアプリケーションまたはサービスからの要求を受信し、ネットワークミニドライバードライバを制御するために使用されます。

また、driver オブジェクトに対して IOCTL および FSCTL 操作を処理する低 i/o ルーチンが2つあります。

[MRxLowIOSubmit\[lowio_OP_FSCTL\]](#) および[MRxLowIOSubmit\[LOWIO_op_IOCTL\]](#)。

ネットワークミニドライバードライバは、これらの低 i/o ルーチンを使用して、ユーザーモードのアプリケーションまたはサービスからネットワークミニドライバードライバの制御と管理を行うこともできます。

次の表に、開始、停止、デバイス制御操作のためにネットワークミニドライバードライバで実装できるルーチンを示します。

MRxDevFcbXXXControlFile	RDBSS はこのルーチン呼び出して、デバイス FCB コントロール要求をネットワークミニドライバードライバに渡します。RDBSS は、デバイス FCB で IRP_MJ_DEVICE_CONTROL、IRP_MJ_FILE_SYSTEM_CONTROL、または IRP_MJ_INTERNAL_DEVICE_CONTROL を受信する応答として、この呼び出しを発行します。

名前	説明
MRxStart	RDBSS は、このルーチンを呼び出してネットワークミニリダイレクターを開始します。
MRxStop	RDBSS は、このルーチンを呼び出してネットワークミニリダイレクターを停止します。

ファイル システム オブジェクトの作成と削除

2020/04/24 • [Edit Online](#)

ネットワークミニダイレクターによって実装されるルーチンの多くは、ファイルの作成と削除を対象としています。RDBSS では、SRV_OPEN、FCB、FOBx 構造体を含む複数の構造体を作成して、このプロセスを抽象化しています。これには参照カウントが含まれます。通常、ファイルシステムオブジェクトを作成または開くには、SRV_OPEN、FCB、および FOBx 構造体を作成する必要があります。通常の手順では、RDBSS がネットワークミニダイレクターによって実装された **MRxCreate** ルーチンを呼び出して、これらの構造体に情報を入力します。また、**MRxShouldTryToCollapseThisOpen** ルーチンと **MRxCollapseOpen** ルーチンを使用して、既存の SRV_オープン構造体の open/create file 要求を折りたたむこともできます。

ネットワークリダイレクターのコンテキストでは、ファイルオブジェクトはファイル制御ブロック (FCB) 構造体とファイルオブジェクト拡張 (FOBx) 構造体を参照します。ファイルオブジェクトと FOBxs の間には1対1の対応があります。多くのファイルオブジェクトは、リモートサーバー上の1つのファイルを表す同じ FCB 構造体を参照します。クライアントは、同じ FCB で複数の異なる open requests (NtCreateFile requests) を持つことができ、これらはそれぞれ新しいファイルオブジェクトを作成します。RDBSS とネットワークミニダイレクターは、受信された NtCreateFile 要求よりも **MRxCreate** 要求を送信することを選択できます。これは、サーバーオープン要求 (SRV_open) を複数の FOBxs で共有する場合に有効です。

RDBSS とネットワークミニダイレクターは、ユーザーがファイルを閉じたときに、必ずしも SRV_開いている構造を閉じるとは限りません。RDBSS は、サーバーとの接続がなくても、開いている構造とデータを SRV_再利用することができます。一部の Windows アプリケーションでは、ファイルを開いて読み取り、閉じ、同じファイルをすばやく開くことができます。このような場合は、SRV_開いている構造体を再利用すると、パフォーマンスが向上します。

これらのデータ構造を閉じて最終処理するために使用されるルーチンがいくつかあります。また、不要になったときに使用されるメモリやその他のリソースを解放します。これらのルーチンには、**MRxCleanupFobx**、**MRxCloseSrvOpen**、**MRxDeallocateForFcb**、**MRxDeallocateForFobx**、および **MRxForceClosed** が含まれます。

次の表に、ファイルシステムオブジェクトの作成および削除操作のためにネットワークミニダイレクターで実装できるルーチンを示します。

名前	説明
MRxAreFilesAliased	2つの FCB オブジェクトが同じファイルを表している場合、RDBSS はこのルーチンを呼び出してネットワークミニダイレクターを照会します。RDBSS は、同じように見える2つのファイルを処理するときにこのルーチンを呼び出しますが、名前は異なります (MS-DOS の短い名前と長い名前など)。
MRxCleanupFobx	RDBSS は、このルーチンを呼び出して、ネットワークミニダイレクターがファイルシステムオブジェクトの拡張を閉じることを要求します。RDBSS は、ファイルオブジェクトに対して IRP_MJ_CLEANUP を受け取ることに応答して、この呼び出しを発行します。
MRxCloseSrvOpen	RDBSS は、このルーチンを呼び出して、ネットワークミニダイレクターが SRV_OPEN オブジェクトを閉じることを要求します。
MRxCollapseOpen	RDBSS は、このルーチンを呼び出して、ネットワークミニダイレクターが開いているファイルシステム要求を既存の SRV_OPEN に折りたたんでいることを要求します。

名前	説明
MRxCreate	RDBSS は、このルーチン呼び出して、ネットワークミニダイレクターがファイルシステムオブジェクトを作成するように要求します。この呼び出しは、IRP_MJ_CREATE の受信にตอบสนองして RDBSS によって発行されます。
MRxDeallocateForFcb	RDBSS は、このルーチン呼び出して、ネットワークミニダイレクターによって FCB が割り当て解除されるように要求します。この呼び出しは、ファイルシステムオブジェクトを閉じる要求にตอบสนองしています。
MRxDeallocateForFobx	RDBSS はこのルーチン呼び出して、ネットワークミニダイレクターに FOBX の割り当てを解除するように要求します。この呼び出しは、ファイルシステムオブジェクトを閉じる要求にตอบสนองしています。
MRxExtendForCache	RDBSS は、このルーチン呼び出して、ファイルがキャッシュマネージャーによってキャッシュされるときに、ネットワークミニダイレクターがファイルを拡張することを要求します。
MRxExtendForNonCache	RDBSS は、このルーチン呼び出して、ファイルがキャッシュマネージャーによってキャッシュされていない場合に、ネットワークミニダイレクターがファイルを拡張することを要求します。
MRxFlush	RDBSS は、このルーチン呼び出して、ネットワークミニダイレクターがファイルシステムオブジェクトをフラッシュすることを要求します。RDBSS は、IRP_MJ_FLUSH_BUFFERS の受信にตอบสนองしてこの呼び出しを発行します。
MRxForceClosed	RDBSS はこのルーチン呼び出して、ネットワークミニダイレクターが強制的に閉じることを要求します。このルーチンは、SRV_OPEN の条件が適切でない場合、または SRV_OPEN が closed とマークされている場合に呼び出されます。
MRxIsLockRealizable	RDBSS は、このルーチン呼び出して、この NET_ROOT でバイト範囲ロックがサポートされているかどうかをネットワークミニダイレクターが示していることを要求します。
MRxShouldTryToCollapseThisOpen	RDBSS は、このルーチン呼び出して、RDBSS が開いている要求を既存のファイルシステムオブジェクトに対して試行するかどうかを指定します。
MRxTruncate	RDBSS は、このルーチン呼び出して、ネットワークミニダイレクターがファイルシステムオブジェクトを切り捨てることを要求します。
MRxZeroExtend	RDBSS は、このルーチン呼び出して、ファイルサイズが FCB の有効なデータ長よりも大きい場合に、ファイルがクリーンアップ時にいっぱいになるファイルシステムオブジェクトをネットワークミニダイレクターが拡張するように要求します。

ファイル システム オブジェクトの I/O ルーチン

2020/04/24 • [Edit Online](#)

ファイルシステムオブジェクトの i/o ルーチンは、読み取り、書き込み、およびその他のファイル操作のための従来のファイル i/o 呼び出しを表します。これらのルーチンは、ネットワークミニダイレクターでは "低 i/o ルーチン" と呼ばれることがよくあります。RDBSS は、特定の IRP の受信に 응답してこれらのルーチンを呼び出します。

低 i/o ルーチンは、スタートアップ時にネットワークミニダイレクターを登録するために使用される [RxRegisterMinirdr](#) ルーチンに渡される `minirdr_` ディスパッチ構造の一部として、ルーチンポインターの配列として渡されます (`driverentry` 内)。ルーチン)。配列エントリの値は、実行する低 i/o 操作です。

これらの低 i/o またはファイルシステムオブジェクトの i/o ルーチンは、通常、RDBSS によって非同期に呼び出されます。そのため、ネットワークミニダイレクターは、実装されている低 i/o ルーチンを安全に非同期的に呼び出すことができるようにする必要があります。ネットワークミニダイレクターは、非同期呼び出しの要求を無視し、同期的にのみ動作するルーチンを実装することもできます。ただし、特定の呼び出しが完了するまでに時間がかかる場合 (読み取りと書き込みなど)、これらを同期操作として実装すると、オペレーティングシステム全体の i/o パフォーマンスが大幅に低下する可能性があります。

すべてのファイルシステムオブジェクト i/o ルーチンは、`RX_` コンテキスト構造体へのポインターをパラメーターとして渡すことを想定しています。これらのルーチンを呼び出す前に、RDBSS は `RX_` コンテキストの `Lowiocontext` 構造に含まれる多数のメンバーの値を設定します。`Lowiocontext` 構造の一部のメンバーはすべてのルーチンに対して設定されますが、一部のメンバーは特定のルーチンに対してのみ設定されます。`RX_CONTEXT` データ構造体には、処理中の IRP が含まれており、実行する i/o 操作の下限を指定する `Lowiocontext` 操作メンバーが含まれています。低 i/o ルーチンのいくつかは、ネットワークミニダイレクターで同じルーチンを指すことができます。これは、この `Lowiocontext` 操作のメンバーが、要求された低 i/o 操作を区別するために使用できるためです。たとえば、ファイルロックに関連するすべての i/o 呼び出しで、ネットワークミニダイレクターで同じ低 i/o ルーチンを呼び出すことができます。これは、要求されたロック操作またはロック解除操作を区別するために、`Lowiocontext. Operation` メンバーを使用します。

低 i/o ルーチン以外の他のファイル i/o ルーチンは、同期呼び出しに基づいていますが、これは Windows オペレーティングシステムの将来のリリースで変更される可能性があります。

`LowIo` パスでは、`RX_` コンテキストの `Lowiocontext threadid` メンバーは、RDBSS で操作を開始した所有スレッドを示すことが保証されます。`Lowiocontext threadid` メンバーを使用すると、別のスレッドに代わって FCB リソースを解放できます。非同期ルーチンが完了すると、初期スレッドから取得された FCB リソースが解放されます。

`MrxLowIoSubmit[LOWIO_OP_XXX]` ルーチンの完了に時間がかかる可能性が高い場合、ネットワークミニダイレクタードライバは、ネットワーク通信を開始する前に、FCB リソースを解放する必要があります。FCB リソースは、[RxReleaseFcbResourceForThreadInMRx](#) を呼び出すことによって解放できます。

次の表に、ファイルシステムオブジェクト i/o (低 i/o) 操作のためにネットワークミニダイレクターで実装できるルーチンを示します。

「「「「	「
MRxLowIoSubmit [LOWIO_OP_EXCLUSIVELOCK]	RDBSS は、このルーチンを呼び出して、ネットワークミニダイレクターがファイルオブジェクトの排他ロックを開くように要求します。 RDBSS は、マイナーコード <code>IRP_MN_LOCK</code> と <code>IrpSp-></code> フラグに <code>SL_EXCLUSIVE_LOCK</code> ビットが設定されている <code>IRP_MJ_LOCK_CONTROL</code> の受信に 응답して、この呼び出しを発行します。

名前	説明
MRxLowIOSubmit [LOWIO_OP_FSCTL]	RDBSS は、このルーチン呼び出して、ファイルシステムコントロール要求をネットワークミニダイレクターに渡します。RDBSS は、IRP_MJ_FILE_SYSTEM_CONTROL の受信に応答してこの呼び出しを発行します。
MRxLowIOSubmit [LOWIO_OP_IOCTL]	RDBSS はこのルーチン呼び出して、ネットワークミニダイレクターに i/o システム制御要求を渡します。RDBSS は、IRP_MJ_DEVICE_CONTROL または IRP_MJ_INTERNAL_DEVICE_CONTROL の受信に応答して、この呼び出しを発行します。
MRxLowIOSubmit [LOWIO_OP_NOTIFY_CHANGE_DIRECTORY]	RDBSS は、このルーチン呼び出して、ディレクトリ変更通知操作のためにネットワークミニダイレクターに要求を発行します。RDBSS は、IRP_MJ_DIRECTORY_CONTROL の受信に応答してこの呼び出しを発行します。
MRxLowIOSubmit [LOWIO_OP_READ]	RDBSS は、このルーチン呼び出して、ネットワークミニダイレクターに読み取り要求を発行します。RDBSS は、IRP_MJ_READ の受信に応答してこの呼び出しを発行します。
MRxLowIOSubmit [LOWIO_OP_SHAREDLOCK]	RDBSS は、このルーチン呼び出して、ネットワークミニダイレクターがファイルオブジェクトに対して共有ロックを開くことを要求します。RDBSS は、マイナーコード IRP_MN_LOCK と IrpSp->フラグに SL_EXCLUSIVE_LOCK ビットが設定されていない IRP_MJ_LOCK_CONTROL を受信する応答として、この呼び出しを発行します。
MRxLowIOSubmit [LOWIO_OP_UNLOCK]	RDBSS は、このルーチン呼び出して、ネットワークミニダイレクターがファイルオブジェクトの1つのロックを解除するように要求します。RDBSS は、IRP_MN_UNLOCK_SINGLE のマイナーコードを使用して IRP_MJ_LOCK_CONTROL を受け取ることに応答して、この呼び出しを発行します。
MRxLowIOSubmit [LOWIO_OP_UNLOCK_MULTIPLE]	RDBSS は、このルーチン呼び出して、ファイルオブジェクトに保持されている複数のロックがネットワークミニダイレクターによって削除されるように要求します。RDBSS は、IRP_MN_UNLOCK_ALL または IRP_MN_UNLOCK_ALL_BY_KEY のマイナーコードを使用して IRP_MJ_LOCK_CONTROL を受け取ることに応答して、この呼び出しを発行します。ロックを解除するバイト範囲は、RX_CONTEXT のLowiocontextに指定されています。
MRxLowIOSubmit [LOWIO_OP_WRITE]	RDBSS はこのルーチン呼び出して、ネットワークミニダイレクターに書き込み要求を発行します。RDBSS は、IRP_MJ_WRITE の受信に応答してこの呼び出しを発行します。

ファイル システム オブジェクトのクエリと設定 ルーチン

2020/04/24 • [Edit Online](#)

ネットワークミニダイレクターによって実装できるルーチンの多くは、ファイルシステムオブジェクトに対するクエリ操作と設定操作です。RDBSS は、IRP を受信してファイルオブジェクトの情報を照会または設定することに応答して、ほとんどの呼び出しを発行します。そのため、RDBSS が受信する IRP と、RDBSS が呼び出す MRx クエリまたは set 操作との間に直接対応しています。

次の表に、ファイルシステムオブジェクトのクエリおよび設定操作を行うためにネットワークミニダイレクターで実装できるルーチンを示します。

IRP	MRx
MRxIsValidDirectory	RDBSS は、このルーチンを呼び出して、パスが有効なディレクトリであるかどうかをネットワークミニダイレクターが示すことを要求します。
MRxQueryDirectory	RDBSS は、このルーチンを呼び出して、ネットワークミニダイレクターがファイルディレクトリシステムオブジェクトの情報を照会するように要求します。
MRxQueryEaInfo	RDBSS は、このルーチンを呼び出して、ネットワークミニダイレクターがファイルシステムオブジェクトの拡張属性情報を照会するように要求します。RDBSS は、IRP_MJ_QUERY_EA の受信に応答してこの呼び出しを発行します。
MRxQueryFileInfo	RDBSS は、このルーチンを呼び出して、ネットワークミニダイレクターがファイルシステムオブジェクトのファイル情報を照会するように要求します。RDBSS は、IRP_MJ_QUERY_INFORMATION の受信に応答してこの呼び出しを発行します。
MRxQueryQuotaInfo	RDBSS は、このルーチンを呼び出して、ネットワークミニダイレクターがファイルシステムオブジェクトのクォータ情報を照会するように要求します。RDBSS は、IRP_MJ_QUERY_QUOTA の受信に応答してこの呼び出しを発行します。
MRxQuerySdInfo	RDBSS は、このルーチンを呼び出して、ネットワークミニダイレクターがファイルシステムオブジェクトのセキュリティ記述子情報を照会するように要求します。RDBSS は、IRP_MJ_QUERY_SECURITY の受信に応答してこの呼び出しを発行します。
MRxQueryVolumeInfo	RDBSS は、このルーチンを呼び出して、ネットワークミニダイレクターがボリューム情報を照会するように要求します。RDBSS は、IRP_MJ_QUERY_VOLUME_INFORMATION の受信に応答してこの呼び出しを発行します。

名前	説明
MRxSetEaInfo	RDBSS は、このルーチンを呼び出して、ネットワークミニダイレクターがファイルシステムオブジェクトに拡張属性情報を設定するように要求します。RDBSS は、IRP_MJ_SET_EA の受信に応答してこの呼び出しを発行します。
MRxSetFileInfo	RDBSS は、このルーチンを呼び出して、ネットワークミニダイレクターがファイルシステムオブジェクトのファイル情報を設定するように要求します。RDBSS は、IRP_MJ_SET_INFORMATION の受信に応答してこの呼び出しを発行します。
MRxSetFileInfoAtCleanup	RDBSS は、このルーチンを呼び出して、クリーンアップ時にファイルシステムオブジェクトのファイル情報を設定するようにネットワークミニダイレクターに要求します。RDBSS は、アプリケーションがハンドルを閉じてから閉じる前に、この呼び出しをクリーンアップ中に発行します。
MRxSetQuotaInfo	RDBSS は、このルーチンを呼び出して、ネットワークミニダイレクターがファイルシステムオブジェクトのクォータ情報を設定するように要求します。RDBSS は、IRP_MJ_SET_QUOTA の受信に応答してこの呼び出しを発行します。
MRxSetSdInfo	RDBSS は、このルーチンを呼び出して、ネットワークミニダイレクターがファイルシステムオブジェクトのセキュリティ記述子情報を設定するように要求します。RDBSS は、IRP_MJ_SET_SECURITY の受信に応答してこの呼び出しを発行します。
MRxSetVolumeInfo	RDBSS は、このルーチンを呼び出して、ネットワークミニダイレクターがボリューム情報を設定するように要求します。RDBSS は、IRP_MJ_SET_VOLUME_INFORMATION の受信に応答してこの呼び出しを発行します。

その他のネットワーク ミニリダイレクター ルーチン

2020/04/24 • [Edit Online](#)

ネットワークミニリダイレクターによって実装される他のいくつかのルーチンでは、状態管理と接続 ID のバッファリングが処理されます。ネットワークミニリダイレクターは、バッファリング状態管理ルーチンを使用して、リダイレクターおよびサーバーに実装されているさまざまなバッファリング方式を処理し、バッファリング状態の変化を通知します。たとえば、

MRxCompleteBufferingStateChangeRequest ルーチンは、サーバーに oplock 解除を送信するメカニズムの一部として SMB リダイレクターによって使用されます。

接続 ID は、複数の接続を多重化するために使用できます。ネットワークミニリダイレクターは接続 ID をサポートする必要がないため、**MRxGetConnectionId** は省略可能です。

次の表に、その他の操作用にネットワークミニリダイレクターで実装できるルーチンを示します。

名前	説明
MRxCompleteBufferingStateChangeRequest	RDBSS は、このルーチンを呼び出して、バッファリング状態の変更要求が完了したことをネットワークミニリダイレクターに通知します。たとえば、SMB リダイレクターは、このルーチンを使用して oplock 解除応答を送信するか、またはファイルが使用されなくなった場合に oplock 中断のハンドルを閉じます。サーバーに対してフラッシュする必要があるバイト範囲ロックは、RX_CONTEXT のネットワークミニリダイレクターに渡されます。これは、そのロックのメンバーです。
MRxComputeNewBufferingState	RDBSS はこのルーチンを呼び出して、ネットワークミニリダイレクターが新しいバッファリング状態の変化を計算するように要求します。
MRxGetConnectionId	RDBSS は、このルーチンを呼び出して、ネットワークミニリダイレクターが、複数のセッションを処理するために使用できる接続の接続 ID を返すように要求します。ネットワークミニリダイレクターで接続 ID がサポートされている場合、返された接続 ID は、name テーブルに格納されている接続構造に追加されます。RDBSS は、接続 ID を非透過的な blob と見なし、指定された名前の net name テーブルを参照している間に、接続 ID blob をバイト単位で比較します。

UNC 名前付けと MUP のサポート

2020/04/24 • [Edit Online](#)

マルチ UNC プロバイダー (MUP) は、リモートファイルシステムを処理できるネットワークリダイレクター (UNC プロバイダー) に汎用名前付け規則 (UNC) 名を使用して、すべてのリモートファイルシステムアクセスを channeling するカーネルモードのコンポーネントです。要求. MUP は、コマンドラインから実行できる次の例に示すように、アプリケーションで UNC パスを使用する場合に関係します。

```
notepad \\server\public\readme.txt
```

MUP は、マップされたドライブ文字 (たとえば、"NET USE" コマンド) を作成する操作中には関与しません。この操作は、複数のプロバイダールーター (MPR) と、ネットワークリダイレクターのユーザーモード WNet プロバイダー DLL によって処理されます。ただし、ユーザーモードの WNet プロバイダー DLL は、この操作中にカーネルモードのネットワークリダイレクタードライバーと直接通信できません。

Microsoft Windows Server 2003、Windows XP、および Windows 2000 では、分散ファイルシステム (DFS) ドライブを表していないマップされたドライブで実行されたリモートファイル操作は、MUP を経由しません。これらの操作は、ドライブ文字のマッピングを処理したネットワークプロバイダーに直接送られます。

Windows Vista リダイレクターモデルに準拠するネットワークリダイレクターの場合は、マップされたネットワークドライブが使用されている場合でも、MUP が関係します。マップされたドライブで実行されるファイル操作は、MUP 経由でネットワークリダイレクターに送られます。この場合、MUP は、関係するネットワークリダイレクターに操作を渡すだけです。

MUP は、mup.sys バイナリに含まれていません。これには、windows Server 2003、windows XP、および windows 2000 の Microsoft DFS クライアントも含まれています。

通常、カーネルネットワークリダイレクターは、リモートリソースへの接続の確立 (たとえば、リモートリソースへのドライブ文字のマッピング) をサポートするために、ユーザーモードの WNet プロバイダー DLL も備えています。MPR は、WNet プロバイダーに対するクエリに基づいてネットワーク接続を確立するユーザーモードの DLL です。MPR を呼び出すと、次のいずれかの結果が得られます。

コマンドプロンプトから発行された "net use x: \\server\share" コマンド。

Windows エクスプローラーから確立されたネットワークドライブ文字接続

WNet 関数を直接呼び出します。

ネットワークリダイレクターは、UNC 名を処理するために、MUP に登録する必要があります。MUP に登録されている複数の UNC プロバイダーを使用できます。これらの UNC プロバイダーは、次の1つまたは複数にすることができます。

- RDBSS に基づくネットワークミニリダイレクター
- RDBSS に基づいていない従来のリダイレクター

MUP は、名前ベースの操作で UNC パスを処理できるプロバイダーを決定します。通常は、IRP_MJ_CREATE 要求です。これは "プレフィックス解決" と呼ばれます。Windows Vista より前の場合、プレフィックス解決操作は次の2つの目的で機能します。

- プレフィックス解決が発生した名前ベースの操作は、プレフィックスを要求するプロバイダーにルーティングされます。成功した場合、MUP は、後続のハンドルベースの操作 (IRP_MJ_READ および IRP_MJ_書き込みなど) が、MUP を完全にバイパスする同じプロバイダーにアクセスすることを保証します。
- プロバイダーと要求されたプレフィックスは、MUP によって管理されるプレフィックスキャッシュに入力されます。その後の名前ベースの操作では、このプレフィックスキャッシュを使用して、プロバイダーがプレフィックスの解決を試行する前に既にプレフィックスを要求しているかどうかを判断します。このプレフィックスキャッシュ内の各エントリは、キャッシュに追加されると、タイムアウト (TTL) が適用されます。このタイムアウトが経過した後、エントリがスローされます。この時点で、MUP は、後続の名前

ベースの操作で、このプレフィックスに対してもう一度プレフィックスの解決を実行します。

MUP に登録されているネットワークリダイレクターに対して、[IOCTL_REDIR_クエリパス](#) 要求を発行して、プレフィックスの解決を実行します。IOCTL_REDIR_クエリパスの入力バッファと出力バッファは次のとおりです。

	IOCTL_REDIR_クエリパスの入力バッファ	IOCTL_REDIR_クエリパスの出力バッファ
0x00000000-0x0000000F	IrpSp-> Parameters. DeviceIoControl. Type3InputBuffer	QUERY_PATH_REQUEST
0x00000010-0x0000001F	IRP->UserBuffer	QUERY_PATH_RESPONSE

IOCTL とデータ構造は、*ntifs*で定義されています。バッファは、非ページプールから割り当てられます。

ネットワークリダイレクターは、IRP 構造体のRequesterModeメンバーがkernelmode であることを確認することで、この IOCTL のカーネルモードの送信を許可する必要があります。

MUP では、クエリパス_要求のデータ構造を使用して要求情報を参照します。

<pre>typedef struct _QUERY_PATH_REQUEST { ULONG PathNameLength; PIO_SECURITY_CONTEXT SecurityContext; WCHAR FilePathName[1]; } QUERY_PATH_REQUEST, *PQUERY_PATH_REQUEST;</pre>	
IOCTL_REDIR_クエリパス_要求	「
PathNameLength	Filepathnameメンバーに格納されている Unicode 文字列の長さ (バイト単位)。
SecurityContext	セキュリティコンテキストへのポインター。
FilePathName	<server><><パス>の形式の NULL で終わる NULL 以外の Unicode 文字列。文字列の長さ (バイト単位) は、PathNameLengthメンバーによって指定されます。

UNC プロバイダーでは、クエリパス_応答データ構造を使用して応答情報を提供する必要があります。

<pre>typedef struct _QUERY_PATH_RESPONSE { ULONG LengthAccepted; } QUERY_PATH_RESPONSE, *PQUERY_PATH_RESPONSE;</pre>	
IOCTL_REDIR_クエリパス_応答	「
0x00000000-0x0000000F	QUERY_PATH_REQUEST 構造体のFilepathnameメンバーに指定された Unicode 文字列パスからプロバイダーによって要求されたプレフィックスの長さ (バイト単位)。

IOCTL_REDIR_クエリパスは、IOCTL ではない_メソッドであることに注意してください。これは、入力バッファと出力バッファが同じアドレスにない可能性があることを意味します。UNC プロバイダーでよくある間違いは、入力バッファと出力バッファが同じであると想定し、入力バッファポインターを使用して応答を提供することです。

UNC プロバイダーが IOCTL_REDIR_QUERY_PATH 要求を受信すると、クエリのFilepathnameメンバーに指定されている unc パスを処理できるかどうかを判断する必要があります_パス_要求構造です。その場合は、__クエリの許容される長さのメンバーを、要求されたプレフィックスの長さ (バイト単位) で更新し、状態_成功を持つ IRP を完了する必要があります。プロバイダーは、指定された UNC パスを処理できない場合、IOCTL_REDIR_クエリ_パス要求に適切な NTSTATUS エラーコードを使用して失敗する必要があります。また、クエリの許容される長さのメンバー_パス_応答構造体を更新することはできません。プロバイダーは、他のメンバー、または任意の条件下でFilepathname文字列を変更することはできません。

\\サーバー\共有プレフィックス名 \が、UNC 名を使用する IRP_MJ_作成またはその他の Irp に応答して認識されない場合は、次のいずれかの方法で返すことをお勧めします。

状態_無効_ネットワーク_パス

ネットワークパスが見つかりません。コンピューター名 (\\サーバーなど) が無効であるか、ネットワークリダイレクターがコンピューター名を解決できません (名前解決メカニズムが使用可能な場合)。

状態_無効_ネットワーク_名

指定された共有名がリモートサーバーで見つかりません。コンピューター名 (\\サーバーなど) は有効ですが、指定された共有名がリモートサーバーで見つかりません。

状態_リソースが不足しています

バッファにメモリを割り当てるために使用できるリソースが不足しています。

デバイス_要求_状態_無効です

IOCTL_REDIR_クエリ_パス要求は、MUP からのものである必要があります。IRP のリクエストモードは常にkernelmode である必要があります。このエラーコードは、呼び出し元のスレッドのリクエストモードがkernelmode でなかった場合に返されます。

状態_無効_パラメーター

クエリ_パス_要求構造のPathNameLengthメンバーが、unicode 文字列で許容される最大長 (UNICODE_文字列_最大_バイト) を超えています。

ステータス_ログオン_失敗または状態_アクセス_拒否されました

無効または正しくない資格情報が原因でプレフィックス解決操作が失敗した場合、プロバイダーはリモートサーバーから返された正確なエラーコードを返す必要があります。これらのエラーコードを状態に変換することはできません_無効_ネットワーク_の名前または状態_正しくない_ネットワーク_パスです。ステータス_ログオン_の失敗や状態などのエラーコードは、適切な資格情報を使用する必要があることを示すフィードバックメカニズムとして機能_アクセス_拒否されます。これらのエラーコードは、ユーザーに資格情報の入力を自動的に求める場合にも使用されます。これらのエラーコードがないと、コンピューターにアクセスできないことをユーザーが想定している可能性があります。

ネットワークリダイレクターがプレフィックスを解決できない場合は、前述の推奨される NTSTATUS コードの一覧とは、意図したセマンティクスに厳密に一致する NTSTATUS コードを返す必要があります。ネットワークリダイレクターでは、実際に発生したエラーを返すことはできません (状態_接続_拒否されています。たとえば、NTSTATUS コードが上記の一覧にない場合)。

プロバイダーによって要求されたプレフィックスの長さは、個々の UNC プロバイダーによって異なります。ほとんどのプロバイダーでは、通常、\\<servername>\<sharename> \\<サーバー名>\<sharename>\<パス>のパスの一部を要求しています。たとえば、\\server\public\dir1\dir2 のパス \指定された \\server\public に要求された場合、プレフィックスはプレフィックスキャッシュに既に存在しているため、プレフィックスを解決せずに、プレフィックスをするプレフィックスの名前ベースのすべての操作がそのプロバイダーに自動的にルーティングされます。\\\\\\ただし、プレフィックス \\server\マーケティング\プレゼンテーションを含むパスは、プレフィックスの解決を行います。

ネットワークリダイレクターがサーバー名 (\\サーバーなど) を要求した場合、このサーバー上の共有に対するすべての要求は、このネットワークリダイレクターに送られます。この動作は、同じサーバー上で別のネットワークリダイレクターによってアクセスされる可能性がない場合にのみ許容されます。たとえば、UNC パスの \\サーバーを要求しているネットワークリダイレクターは、他のネットワークリダイレクターによるこのサーバー上の他の共有へのアクセスを禁止します (WebDAV は \\server\web にアクセスします)。

FsrtlregisterRDBSS プロバイダーを呼び出すことによって、REDIR プロバイダーとして登録されている従来のネットワークリダイレクターは、のクエリ_パス要求で IOCTL_を受け取ります。

UNC プロバイダーとしてのサポートを示すネットワークミニダイレクターは、このプレフィックス要求を、IRP_MJ_CREATE 呼び出しとして受信します。この作成要求は、ファイル_作成_ツリー_接続フラグが設定されたユーザーモードのCreatefile呼び出しに似ています。ネットワークミニダイレクターは、MRxLowIOSubmit[LOWIO_OP_IOCTL] の呼び出しとしてプレフィックス要求を受け取りません。プレフィックス要求の場合、RDBSS はMRxCreateSrvCall要求をネットワークミニダイレクターに送信し、その後MRxSrvCallWinnerNotifyとMRxCreateVNetRootを呼び出します。ネットワークミニダイレクターが RDBSS に登録すると、ネットワークミニダイレクターのドライバーディスパッチテーブルが RDBSS によってコピーされ、内部 RDBSS エントリポイントをポイントします。RDBSS は、この IOCTL_REDIR_クエリ_パスをネットワークミニダイレクターの内部で受信し、MRxCreateSrvCall、MRxSrvCallWinnerNotify、およびMRxCreateVNetRootを呼び出します。元のIOCTL_REDIR_クエリ_パス IRP は、MRxCreateSrvCallルーチンに渡される RX_CONTEXT 構造に含まれます。また、MRxCreateSrvCallに渡される RX_コンテキストの次のメンバーが変更されます:

元の IRP が IRP_MJ_デバイス_コントロールでも、MajorFunctionメンバーは IRP_MJ_CREATE に設定されます。

SuppliedPathNameメンバーは、クエリ_パス_要求構造のfilepathnameメンバーに設定されます。

SuppliedPathNameメンバーは、クエリ_パス_要求構造のPathNameLengthメンバーに設定されます。

SecurityContextメンバーは、クエリ_パス_要求構造のSecurityContextメンバーに設定されます。

ThisIsATreeConnectOpenメンバーはTRUEに設定されます。

Create.Flagsメンバーには、RX_コンテキスト_作成_フラグ_UNC_名前ビットセットがあります。

ネットワークミニダイレクターがプレフィックス要求の詳細を表示する必要がある場合は、MRxCreateSrvCallに渡される RX_コンテキストでこれらのメンバーを読み取ることができます。それ以外の場合は、サーバー共有への接続を試行するだけで、MRxCreateSrvCallの呼び出しが成功した場合には状態_SUCCESSを返すことができます。RDBSS は、ネットワークミニダイレクターに代わってプレフィックス要求を行います。

ネットワークミニダイレクターがこの IOCTL を直接受信できるケースが1つあります。ネットワークミニダイレクターは、RDBSS を初期化して登録する前に、ドライバーディスパッチテーブルのコピーを保存できます。RxRegisterMinirdrを呼び出して RDBSS に登録すると、ネットワークミニダイレクターは、RDBSS によってインストールされた新しいドライバーディスパッチテーブルのエントリポイントのコピーを保存し、元のドライバーディスパッチテーブルを復元できます。復元されたドライバーディスパッチテーブルを変更する必要があります。これにより、受信した IRP がネットワークミニダイレクターに関連するものであることを確認した後、その呼び出しは RDBSS ドライバーのディスパッチエントリポイントに転送されるようになります。ドライバーが RDBSS を初期化してRxRegisterMinirdrを呼び出すと、RDBSS はネットワークミニダイレクターのドライバーディスパッチテーブルをコピーします。Rdbsslibにリンクするネットワークミニダイレクターは、RxRegisterMinirdrを呼び出した後に RDBSS スタティックライブラリを初期化し、そのドライバーディスパッチテーブルを復元するために、driverentryルーチンからRxDriverEntryを呼び出す前に、元のドライバーディスパッチテーブルを保存する必要があります。これは、RDBSS がRxDriverEntryルーチンとRxRegisterMinirdrルーチンの両方でネットワークミニダイレクターディスパッチテーブルを介してコピーされるためです。

プレフィックスの解決中にプロバイダーが照会される順序は、次のキーの下に格納されている REG_SZ ProviderOrder レジストリ値によって制御されます。

```
HKLM\System\CurrentControlSet\Control\NetworkProvider\Order
```

ProviderOrder レジストリ値内の個々のプロバイダー名は、先頭または末尾の空白を除いてコンマで区切られます。

たとえば、次の値に文字列を含めることができます。

```
RDPNP,LanmanWorkstation,WebClient
```

UNC パス \\<server>\<共有>\<パス>である場合、MUP は、プレフィックス (\\サーバー\共有または \\サーバーなど) が MUP プレフィックスキャッシュに見つからない場合にプレフィックス解決要求を発行します。MUP は、プロバイダーがプレフィックスを要求する (またはすべてのプロバイダーがクエリされた) まで、プレフィックス解決要求を各プロバイダーに送信します。

1. TS クライアント (RDPNP)

2. SMB リダイレクター (LanmanWorkstation)

3. WebDAV リダイレクター (WebClient)

ProviderOrder レジストリ値を変更すると、Windows Server 2003、Windows XP、および Windows 2000 の MUP で再起動が有効になります。

MUP では、一覧に示されている各プロバイダー名を使用して、次のレジストリキーでプロバイダーのレジストリキーを検索します。

```
HKLM\System\CurrentControlSet\Services\<ProviderName>
```

次に、NetworkProvider サブキーの DeviceName 値を読み取って、プロバイダーが登録するデバイス名を見つけます。プロバイダーによって実際に登録されると、MUP は、に渡されたデバイス名と既知のプロバイダーのデバイス名のリストを照合し、プレフィックスの解決のためにプロバイダーを順序付きリストに配置します。この一覧に表示されるプロバイダーの順序は、前に説明した ProviderOrder レジストリ値に指定されている順序に基づいています。

このプロバイダーの順序は、複数のプロバイダールーター (MPR) にも適用されることに注意してください。これは、WNet プロバイダーに対するクエリに基づいてネットワーク接続を確立するユーザーモード DLL です。

Windows Server 2003 Service Pack 1 および Windows XP Service Pack 2 より前の場合、MUP 動作では、ProviderOrder レジストリ値で指定された順序ですべてのプロバイダーに対して "並列" でプレフィックス解決要求を発行し、すべてのプロバイダーを待機していました。を実行して、プレフィックス解決操作を完了します。したがって、最初のプロバイダーがプレフィックスを要求した場合でも、その他のすべてのプロバイダーがプレフィックス解決操作を完了するまで、MUP は待機します。複数のプロバイダーがプレフィックス要求で応答する場合、MUP は ProviderOrder レジストリ値に指定されている順序に基づいてプロバイダーを選択します。

Windows XP Service Pack 2 以降、および Windows Server 2003 Service Pack 1 以降では、この動作は少し変更されました。MUP は、プレフィックス解決要求を直列に発行し、最初のプロバイダーがプレフィックスを要求するとすぐに停止します。したがって、上記の例では、RDPNP がプレフィックスを要求した場合、MUP は SMB または WebDAV のリダイレクターを呼び出しません。

この動作が変更された主な理由は、"シリアルプレフィックスの解像度" スキームによって、ProviderOrder 値の優先順位が低いネットワークリダイレクターのケースにより、ProviderOrder 値。たとえば、ファイアウォールが設定されたリモートサーバーで、特定の種類の TCP/IP パケット (HTTP へのアクセスなど) をブロックし、他のユーザー (SMB アクセスなど) を許可するように構成されているとします。この場合、SMB ネットワークリダイレクターが ProviderOrder 値の最初のプロバイダーとして構成されていて、そのプレフィックスを迅速に要求しても、WebDAV リダイレクターは、への TCP 接続を待機することにより、プレフィックスの解決の完了を大幅に遅延する可能性があります。タイムアウト。

MUP と DFS の対話

2020/04/24 • [Edit Online](#)

アプリケーションでは、汎用名前付け規則 (UNC) パスを使用する場合に使用するには、どのネットワーク プロバイダーを決定する、要求が複数 UNC プロバイダー (MUP) に送信されます。MUP はチャネルがリモート ファイル システムの要求を処理できる適切なネットワークリダイレクター (UNC プロバイダー) への要求です。MUP が最初に、特定の要求を渡す、分散ファイル システム (DFS) クライアントが有効になっている場合 "\\server\共有" DFS クライアント要求が DFS のかを判断するには、通常のリモート ファイル 共有ではなく共有します。

既定の動作は、DFS クライアントが有効になっています。DFS クライアントは、次の下にあるレジストリ エントリの値に応じて無効になっています。

```
HKLM\System\CurrentControlSet\Services\Mup
```

DisableDfs のレジストリ エントリは、1 の DWORD 値を持つ、DFS クライアントが無効になります。

MUP、(有効) の場合、DFS クライアントによって保持されるプレフィックスのキャッシュにない名前ベースの操作で指定されたパスのプレフィックスと仮定すると暗黙的にすべてリダイレクターよりも優先されます。DFS クライアントが指定した UNC パスが DFS パスであるかどうかを確認しようとするします。この機能を使用するには、適切なサーバーの IPC\$ 共有に参照の要求を送信します。パスを判断して、DFS のパスを指定する、DFS クライアントが操作を処理します。それ以外の場合、DFS クライアントは、名前に基づく要求を MUP の適切なリダイレクターによって処理されるプレフィックスの解決に渡します。

IPC\$ 共有にアクセスする要求がシステムに送信されると、LAN Manager サーバー (SMB サーバーとも呼ばれます)、srv.sys が無効にするか (UNIX システムの例) にインストールされていないに接続するいくつかの試行が行われるため、遅延がある可能性があります、IPC\$ 共有します。この遅延は 5 ~ 7 (秒単位) では通常が、速度と接続するネットワーク インフラストラクチャとその他の条件の待機時間に基づくなくなりました。

Microsoft Windows Vista における MUP の変更点

2020/04/24 • [Edit Online](#)

Windows Vista では、ネットワークリダイレクターに影響を与える可能性がある複数の UNC プロバイダー (MUP) に対して多数の変更が実装されています。

MUP と分散ファイルシステム (DFS) クライアントは別々のバイナリファイルにあります。MUP コンポーネントは `mup.sys` にあり、DFS クライアントは `dfsc` にあります。Windows Server 2003、Windows XP、および Windows 2000 では、MUP カーネルコンポーネントである `mup.sys` にも、DFS クライアントが含まれていました。

Windows Vista では、新しいリダイレクターモデルが定義されています。

- MUP は、`IoRegisterFileSystem` を呼び出すことによって、i/o マネージャーを使用してファイルシステムとして登録します。
- ネットワークリダイレクターは、Windows Vista で導入された新しいルーチンである `FsRtlRegisteruncproviderex` を使用して、MUP に登録します。
- ネットワークリダイレクターは、名前のないデバイスオブジェクトを `FsRtlRegisterfinalcx` に渡します。
- ネットワークリダイレクターは、デバイス名を `FsRtlRegisteruncproviderex` に渡します。
- ネットワークリダイレクターは、i/o マネージャーを使用してファイルシステムとして登録しません (`IoRegisterFileSystem` を呼び出すことはありません)。
- MUP からネットワークリダイレクターへのすべての呼び出し (プレフィックス解決、`Ioctl`、`FSCTLs` を含む) は、`Apc` が有効になっています。他のコンポーネントから MUP へのすべての呼び出しは、`Apc` が有効になっていると想定されます。
`FsRtlCancellableWaitForSingleObject` または `FsRtlCancellableWaitForMultipleObjects` で呼び出しを使用し、Windows Vista で導入された新しいルーチンがある場合、i/o 要求を発行したスレッドが末尾。
- プレフィックスの解決は、Windows Vista で導入された新しい `IOCTL` である、`IOCTL_REDIRECT_EX` を使用して実行されます。
- MUP に登録されているネットワークリダイレクターのデバイス名は、MUP デバイスオブジェクトへのシンボリックリンクになります。

Windows Vista リダイレクターモデルに準拠するネットワークリダイレクターの場合、MUP では、オブジェクトマネージャーの名前空間にシンボリックリンクを作成します。これには、`FsRtlRegister` の呼び出しでネットワークリダイレクターによって指定されたデバイス名が使用されます。このシンボリックリンクのターゲットは、MUP デバイスオブジェクト (`\Device\Mup`) です。

MUP をファイルシステムとして登録し、ネットワークリダイレクターのデバイス名を、MUP デバイスオブジェクトにシンボリックリンクとして登録する利点は、名前ベースの操作だけでなく、すべてのリモートファイルシステム i/o 操作が、MUP を経由することです。したがって、リモートファイルシステムスタックに存在する必要があるファイルシステムフィルタードライバーは、単に MUP デバイスオブジェクトにアタッチできます。ファイルシステムフィルタードライバーは、プロバイダーのデバイスオブジェクト名 (`\Device\LanmanRedirector` など) をドライバにハードコーディングする必要はありません。これにより、ファイルシステムフィルタードライバーは、すべてのネットワークリダイレクターに発行されたすべての i/o 操作を1つの添付ファイルで監視できます。これにより、Windows Vista より前のファイルシステムフィルタードライバーによって検出された重複 i/o 操作が回避されます。これは、DFS (`mup.sys`) と個々のネットワークリダイレクター (`\Device\LanmanRedirector` など) に個別に接続されています。i/o 操作を両方に対して監視します。

MUP デバイスオブジェクトに接続されているファイルシステムフィルタードライバーは、特定のネットワークリダイレクターに送信されるトラフィックを選択的にフィルタ処理できます。この場合、フィルタードライバーは、`FsRtlMupGetProviderIdFromName` ルーチンを呼び出すことによって、目的のネットワークリダイレクターのデバイス名をプロバイダー識別子にマップします。フィルタードライバーは、特定のファイルオブジェクトのトラフィックをフィルタ処理するかどうかを判断できます。そのためには、

`FsRtlMupGetProviderInfoFromFileObject` ルーチンの呼び出しによって取得されたプロバイダー識別子を、のプロバイダー

識別子と比較します。対象のネットワークディレクター。

Windows Vista リダイレクターモデルに準拠するネットワークリダイレクターの場合:

- リモートファイルシステムスタック上のすべてのファイルオブジェクトが、MUP に解決されます。そのため、[IoGetDeviceAttachmentBaseRef](#)は、ファイルオブジェクトを所有するネットワークリダイレクターではなく、MUP のデバイスオブジェクトを返します。ただし、ファイルオブジェクトの内容はネットワークリダイレクターによって所有されています。
- ネットワークリダイレクター (\デバイス\LanmanRedirector\server\share など) のデバイス名に対して発行された IRP_MJ_ の作成は、MUP プレフィックスを経由せずに、そのネットワークリダイレクターを対象とします。解決方法は、Windows Server 2003、Windows XP、および Windows 2000 の場合とまったく同じです。

Windows Vista RDBSS (動的または静的にリンク) に基づいていないネットワークリダイレクターは、"従来のリダイレクター"と呼ばれます。これらのレガシネットワークリダイレクターには次のものが含まれます。

- Windows Server 2003、Windows XP、および Windows 2000 用に記述されたネットワークリダイレクターは、[Fsrtlregisteruncprovider](#)を使用して、MUP に直接登録します。
- Windows server 2003、windows XP、または Windows 2000 の rdbsslib ライブラリと静的にリンクする Windows Server 2003、Windows XP、および Windows 2000 用に作成されたネットワークミニリダイレクター。
- [Fsrtlregister](#)の登録を使用して、MUP に直接登録する Windows Vista 用に記述されたネットワークリダイレクター。

Windows Vista RDBSS (RDBSS) に動的にリンクするネットワークミニリダイレクターは、Windows Vista リダイレクターモデルに自動的に準拠します。これは、RDBSS が[Fsrtlregister Providerex](#)を使用して MUP に登録するためです。Windows Vista RDBSS (rdbsslib) に対して静的にリンクされるネットワークミニリダイレクターは、Windows Vista リダイレクターモデルにも自動的に準拠します。これは、RDBSS が[Fsrtlregister Providerex](#)を使用して MUP に登録するためです。

Windows Vista 用に作成された従来のネットワークリダイレクターは、MUP に直接登録する必要があります Windows Vista リダイレクターモデルに準拠している必要があります。

Windows Server 2003、Windows XP、および Windows 2000 用に記述されたネットワークリダイレクターは、[Fsrtlregister プロバイダー](#)を使用して直接 MUP に登録すると、windows server 2003、windows xp、および windows 2000 で行ったのとまったく同じ方法で動作します。Windows server 2003、windows XP、および Windows 2000 の rdbsslib ライブラリと静的にリンクする Windows Server 2003、Windows XP、および windows 2000 用に作成されたネットワークミニリダイレクターは、Windows Server 2003 とまったく同じように動作します。Windows XP および Windows 2000。これらのレガシネットワークリダイレクターとミニリダイレクターは、次の動作を示しています。

- ファイルシステムの登録を監視するファイルシステムフィルタードライバーに表示されます。
- デバイスオブジェクトにはという名前が付けられます。デバイス名はシンボリックリンクではなく、\デバイス\MUP を指していません。
- ファイルオブジェクトは、ネットワークリダイレクターの名前付きデバイスオブジェクトに解決されます。
- MUP は、プレフィックス解決操作にのみ関係します。ネットワークプロバイダーが特定されると、状態_再解析を返すことによって、MUP "がその方法を利用できなくなります。後続のすべての操作は、MUP を通過しません。

この動作は、プロバイダーのデバイス名が\デバイス\MUP にシンボリックリンクされている場合に発生する可能性がある、ダブルフィルター処理を防ぐために保持されています。この2つのフィルター処理は、次の理由で発生します。

- ファイルシステムフィルタードライバーは、既に \デバイス\MUP にアタッチされています。
- ファイルシステムフィルタードライバーは、登録している任意のファイルシステムにアタッチします。名前付きデバイスオブジェクトを使用するネットワークリダイレクターは自身をファイルシステムとして登録するため、ファイルシステムフィルタードライバーは、同じ i/o を2回フィルター処理することになります。

Windows Vista での MUP との呼び出しは、Apc が有効になっているため、次のような影響があります。

- 必要に応じて、必要に応じて、適切な方法によってスレッドの中断に対して MUP から呼び出されるコードパスを保護する

ことが重要です。特に、IOCTL_REDIR_クエリ_パスハンドラーによって実行されます。スレッドの中断は、長時間に及ぶ可能性がある "無制限の待機" 操作であることに注意してください。

- (システムスレッドではなく) ユーザーモードスレッドに関係する "wait for i/o" 操作では、常に "キャンセル可能な wait" が使用されるようにすることが重要です。詳細については、[FsRtlCancellableWaitForSingleObject](#) ルーチンと [FsRtlCancellableWaitForMultipleObjects](#) ルーチンを参照してください。
- 何らかの重要なロックを保持しているスレッドが中断されると、デッドロックが発生する可能性があります。ユーザーモードのスレッドが存在する場合は、デッドロック状態をチェックするために中断されないように、テストを実行することが重要です。
- テストを実行して、"i/o 操作の待機" が本当にキャンセル可能なかどうかを確認し、ユーザーモードアプリケーションがスレッドをすぐに終了できるようにすることが重要です。これにより、アプリケーションは、スレッドを終了します。

Windows Vista の MUP で使用されるプレフィックスキャッシュサイズとタイムアウトは、次のレジストリ値によって制御されるようになりました。

- PrefixCacheSizeInKB
- PrefixCacheTimeoutInSeconds.

これらのレジストリ値は、再起動せずに動的に変更できます。これらのレジストリ値は、次のレジストリキーの下にあります。

```
HKLM\System\CurrentControlSet\Services\Mup\Parameters.
```

プロバイダーが個々のリダイレクターにプレフィックス解決要求を発行する順序を決定する ProviderOrder レジストリ値は、システムを再起動しなくても動的に変更できます。このレジストリ値は、次のレジストリキーの下にあります。

```
HKLM\CurrentControlSet\Control\NetworkProvider\Order
```

Windows Vista では、MUP は、ネットワークリダイレクターが MUP に登録されているかどうかによって、[Fsrtlregisteruncprovider](#) または [fsrtlregister](#) を呼び出すことによって、プレフィックスの解決方法が異なります。[FsrtlregisterREDIR](#) プロバイダーを呼び出すことによって MUP に登録するレガシネットワークのリダイレクターは、プレフィックス解決のために _パス要求に対して、[IOCTL__クエリ](#) を受け取ります。これは、Windows Server 2003、Windows XP、および Windows 2000 で使用される方法と同じです。

Windows Vista リダイレクターモデルに準拠し、[FsrtlregisterREDIR Providerex](#) を呼び出すことによって MUP に登録されたネットワークリダイレクターは、[_、_クエリ__パス](#) を使用して、プレフィックスに対する EX 要求を送信します。解決: Windows Vista では、ネットワークミニリダイレクターは rdbsslib に静的にリンクされているか、rdbss で動的にリンクされているため、RDBSS を通じて間接的に [Fsrtlregister](#) を呼び出します。

IOCTL_REDIR_クエリ_パス_EX の入力バッファーと出力バッファーは、次のようになります。

	パラメーターはで使用できます。	データ構造の形式
入力バッファー	IrpSp-> Parameters. DeviceIoControl. Type3InputBuffer	QUERY_PATH_REQUEST_EX
出力バッファー	IRP->UserBuffer	QUERY_PATH_RESPONSE

IOCTL とデータ構造は、ntifs で定義されています。バッファーは、非ページプールから割り当てられます。

ネットワークリダイレクターは、`Irp>irp->requestormode` が `kernelmode` でであることを確認することによって、この IOCTL のカーネルモードの送信者のみに優先する必要があります。

MUP では、クエリの _パス_ 要求情報に `_EX` データ構造を使用します。

```
typedef struct _QUERY_PATH_REQUEST_EX {
    PIO_SECURITY_CONTEXT  pSecurityContext;
    ULONG  EaLength;
    PVOID  pEaBuffer;
    UNICODE_STRING  PathName;
} QUERY_PATH_REQUEST_EX, *PQUERY_PATH_REQUEST_EX;
```

「	」
pSecurityContext	セキュリティコンテキストへのポインター。
EaLength	拡張属性バッファの長さ (バイト単位)。
pEaBuffer	拡張属性バッファへのポインター。
■	<server> <> <パス>の形式の NULL で終わる NULL 以外の Unicode 文字列。

UNC プロバイダーでは、クエリパス_応答データ構造を使用して応答情報を提供する必要があります。

```
typedef struct _QUERY_PATH_RESPONSE {
    ULONG  LengthAccepted;
} QUERY_PATH_RESPONSE, *PQUERY_PATH_RESPONSE;
```

「	」
■	QUERY_PATH_REQUEST_EX 構造体のPathNameメンバーに指定された Unicode 文字列パスからプロバイダーによって要求されたプレフィックスの長さ (バイト単位)。

IOCTL_REDIR_クエリパス_EX は、IOCTL ではないメソッド_メソッドであることに注意してください。これは、入力バッファと出力バッファが同じアドレスにない可能性があることを意味します。UNC プロバイダーでよくある間違いは、入力バッファと出力バッファが同じであると想定し、入力バッファポインターを使用して応答を提供することです。

UNC プロバイダーが_クエリ_PATH_EX 要求の IOCTL を受け取ると、クエリのPathNameメンバーに指定されている unc パスを処理できるかどうかを判断する必要があります_パス_要求_exデータ。その場合、UNC プロバイダーは、_クエリの許容される長さのメンバーを、要求されたプレフィックスの長さ (バイト単位) で更新する必要があります。これにより、状態_SUCCESS で IRP を完了します。プロバイダーは、指定された UNC パスを処理できない場合、IOCTL_REDIR_QUERY_PATH_EX 要求を適切な NTSTATUS エラーコードで失敗させる必要があります。また、クエリの許容される長さのメンバーを更新することはできません_パス_応答構造体。プロバイダーは、任意の条件下で、他のメンバーまたはパス名 文字列を変更することはできません。

Windows Vista では、UNC プロバイダーとしてのサポートを示す RDBSS の使用に基づくネットワークミニリダイレクターは、このプレフィックス要求を、ファイル_作成_ツリーを使用したユーザーモードの Createfile 呼び出しと同様に、通常のツリー接続作成として受信し_接続フラグを設定します。RDBSS は、[MRxCreateSrvCall](#)要求をネットワークミニリダイレクターに送信し、その後に[MRxSrvCallWinnerNotify](#)と[MRxCreateVNetRoot](#)を呼び出します。このプレフィックス要求は、[MRxLowIOSubmit\[LOWIO_OP_IOCTL\]](#) への呼び出しとして受信されません。ネットワークミニリダイレクターが RDBSS に登録すると、ネットワークミニリダイレクターのドライバディスパッチテーブルが RDBSS によってコピーされ、内部 RDBSS エントリポイントをポイントします。RDBSS は、この IOCTL_REDIR_クエリパス_EX をネットワークミニリダイレクターの内部で受信し、[MRxCreateSrvCall](#)、[MRxSrvCallWinnerNotify](#)、および[MRxCreateVNetRoot](#)を呼び出します。元の IOCTL_REDIR_クエリパス_EX IRP は、[MRxCreateSrvCall](#)ルーチンに渡される RX_コンテキストに含まれます。また、

MRxCreateSrvCallに渡される RX_コンテキストの次のメンバーが変更されます:

元の IRP が IRP_MJ_デバイス_コントロールでも、MajorFunctionメンバーは IRP_MJ_CREATE に設定されます。

SuppliedPathNameメンバーは、クエリ_PATH_REQUEST_EX 構造体の PrefixClaimのメンバーに設定されます。

SuppliedPathName メンバーは、クエリ_PATH_REQUEST_EX 構造体のPrefixClaimメンバーに設定されます。

ThisIsATreeConnectOpenメンバーはTRUEに設定されます。

ThisIsAPrefixClaimメンバーはTRUEに設定されます。

SecurityContextメンバーは、クエリ_PATH_REQUEST_EX 構造体のSecurityContextメンバーに設定されます。

EaBufferメンバーは、クエリ_PATH_REQUEST_EX 構造体のpEaBufferメンバーに設定されます。

EaLengthメンバーは、クエリ_PATH_REQUEST_EX 構造体のEaLengthメンバーに設定されます。

Create. Flagsメンバーには、RX_コンテキスト_作成_フラグ_UNC_名前ビットセットが設定されます。

ネットワークミニリダイレクターがプレフィックス要求の詳細を表示する必要がある場合は、MRxCreateSrvCallに渡される RX_コンテキスト構造でこれらのメンバーを読み取ることができます。それ以外の場合は、サーバー共有への接続を試行するだけで、MRxCreateSrvCallの呼び出しが成功した場合には状態_SUCCESS を返すことができます。RDBSS は、ネットワークミニリダイレクターに代わってプレフィックス要求を行います。

Microsoft Windows Vista におけるトランザクション のサポート

2020/04/24 • [Edit Online](#)

Windows vista ではありません現在 RDBSS に基づいてネットワーク ミニ リダイレクターのトランザクションのサポートです。

ネットワーク リダイレクターとファイル システム の処理

2020/04/24 • [Edit Online](#)

ファイルシステムドライバーによって実行されるほとんどのファイル操作は、通常、ユーザーのスレッドコンテキストで完了します。これらの操作には、ファイルシステムに対する同期ファイル i/o 呼び出しのすべてが含まれます。このような場合は、すべての作業がインラインで実行されます。操作はカーネル内でブロックする可能性があります、作業は同じスレッドで実行されます。Windows のファイルシステムでは、多くの場合、ファイルシステムディスパッチ (FSD) としてユーザーのスレッドコンテキストを使用してこの作業を呼び出します。ほとんどの場合、ファイルシステムは FSD での作業を完了しようとします。

アプリケーションでブロックしない場合があります (非同期の読み取りまたは非同期の書き込みなど)。このような場合は、ファイル i/o 操作をシステムワーカースレッドにディスパッチして完了する必要があります。Windows のファイルシステムでは、多くの場合、ファイルシステムプロセス (FSP) としてシステムワーカースレッドコンテキストを使用してこの作業を呼び出します。ネットワークミニリダイレクターに送信される非同期 IRP_MJ_書き込み要求の例を次に示します。これは、FSP を使用する必要があるケースを示しています。

非同期の IRP_MJ_書き込み要求を実行するには、RDBSS またはネットワークミニリダイレクターが、ファイルオブジェクトの FCB リソース (ファイルオブジェクトの変更に使用される同期オブジェクト) を取得する必要があります。FCB リソースが既に別のスレッドと RDBSS によって保持されている場合、またはネットワークミニリダイレクターがユーザーのスレッドコンテキスト (FSD) で FCB リソースを取得しようとした場合、この操作はブロックされます。ファイルシステムへの非同期要求はブロックされません。非同期要求 (IRP_MJ_書き込み要求など) の場合、ファイルシステムドライバーは、必要なリソースが使用可能かどうかを最初に確認します (たとえば、ネットワークミニリダイレクターの FCB リソース)。リソースが使用できない場合、ファイルシステムドライバーは、システムワーカースレッド (FSP) によって後で完了するために、要求を作業キューにポストします。ファイルシステムは、FSD スレッドから状態 `_PENDING` を返します。ユーザーアプリケーションに対して、FSD は状態 `_直ちに返されますが、実際の作業は FSP のシステムワーカースレッドによって処理されます。`

ファイルシステムドライバーが FSP にポストする前に、いくつかのタスクを完了する必要があります。ファイルシステムドライバーは、システムワーカースレッドによって作業が完了するため、FSD 内のユーザーのスレッドからセキュリティコンテキストをキャプチャする必要があります。RDBSS は、ネットワークミニリダイレクターの `RxFsPostRequest` ルーチンでこれを自動的に行います。このルーチンは、RDBSS によって呼び出されます。ネットワークミニリダイレクターは、RX の `Postrequest` メンバーによって保留中のステータス_を返すたび_コンテキスト構造を `TRUE` に設定します。作業が作業キューに投稿された場合、ファイルシステムドライバーは、ユーザーバッファがシステムワーカースレッドによって後で使用できるようになるようにする必要があります。このタスクを実行するには、次の2つの方法があります。

1. ファイルシステムドライバーは、FSP にポストする前にユーザーバッファをカーネルメモリ領域にマップして、後でシステムワーカースレッド (FSP) がバッファにアクセスできるようにします。ファイルシステムドライバーによって一般的に使用されるメソッドは、FSP のユーザーバッファをロックすることです。これは、メモリページは常にシステムワーカースレッドによってマップされるためです。
2. ファイルシステムは、FSP から呼び出し元プロセスのスレッドを保存できます。また、FSP では、システムワーカースレッドがこの呼び出しプロセスにアタッチできます。 `Kestackattachprocess` を使用すると、システムワーカースレッドはユーザーの呼び出しプロセスにアタッチされ、ユーザーバッファにアクセスして、作業の完了時に `KeUnstackDetachProcess` を使用してユーザーの呼び出しプロセスからデタッチします。

RDBSS は `RxFsPostRequest` ルーチン内のメソッド1を使用してユーザーバッファを自動的にロックします。これは、RX_コンテキスト_フラグ_、必要なビットが RX_コンテキスト構造の `Flags` メンバーで設定されていない場合に限ります。_ユーザーバッファは次の要求に対してロックされます:

- `Irps_MJ_DIRECTORY_` 制御するために、IRP のマイナー機能を使用して_します。クエリ_ディレクトリ_
- `IRP_MJ_クエリ_EA`

- Irp_MJ は、minor 関数に IRP___が含まれていない場合に読み取りを行います。
- IRP_MJ_設定_EA
- Irp_MJ_、minor 関数に IRP_の正常な_MDL が含まれていない限り書き込みを行う

メソッド2は、通常、少量の情報だけが渡されて返される場合には、メソッドを使用する Irp の処理に使用されます。これらの Irp には、次のものが含まれます。

- IRP_MJ_DEVICE_CONTROL
- IRP_MJ_ファイル_システム_コントロール
- IRP_MJ_INTERNAL_DEVICE_CONTROL

RDBSS では、操作のMrxLowIoSubmit配列の非同期呼び出しのみがサポートされています。ネットワークミニダイレクターが非同期呼び出しとして他の操作 (**MRxQueryFileInfo**など) を実装する必要がある場合、ネットワークミニダイレクターは要求を FSP にポストする必要があります。ネットワークミニダイレクターが rx_コンテキスト、rx_コンテキスト構造のFlagsメンバーで設定された非同期_操作ビットを持つMrxQueryFileInfoの要求を受け取った場合、ネットワークミニダイレクターには、非同期操作のためにこの要求を FSP にポストする場合は、MrxQueryFileInfoルーチンの動作では、ネットワークミニダイレクターはまず、ユーザーのスレッドのセキュリティコンテキストをキャプチャし、ユーザーバッファをカーネル領域にマップする必要があります (または、ユーザーの呼び出しにアタッチするようにシステムワーカースレッドを設定します)。FSP での実行中に処理されます)。次に、ネットワークミニダイレクターは、RX_コンテキスト構造のPostrequestメンバーをTRUEに設定し、FSD から STATUS_PENDING を返します。この作業は、システムワーカースレッド (FSP) による操作のために、RDBSS によって作業キューにディスパッチされます。

ファイル システムのセキュリティの概要

2020/04/24 • [Edit Online](#)

WDK を使用するファイル システムの開発者は、ファイル システムの実装時にセキュリティの考慮事項に注意してください。ファイル システム フィルター ドライバー開発者は、これらの問題では、注意すべきし、セキュリティに関する考慮事項を自社製品に組み込むためには使い慣れた Microsoft Windows セキュリティ インターフェイスを使用する必要があります。

このセクションでは、Windows のセキュリティとファイル システムに重点を置いています。Windows、またはファイル システムの開発者に Windows 環境で Windows 以外のセキュリティ モデルを実装する場合は、セキュリティ拡張機能を開発する必要がある場合、入門書にするものではありません。

このセクションには、多くサンプル コードにはが含まれています。サンプルでは、実際のコードから抽出されますが、省略形でここに表示されます。開発者は、特定の環境で使用するため、これらのサンプルを調整する必要があります。

ドライバーのセキュリティについて

2020/04/24 • [Edit Online](#)

ドライバーに関しては、セキュリティの欠陥を何もユーザーが実行できる方法は、システムがクラッシュすることでの誤動作したり、使用できなくなつたためのドライバーが原因となるです。ほとんどの開発者は、そのドライバーで作業しているは、ドライバーが正常に動作するかどうか悪意のある侵入者はしようし、システム内でのセキュリティホールを悪用の取得に、フォーカスがあります。これは、ファイルシステムといくつかを記述するドライバーの最も複雑な種類のファイル システム フィルター ドライバーのさらに多くの場合です。

ただし、製品のリリースでは後、は、プローブし、セキュリティ上の弱点を識別しようとしています。ユーザーがあります。したがって、それによって開発者はこのようなセキュリティホールが存在する可能性を最小限に抑えるために設計および実装フェーズ中にこれらの問題を検討してください。目標は、リリース済み製品の一部になる前に、できるだけ多くのセキュリティホールを排除します。

セキュリティで保護されたドライバーを達成するには (意識的ドライバーに潜在的な脅威の考え) デザイナー、(防御的なコーディングの脆弱性のソースは、一般的な操作)、実行者と (事前にしようとして、テスト チームの協力が必要検索の悪用) します。によってこれらのアクティビティのすべてを適切に調整、ドライバーのセキュリティが大幅に拡張します。

セキュリティの脅威モデル

2020/04/24 • [Edit Online](#)

セキュリティを考慮して、一般的な方法としては、可能な攻撃の種類を記述する特定の脅威モデルを作成する方法があります。この手法は、ファイルシステムまたはファイルシステムフィルタードライバーを設計する場合に役立ちます。これは、開発者がドライバーに対する潜在的な攻撃ベクトルを考慮するように強制するためです。潜在的な脅威を特定した後、ドライバーの開発者は、ドライバーコンポーネントの全体的なセキュリティを強化ために、これらの脅威に対する防御手段を検討できます。

セキュリティ脅威モデルを検討する際には、ユーザー i/o 要求 (セキュリティチェックの対象) とドライバー自体によって開始される i/o 操作 (既定ではセキュリティチェックの対象外) に代わって、操作ドライバーが管理するアクションを区別することも重要です。1つのドライバーに対するユーザーモード要求は、内部 FSCTL または IOCTL 要求 (たとえば、Srv.sys ドライバー) を介して別のドライバーに渡されることもあります。この場合、これらの問題はさらに複雑になります。

カーネルモードドライバーの開発者にとっては、次の重要な問題を考慮する必要があります。

- ドライバーによって開始された i/o 操作は、ローカルシステムのセキュリティチェックをバイパスします。
- ドライバーによって開始された i/o 操作は、ほとんどのパラメーターの検証チェックをバイパスします。
- ドライバーは、信頼されたコンピューティングベースの一部であるため、システム全体を制御できます。

たとえば、悪意のあるプログラムがシステムにドライバーを正常に読み込むことができたとします。これにより、非常に優れた制御が提供され、実質的にシステムが危険にさらされることになります。アプリケーションがドライバーの機能を利用して同じことを実現できる場合は、アプリケーションに制御を渡し、システムを侵害することになります。

Microsoft では、セキュリティを考慮する際に "STRIDE" モデルを使用します。

- S--[なりすまし id](#)。
- [データの改ざん](#)。
- R--[否認](#)。
- I--[情報の漏えい](#)。
- D--[サービスの拒否](#)。
- E--[特権の昇格](#)。

ここでは基本的な原則として、発生する可能性がある特定の種類のシステム侵害を特定します。これらの原則の一部は一般にドライバーに対して有効ですが、ファイルシステムおよびファイルシステムフィルタードライバーには有効です。

なりすまし

2020/04/24 • [Edit Online](#)

Id 偽装の概念が他のユーザーの id を使用する特権のないコードを許可し、そのため、セキュリティ資格情報。たとえば、何らかの形式のパスワードのメカニズムを使用するドライバーは、この種の攻撃の対象は。このようなすべてのドライバーでは、ですが、id 偽装に基づくセキュリティの欠陥に対して脆弱なセキュリティの欠陥があります。デザイナーとドライバーの実装者は、脆弱性のレベルを評価する必要があります。

Id のなりすましの他のより複雑な例が存在します。たとえば、スマートカードの紛失または盗難時、復号化キー用のスマートカードに依存する暗号化フィルター ドライバーは物理スプーフィング攻撃は。そのため、フィルター ドライバーがいくつかの追加要件、生体認証の確認やパスワードなどのこのカテゴリの攻撃から保護するために追加します。

独自のセキュリティ チェックを実行しようとするドライバーは、セキュリティの確認中に適切な資格情報を使用するように特に注意する必要があります。そのためには、障害は、いるセキュリティ記述子が正しいために、他のユーザーによって実行されている可能性のある操作を実行することを見出した悪意のあるユーザーは、スプーフィング攻撃を簡単に提供できます。

一般に、ドライバーは最適な設計と実装を独自に構築するのではなく、オペレーティング システムを内の既存のセキュリティ機構を利用する場合。これにより、実装がエラーを含めることができます、潜在的な場所の数を最小限に抑えられます。

データの改ざん

2020/04/24 • [Edit Online](#)

データの改ざんはドライバーの脅威ですが、ファイルシステムおよびファイルシステムフィルタードライバーにとって深刻な脅威となります。すべてのドライバーについて、ユーザーモードコンポーネントとカーネルモードコンポーネントの間で共有される制御構造は、カーネルモードコンポーネントによって使用されている間にユーザーモードコンポーネントによって変更される可能性があります。ファイルシステムとフィルタードライバーは、このクラスの攻撃に対して特に脆弱です。これは、仮想アドレスを使用してユーザーバッファに直接アクセスするデータ転送の種類、メソッドに強力に依存しているためです。高速 i/o も、生のユーザーモードバッファに直接アクセスします。ここでのリスクは、アプリケーションがデータを変更するときに、ドライバーがこのデータを使用している可能性があることです。通常、ドライバーはデータを検証することによって、このような保護を試みます。ただし、データの検証は、検証後にデータを変更できない場合にのみ機能します。

ファイルシステムおよびファイルシステムフィルタードライバーの場合、ユーザーモードアプリケーションと各種カーネルモードドライバーの間で情報を転送するために使用される IOCTL および FSCTL 操作は多数あります。さらに、このようなドライバーでは、ユーザーモードサービスとカーネルモードドライバーの間で制御とデータの情報を転送するプライベート IOCTL および FSCTL 操作が必要になることがよくあります。

この設計モデルに固有のものは、ドライバーのサービスまたはアプリケーションのみがインターフェイスを利用することを前提としています。このような設計と実装は、次の理由により危険にさらされます。

- 悪意のあるアプリケーションがドライバーに有効なバッファを送信し、その後、ドライバーの脆弱性を調査して検出するためにデータを変更します。
- コントロールアプリケーション内でエラーが発生すると、コントロールバッファのデータの内容が無効になる可能性があります。たとえば、例外によって制御フローが変更され、スタック領域が再利用される場合、スタックベースの制御領域が無効になることがあります。

このカテゴリの問題に対して保護するには、実装に関して一定の警戒を行う必要があります。バッファは、検証する前に、非揮発性メモリ (カーネル専用または読み取り専用メモリ) に配置する必要があります。バッファに他のバッファまたは制御構造への参照が含まれている場合は、それを検証する前に、非揮発性メモリ内に配置する必要があります。

また、開発者は、Fa生物 Odevicecontrol ディスパッチを使用する IOCTL が生のユーザーバッファにデータを渡すことにも注意する必要があります。そのため、ioctl の高速 i/o バージョンを実装するドライバーは、問題を防ぐために適切な手順を実行する必要があります。

データを単独で検証するだけでは十分ではないことに注意してください。たとえば、[ProbeForWrite](#) への呼び出しが成功した場合、バッファが有効であることを示している可能性があります。それ以降にアプリケーションアドレス空間が変更されると、その状態が変更される可能性があります。たとえば、ドライバーが実際に直接バッファを使用する前に、アプリケーションを終了させることができます。そのため、ドライバーは、アプリケーションのアドレス空間内の変更に対して保護する必要があります。通常、これは `__try` および `__` を使用した構造化例外処理を使用して実行されます。ただし、ユーザーバッファアドレスに直接アクセスするコードは除きます。

否認

2020/04/24 • [Edit Online](#)

否認不可の概念は、ユーザーが特定の操作を実行してし、その後を拒否することを実行可能性があります。ほとんどのドライバーの問題の異常の種類になります。ファイル システムでは、ただし、ログ記録に使用されます (たとえば、重要なファイルの削除) 操作を追跡し、操作の明確な痕跡があることを確認します。これは、に対してこのような否認不可を確保するためのメカニズムを提供します。

さらに、オペレーティング システムでは、特定のセキュリティ識別子にオブジェクトの所有権を割り当てることができます。適切な権限を持たない所有権情報を変更ことはできません (SeTakeOwnershipPrivilege) 特定のオブジェクトの所有権を追跡できるようにします。オブジェクトの所有権を別の形式の否認不可に対する保護を提供します。

情報の漏洩

2020/04/24 • [Edit Online](#)

ドライバースの場合、情報漏えいは通常、不適切なバッファ処理によってアプリケーションに情報を公開することに関連します。たとえば、i/o 要求がバッファリングされているドライバは、通常、*IoStatus* ブロック構造体の情報メンバーを設定することによって返されるデータの量を示します。I/o マネージャーは、その情報を使用して、結果をアプリケーションのバッファにコピーします。

ドライバースが、返されるデータが多いことを示している場合は、i/o マネージャーによって *SystemBuffer* から追加のデータがコピーされます。ただし、ドライバースが *SystemBuffer* のバランスを満たしていなかった場合、そのメモリ内のものはすべてアプリケーションに返され、アプリケーションに機密データが公開される可能性があります。この問題は、使用されるデータバッファがクリアされていないために、他のシステムに追加情報が送信される可能性があるネットワークドライバースの問題を監視します。たとえば、ICMP ping 応答に追加情報が含まれている場合があります。データを誤って公開した場合のこの問題は、さまざまなシステムで非常に実際に発生します。

ファイルシステムまたはファイルシステムフィルタードライバースの場合、データへのアクセスを許可しないユーザーにファイル情報を公開するリスクがあります。これは、さまざまな方法で行うことができます。

- [Zwcreatefile](#) を使用してファイルを開き、その中間ハンドルを介してデータへのアクセスを提供するフィルタードライバース。
[Zwcreatefile](#) 関数は、既定でファイルを開いてセキュリティチェックをバイパスします。これは、要求がカーネルモードから送信されるためです。そのため、このハンドルを使用してアクセスすると、通常はアプリケーションで使用できない情報が開示される可能性があります。

フィルタードライバースが、公開されていないデータを公開していないことを確認するためにアクセスチェックを強制する必要がある場合は、フィルタードライバースで `___OBJ` を指定して、[ZwCreateFile](#) 関数。

- カーネルモードでハンドルを開くフィルタードライバース (アクセスチェックをバイパスします)。ただし、`OBJ_KERNEL` ハンドルは指定しません。作成されたハンドルは、現在のプロセスのハンドルテーブルに配置されます。このハンドルは、データへのフルアクセスを使用して、ユーザーモードで表示されます。偽のアプリケーションは、このようなハンドルを監視して、データへのアクセスに使用することができます。
- `IRP_MJ_CREATE` 要求を送信し、システムワーカースレッドのコンテキストで処理するファイルシステムまたはファイルシステムフィルタードライバース。IRP がシステムワーカースレッドによって処理される場合、通常、作成操作はシステムスレッドの特権を使用して完了します。そのため、これらのシステム特権を使用してアクセスすると、通常はアプリケーションでは利用できない情報が開示される可能性があります。
- ファイルシステムまたはファイルシステムフィルタードライバース。呼び出し元のプロセスに特定のデータへのアクセスを許可する必要があることを確認せずに、ファイルオブジェクトベースの i/o を実行します。

情報の管理と保護には固有の役割があるため、ファイルシステムとファイルシステムフィルタードライバースは、情報の保護を確保するために特に注意する必要があります。

サービス拒否

2020/04/24 • [Edit Online](#)

サービス拒否には、特定のサービスにアクセスする必要がありますが与えられましたが、実際は拒否されました不適切な場合に発生します。たとえば、により、システムが使用できなく特権のないアプリケーションを実行できる操作は、サービス拒否では効果的にします。任意の操作または一連の操作は、これです。

- システムがクラッシュします。
- エラーの原因のスレッドやプロセスの途中で終了します。
- デッドロック状態を作成します。ステイルメイトでの永続的な状態で待機している 2 つまたは複数のスレッドが停止したときに、デッドロックが発生します。各スレッドは、いずれかの他のスレッドによって保持されているリソースを待機しています。
- ライブのロックの条件を作成します。ライブのロックは、リソース (通常、キューのロック) の取得を待機しているし、そのリソースを所有するスレッドは、同様の進行していない状態であるために、2 つ以上のプロセスは進行できないときに発生します。

このような問題は多くの場合に、通常のアプリケーションによって悪用されることがある潜在的なバグが含まれているために、ドライバー内で発生します。この種類の攻撃は、単純なことができから保護するが困難なします。ドライバーでは、このような問題の一般的な原因は次のとおりです。

- ユーザーの不適切なバッファの検証。
- バッファ オーバーフローやアンダー フロー。

ファイル システムとファイル システム フィルター ドライバーは、このような問題の多くの場合があります。たとえば、最大_パスの値は、Win32 サブシステムの歴史的な理由から 260 として定義されます。多くのドライバー コンポーネントでは、最大パス サイズに対して示すものとします。残念ながら、これは、NTFS ファイル システム パスの最大が 32,767 の Unicode 文字 (65,534 バイト)。フィルター ドライバーが、最大値をエンコードする場合_パスの長さが、そのコード ベースに想定して、フィルター ドライバーによって管理されるパス内でこれを超えるパスを作成するアプリケーションから単純なサービス拒否攻撃が発生します。

もう 1 つの一般的な問題は、アプリケーションは、プライベート FSCTL 要求にユーザー モードのポインターを多くの場合、埋め込みです。ファイル システムはサービス拒否攻撃の 3 つのカテゴリです。

- すべての使用可能なディスク領域を消費します。
- すべての使用可能なディスク帯域幅を使用します。
- ユーザーにアクセスが必要ファイルへのアクセスをブロックします。

通常、ことはほとんどありませんファイル システムの開発者は、この種の攻撃を防ぐために実行できます。ただし、管理者がこれらの種類のサービス拒否攻撃の制限を許可するために開発者が使用できる手順があります。

ファイル システムに関連する最も簡単なサービス拒否攻撃では、すべてのディスクの空き領域を使用します。これを行うアプリケーションを作成することは簡単、結果が広範囲に及び、ディスクに書き込みが長いことができない場合多くのアプリケーションとサービス、システムでは機能しません。問題を緩和するテクノロジーは、ディスク クォータは、ディスク領域の量は管理者によって適切に使用するユーザーが所有するファイルの使用を制限することができます。そのため、理にかなってファイル システムを開発するときにディスク クォータのサポートが含まれます。

すべてのディスク帯域幅を消費する悪意のある、または正しく記述アプリケーションもよい。この種の攻撃の対象の通常ユーザーの結果は、動作が遅くなったりシステムです。現時点では、オペレーティング システムでは、アプリケーションによって消費される帯域幅を調整するためのメカニズムがありません。ファイル システムでは、各ファイルを開くオブジェクトとファイル ハンドル用のカーネル メモリも消費されます。悪意のあるアプリケーションは、継続的に多数のファイルを開くし、メモリがなくなるまで開いたままにしてお試してくださいでした。これらの問題の主要な問題を緩和する技法は監査と、管理者は、大量の I/O を実行または大量の他のリソース

を使用するアプリケーションには、コンピューターを監視できるようにログ記録します。ここでも、ファイル システムを監査すると、この種類のサービス拒否攻撃を最小限にできるようにサポート対象のファイル システム フィルター ドライバー 賢くなります。

悪意のあるアプリケーションは、他のユーザーが通常使用のために必要なファイルにアクセスすることを防ぐしようとすることができません。これらの問題を最小限に抑える、重要な戦略では、ファイル システムを開発するときに、ファイル オブジェクトに関連付けられているセキュリティ情報が正しく実装されていることを確認します。

最後に、すべてのドライバーでは、使用可能なメモリまたは悪意のあるまたは異常なアプリケーションからの要求に応答には、その他のリソース消費について心配する必要があります。

特権の昇格

2020/04/24 • [Edit Online](#)

特権の昇格では、アプリケーションは、権限や特権は使用可能にならないが獲得したときに発生します。特権の昇格攻撃の多くは、その他の脅威の悪用に似ています。たとえば、バッファ オーバーラン攻撃巧妙実行可能コードを記述しようとします。これにより、ローカル変数としてスタックからバッファを割り当てるときに、x86 ベースのアーキテクチャでは動作します。スタックには、現在のプロシージャ呼び出しのリターン アドレスも含まれています。悪意のある開発者が潜在的なバッファ オーバーフローがあることを突きとめます。する場合、リターン アドレスを上書きできるようにデータは、バッファに格納できます。CPU では、前の呼び出し元に戻す「廃止」命令を実行するときは、コントロールを悪意のある開発者と実際の呼び出し元ではないで指定された場所に戻ります。

ファイル システムとファイル システム フィルター ドライバーを昇格の可能性は、特権の攻撃は、次の理由の組み合わせにより非常に高く。

- ファイル システムとファイル システム フィルター ドライバーは、特権を含む、データへのアクセスの管理にかかわるアクティブにします。
- ファイル システムとファイル システム フィルター ドライバーは、特別な特権を悪用し、アクセス権をそれらの機能を実装します。
- ファイル システムに直接関連オペレーティング システムの特権の多く (SeChangeNotifyPrivilege、たとえば、ディレクトリをスキャンする機能を制御する)。

この種類の脆弱性攻撃は、ファイル システムを実装するにとって最も重要な。このエクスプロイトは、ファイル システム フィルター ドライバー データ ストレージ (暗号化のフィルターの例) を積極的に管理している可能性がありますを回避または通常のファイル システムのセキュリティの運用をバイパスする問題になることができます。

ファイルシステムを実装してセキュリティの脅威を最小限にする

2020/04/24 • [Edit Online](#)

セキュリティの脅威となる実装の問題は、一連の一般的な問題に分類されます。

- バッファ処理します。
- 認証と識別します。
- アクセス制御。
- 管理を処理します。

特に、これらの問題の小説します。これらの問題はよく知られていない、ドライバーでこれらの問題が再び発生します。問題の一部は、既存のほとんどの開発ツールのユーザーに警告またはこの種の問題を緩和するためはありません。ただし、賢く防御的な開発手法を使用して、これらの問題のほとんど削除できます。

ここでは、次のトピックについて説明します。

[バッファ処理](#)

[認証と Id](#)

[アクセス制御](#)

[ハンドルの管理](#)

バッファ処理

2020/04/24 • [Edit Online](#)

ドライバー内の最も一般的なエラーは、バッファが無効であるか小さすぎるバッファ処理に関連している可能性があります。これらのエラーにより、バッファオーバーフローが発生したり、システムクラッシュが発生したりする可能性があります。これにより、システムのセキュリティが侵害されます。

ドライバーの観点から見ると、バッファには次の2種類があります。

- ページングされたバッファ。メモリ内に存在していても、存在しない場合もあります。
- 非ページバッファ。メモリに常駐している必要があります。

もちろん、無効なアドレスはページングも非ページでもありませんが、オペレーティングシステムは、バッファの原因であるページフォールトを解決するために動作を開始するので、無効なアドレスを "標準" アドレス範囲の1つ (ページングされたカーネルアドレス) に分離します。非ページカーネルアドレスまたはユーザーアドレス) を入力すると、適切な種類のエラーが発生します。バッファエラーが発生するのは、常にバグチェック (ページ_FAULT_(非ページ_領域)、または例外 (状態_アクセス_違反など) によって処理されます。バグチェックの場合、システムは操作を停止します。例外が発生した場合、スタックベースの例外ハンドラーが呼び出され、いずれも例外を処理しないと、バグチェックが呼び出されます。

に関係なく、ドライバーがバグチェックにつながる原因となるアプリケーションプログラムによって呼び出される可能性のあるアクセスパスは、ドライバー内でのセキュリティ違反です。これにより、アプリケーションはシステム全体に対してサービス拒否攻撃を行うことができます。

この領域における最も一般的な問題の1つは、ドライバーの作成者が運用環境について多くのことを想定していることです。次のようなことが考えられます。

- 上位ビットがアドレスに設定されていることを確認しています。これは、Boot.ini ファイルで/3GB オプションを設定することによって、システムが 4 Gb のチューニング (4GT) を使用している x86 ベースのコンピューターでは機能しません。この場合、ユーザーモードアドレスによって、アドレス空間の3ギガバイト (GB) の上位ビットが設定されます。

- **ProbeForRead**と**ProbeForWrite**を使用してアドレスを検証します。これにより、プローブ時のアドレスが有効なユーザーモードアドレスであることが確認されますが、プローブ操作後も有効なままにする必要があるものではありません。そのため、この手法では、定期的に再現可能なクラッシュにつながる可能性がある微妙な競合状態が導入されます。**ProbeForRead**と**ProbeForWrite**の呼び出しは、さまざまな理由で必要になります。アドレスがユーザーモードアドレスであり、バッファの長さがユーザーアドレス範囲内であるかどうかを検証するためです。プローブが省略されている場合、ユーザーは有効なカーネルモードアドレスを渡すことができます。これは `_` でキャッチされません。`_try` と `_` (構造化例外処理) を除き、`_` は大規模なセキュリティホールを開きます。したがって、**ProbeForRead**と**ProbeForWrite**の呼び出しはアラインメントを保証するために必要であり、ユーザーモードアドレスと長さを組み合わせて、ユーザーアドレスの範囲内にあることを確認する必要があります。ただし、アクセスを保護するためにブロックが必要であることを除いて、`_try` と `_`。

ProbeForReadでは、メモリアドレスが有効であるかどうかにかかわらず、使用可能なユーザーモードアドレス範囲 (たとえば、4gt を使用しないシステムでは 2 GB 未満) にアドレスと長さが含まれていることのみが検証されることに注意してください。これに対して、**ProbeForWrite**は、指定された長さの各ページの最初のバイトへのアクセスを試み、これらが有効なメモリアドレスであることを確認します。

- アドレスが有効であることを確認するために、メモリマネージャー関数 (たとえば、**M誤 Addressvalid**) に依存します。プローブ関数と同様に、これにより、再現性可能なクラッシュにつながる可能性がある競合状態が導入されます。
- 構造化例外処理を使用できません。`_try` および `_` は、コンパイラ内の関数を除き、例外処理にオペレーティングシステムレベルのサポートを使用します。カーネルレベルでの例外は、**ExRaiseStatus**または関連する関数のいずれかを呼び出すことによってスローされます。例外を発生させる可能性のある呼び出しに対して構造化例外処理を使用しないドライバーでは、バグチェックが発生します (通常、`KMODE_例外_` 処理されません)。

エラーを発生させることが想定されていないコードの周囲に構造化例外処理を使用することは誤りであることに注意してください。これは、実際に検出されるバグをマスクするだけです。__使用する__と、ルーチンの最上位ディスパッチレベルのラッパーを除き、この問題に対する適切な解決策ではありませんが、これはリフレックスソリューションがドライバーライターによって試行されることもあります。

- 残りのユーザーメモリの内容に依存しています。たとえば、ドライバーがユーザーモードのメモリ位置に値を書き込み、後で同じルーチンでそのメモリ位置を参照しているとします。悪意のあるアプリケーションによってそのメモリが積極的に変更され、その結果、ドライバーがクラッシュする可能性があります。

ファイルシステムの場合、これらの問題は、通常、ユーザーバッファーに直接アクセスすることに依存しているため、特に深刻です (メソッド_転送方法はありません)。このようなドライバーはユーザーバッファーを直接操作するため、オペレーティングシステムレベルのクラッシュを防ぐために、バッファー処理のための予防メソッドを組み込む必要があります。高速 i/o では常に生のメモリポインターが渡されるため、高速 i/o がサポートされている場合、ドライバーは同様の問題から保護する必要があります。

WDK には、FASTFAT および CDFS ファイルシステムのサンプルコードで、次のようなバッファー検証の例が多数含まれています。

- Fastfat\deviosup のFatLockUserBuffer関数は、[MmProbeAndLockPages](#)を使用してユーザーバッファーの背後にある物理ページをロックダウンし、MmGetSystemAddressForMdlSafeのFatMapUserBufferを使用して仮想マシンを作成します。ロックダウンされているページのマッピング。
- Fastfat\fsctl のFat Getvolumebitmap関数は、[ProbeForRead](#)と[ProbeForWrite](#)を使用して、最適化 API のユーザーバッファーを検証します。
- Cdfs\read.c では、Cdcommonread関数は__を使用します。c では、コードを除いてユーザーバッファーをゼロにして__tryと__を__します。Cdcommonreadのサンプルコードは、try キーワードと except キーワードを使用していることに注意してください。WDK 環境では、C のこれらのキーワードは、__tryと__を除き、コンパイラの拡張機能__定義されています。コードをC++使用するすべてのユーザーは、例外を適切に処理するためにネイティブコンパイラ型C++を使用する必要があります。__tryはキーワードですC++が、C キーワードではなく、カーネルドライバーでは無効な例外処理の形式を提供します。

認証と ID

2020/04/24 • [Edit Online](#)

認証または id の問題、個々のサービスにこのタスクのままでは、ほとんどのドライバーは関与しません。アクセスの管理は 1 つのケースを認証または id の処理でドライバーを関係になる可能性があります。この場合、認証手順は、通常、セキュリティ参照モニターへの呼び出しを処理します。認証と id の情報は、セキュリティトークンを指定したスレッドまたはプロセスのセキュリティ資格情報をカプセル化する内部データ構造で通常、オペレーティングシステムによって追跡されます。

ドライバーのアクセス制御

2020/08/04 • [Edit Online](#)

不適切なアクセスから保護するために、ほとんどのドライバーは、デバイスオブジェクトに対して i/o マネージャーによって適用される既定のアクセス制御に依存しています。ドライバーでは、他のメカニズムを使用できます。通常は、ドライバーをインストールするときに、明示的なセキュリティ記述子を適用するのが最も簡単です。デバイスオブジェクトにセキュリティ記述子を適用する例については、後のセクションで説明します。

独自のセキュリティポリシーを実装するドライバーは、標準の Windows Api に依存して、セキュリティアクセスの管理を支援することができます。この場合、ドライバーはセキュリティ記述子の格納を管理し、セキュリティを検証するためにセキュリティ参照モニタールーチンを呼び出します。これには、次のような多くのルーチンが含まれます。

- **Seaccesscheck**--このルーチンは、セキュリティ記述子を呼び出し元のセキュリティ資格情報と比較します。
- **SePrivilegeCheck**--このルーチンは、指定された特権が呼び出し元に対して有効になっているかどうかを判断します。
- **SeSinglePrivilegeCheck**--このルーチンは、呼び出し元に対して特定の特権が有効になっているかどうかを判断します。
- **SeAuditingFileOrGlobalEvents**--このルーチンは、システムで監査が有効になっているかどうかを示します。
- **SeOpenObjectAuditAlarm**--このルーチンは、開いているオブジェクトイベントを監査します。

この一覧は完全ではありませんが、アクセス検証を実行するためにドライバー内で使用できる主な機能をいくつか説明しています。

ハンドルの管理

2020/04/24 • [Edit Online](#)

ドライバー内のセキュリティ問題の重要な原因として、ユーザーモードコンポーネントとカーネルモードコンポーネントの間で渡されるハンドルの使用があります。カーネル環境内のハンドルの使用には、次のような既知の問題がいくつかあります。

- 間違った種類のハンドルをカーネルドライバーに渡すアプリケーション。カーネルドライバーは、ファイルオブジェクトが必要なイベントオブジェクトを使用しようとするとクラッシュすることがあります。
- 必要なアクセス権を持たないオブジェクトをハンドルに渡すアプリケーション。カーネルドライバーは、ユーザーに適切なアクセス許可が与えられていなくても、カーネルモードからの呼び出しによって動作する操作を実行することがあります。
- アドレス空間に有効なハンドルではない値を渡し、システムに対して悪意のある操作を実行するシステムハンドルとしてマークされているアプリケーション。
- デバイスオブジェクト (このドライバーによって作成されなかったハンドル) の適切なハンドルではない値を渡すアプリケーション。

これらの問題から保護するために、カーネルドライバーは、渡されたハンドルが有効であることを確認するために特に注意する必要があります。最も安全なポリシーは、ドライバーのコンテキスト内で必要なハンドルを作成することです。カーネルドライバーによって作成されるこれらのハンドルは、OBJ_KERNEL_HANDLE オプションを指定する必要があります。これにより、任意のプロセスコンテキストで有効なハンドルと、カーネルモードの呼び出し元からのみアクセスできるハンドルが作成されます。

アプリケーションプログラムによって作成されたハンドルを使用するドライバーの場合、これらのハンドルの使用は細心の注意を払って行う必要があります。

- 適切な *Accessmode* (通常は *Irql->irp->requestormode*)、*DesiredAccess*、*ObjectType*を指定して、[obreferenceobjectbyhandle](#)を呼び出してハンドルをオブジェクトポインターに変換することをお勧めします。
IoFileObjectType や *ExEventObjectType* などのパラメーター。
- 呼び出し内で直接ハンドルを使用する必要がある場合は、関数の Zw variant ではなく、Nt バリエーションの関数を使用することをお勧めします。これにより、前のモードがモードになり、信頼されていないため、オペレーティングシステムによってパラメーターチェックが行われ、検証が処理されます。ポインターである Nt 関数に渡されるパラメーターは、前のモードがモードの場合、検証に失敗することがあります。Nt ルーチンと Zw ルーチンは、エラーをチェックする必要があるエラー情報を含む *IoStatusBlock* parameter with 返します。
- __使用してエラーを適切にトラップし、必要な場合を除き__する必要があります。キャッシュマネージャー (Cc)、メモリマネージャー (Mm)、およびファイルシステムランタイムライブラリルーチン (FsRtl) の多くは、エラーが発生した場合に例外を発生させます。

適切な予防措置を取らずに、ユーザーモードアプリケーションから渡されたハンドルやパラメーターに依存するドライバーはありません。

Nt バリエーションを使用してファイルを開く場合は、ファイルを閉じるために Nt バリエーションも使用する必要があることに注意してください。

セキュリティのテスト

2020/04/24 • [Edit Online](#)

セキュリティのテストは、自動化されたプロセスではありません。代わりに、既存のツールの使用と、特定のドライバーまたはドライバーの徹底的な脅威の分析が組み合わされています。ドライバーのテストは、多くの独立した手順のためから成ります。

- アクティブにドライバーが使用されている環境内で行われる可能性のある攻撃の種類を識別するために徹底的な脅威分析します。たとえばの高度に制御された環境に存在するドライバーには、任意の攻撃の対象となる大容量の配布用のドライバーよりも単純な脅威の分析があります。
- デバイスパスの Exerciser など、WDK で"Windows 用に設計されています"のロゴ テストを含む既存のツールを使用してテスト デバイスは、ユーティリティをテストする攻撃を制御します。WDK に IFS テストは、記憶域スタックの問題を徹底的にテストすることも必要があります。
- 追加のテストは、具体的には、脅威の分析の一部として識別されるシナリオをテストするための通常のドライバーの開発の一環として開発する必要があります。これらのテストは、通常ドライバーとプローブの特定のドライバーを一意になります。

理想的には、テスト目的で、検証テストの開発が含まれます、ソフトウェアと関連のない開発リソースの特定の種類のファイル システムまたはファイル システム フィルター ドライバー製品を使い慣れて元デザイナーからの入力には開発、および 1 つまたは複数のユーザー セキュリティ侵入の分析と防止に精通します。

ファイル システム向けのセキュリティ機能

2020/04/24 • [Edit Online](#)

その他のほとんどの種類のドライバーとは異なりファイル システムは密接、通常のセキュリティの処理に関係します。これは、セキュリティと Microsoft Windows 内での実装の性質のためです。一般的な Windows セキュリティ モデルは、セキュリティ記述子オブジェクトに関連付けます、--この場合、ファイル_オブジェクト。Windows セキュリティをサポートするファイル システムは、格納して、セキュリティ記述子を取得する責任を負います。さらに、ファイル システムは、標準のカーネル モード ドライバーの通常のスコープ外にあるその他のいくつかの特別なセキュリティ考慮事項に対応します。

このセクションでは、Windows セキュリティをサポートするためにファイル システムに追加することがありますを主な機能について説明します。これらの [なし] は必須で、これらのインターフェイスを使用せずにファイル システムを構築できます。無視しているときにいくつかのセキュリティ機能を実装することはさらに、他のユーザー - これは、ファイル システムの実装に固有です。

ここでは、次のトピックについて説明します。

[セキュリティ記述子](#)

[権限](#)

[監査](#)

[拡張属性のカーネル](#)

ファイルシステムのセキュリティ記述子

2020/08/04 • [Edit Online](#)

Windows ファイルシステムでは、ファイルシステム内の個々のストレージユニットに関連付けられた[セキュリティ記述子](#)の保存と管理がサポートされている場合があります。セキュリティ制御の粒度は、完全にファイルシステムに任されています。たとえば、あるファイルシステムでは、特定の記憶域ボリューム上のすべてを対象とする1つのセキュリティ記述子を保持しながら、1つの特定のファイルのさまざまな部分をカバーするセキュリティ記述子を提供する場合があります。ほとんどの開発者が使い慣れているモデルは、既存の Windows ファイルシステムによって提供されるモデルです。

- NTFS では、ファイルごと (またはディレクトリ) のセキュリティ記述子モデルがサポートされています。NTFS はセキュリティ記述子の格納に効率的であり、さまざまなファイルで使用されている場合でも、各セキュリティ記述子のコピーを1つだけ格納します。
- FAT、CDFS、UDF では、セキュリティ記述子はサポートされていません。
- RDBSS と SMB ネットワークリダイレクターは、リモートボリュームによって提供されるものと同等のサポートを提供します。

ただし、これらのファイルシステムは、ファイルシステムに対する Windows セキュリティのすべての実装を表すとは限りません。

Windows セキュリティ記述子は、次の4つの部分で構成されます。

- オブジェクトの所有者のセキュリティ識別子 (SID)。オブジェクトの所有者は、常にオブジェクトのセキュリティをリセットできます。これは、たとえば、オブジェクトへのすべてのアクセスを削除できるようにするための適切な方法です。所有者がすべての操作を実行する機能を削除したとしても、この固有の権限によって、オブジェクトに対するセキュリティ権限を復元できます。
- オブジェクトの既定のグループのオプションのセキュリティ識別子 (SID)。グループ所有権の概念は、Windows では必要ではありませんが、一部のアプリケーションでは便利です。
- セキュリティ記述子の監査ポリシーを記述するシステムアクセス制御リスト (SACL)。
- セキュリティ記述子のアクセスポリシーを記述する随意アクセス制御リスト (DACL)。

次の図は、windows セキュリティ記述子を示しています。

Control	Reserved	Revision
Owner		
Group		
SACL		
DACL		

セキュリティ記述子は可変サイズのオブジェクトであり、個々のサブコンポーネントのサイズも変数になります。セキュリティ記述子のオフラインストレージを容易にするために、セキュリティ記述子は自己相対形式にすることができます。この場合、ヘッダーは、セキュリティ記述子の特定のコンポーネントに対するバッファー内のオフセットです。インメモリ形式は、セキュリティ記述子のさまざまな部分へのポインター値で構成されます。ファイルシステムの場合、通常は自己相対形式が最も便利です。これは、永続的なストレージからセキュリティ記述子を簡単に保存および取得できるようにするためです。セキュリティ記述子を構築するアプリケーションは、メモリ内の形式を使用する可能性が高くなります。セキュリティ参照モニターには、ある形式から別の形式に変換する変換ルーチンが用意されています。

ここでは、次のトピックについて説明します。

[セキュリティ識別子](#)

[アクセス マスク](#)

[アクセス制御エントリ](#)

セキュリティ識別子

2020/04/24 • [Edit Online](#)

セキュリティ識別子は、セキュリティエンティティどうしを区別するための明確な値として Windows によって使用されます。たとえば、システムの個々のユーザーに対して作成された新しいアカウントごとに、一意のセキュリティ識別子が割り当てられます。ファイルシステムの場合は、この SID だけが実際に使用されます。

次の図は、セキュリティ識別子の構造を示しています。

Identifier Authority	SubAuthorityCount	Revision
SubAuthority (Variable Array)		

一意の Sid に加えて、Windows システムは既知の一連の識別子を定義します。たとえば、ローカル管理者はよく知られた SID です。Windows には、カーネル環境内の Sid とユーザー名を変換するためのカーネル内メカニズムが用意されています。これらの関数呼び出しは、ksecdd ドライバーから使用できます。これらの関数は、ユーザーモードヘルパーサービスを使用してこれらの関数を実装します。したがって、ファイルシステム内での使用は、ユーザーモードサービスとの通信に関する通常の規則に従う必要があります。これらの呼び出しは、ページングファイル i/o 中には使用できません。

関数には、次のようなものがあります。

- **SecMakeSPN**—特定のセキュリティサービスプロバイダーと通信するときに使用できるサービスプロバイダー名文字列を作成します。
- **SecMakeSPNEx**— SecMakeSPN の拡張バージョンです。この機能は、Microsoft Windows XP 以降のバージョンの Windows で使用できます。
- **SecMakeSPNEx2**— SecMakeSPNEx の拡張バージョンです。この機能は、windows Vista、Windows Server 2008、およびそれ以降のバージョンの Windows で使用できます。
- **SecLookupAccountSid**— SID が指定されている場合、このルーチンはアカウント名を返します。この機能は、Windows XP 以降で使用できます。
- **SecLookupAccountName**—アカウント名を指定すると、このルーチンは SID を取得します。この機能は、Windows XP 以降で使用できます。
- **SecLookupWellKnownSid**—既知の sid の種類を指定した場合、このルーチンは正しい sid を返します。この機能は、Windows Server 2003 以降で使用できます。

また、次の標準ランタイムライブラリルーチンを使用して、すべてのカーネルドライバーで SID を作成することもできます。

- **Rtlinitializesid**—新しい SID のバッファーを初期化します。
- **RtlLengthSid**—指定したバッファー内に格納されている SID のサイズを決定します。
- **Rtlvalidsid**—指定した SID バッファーが有効な形式のバッファーであるかどうかを判断します。

RtlLengthSid と RTLVALIDSID は、SID の 8 バイトの固定ヘッダーが存在すると想定しています。そのため、ドライバーは、これらの関数を呼び出す前に、SID ヘッダーの最小の長さを確認する必要があります。

他にもいくつかの RTL 関数がありますが、これらは SID を構築するときに必要な主な機能です。

次のコード例は、"local system" エンティティの SID を作成する方法を示しています。

```

{
    //
    // temporary stack-based storage for an SID
    //
    UCHAR sidBuffer[128];
    PISID localSid = (PISID) sidBuffer;
    SID_IDENTIFIER_AUTHORITY localSidAuthority =
        SECURITY_NT_AUTHORITY;

    //
    // build the local system SID
    //
    RtlZeroMemory(sidBuffer, sizeof(sidBuffer));

    localSid->Revision = SID_REVISION;
    localSid->SubAuthorityCount = 1;
    localSid->IdentifierAuthority = localSidAuthority;
    localSid->SubAuthority[0] = SECURITY_LOCAL_SYSTEM_RID;

    //
    // make sure it is valid
    //
    if (!RtlValidSid(localSid)) {
        DbgPrint("no dice - SID is invalid\n");
        return(1);
    }
}

```

これは、Windows Server 2003 で導入されたより単純な関数 `SecLookupWellKnownSid` を使用して行うこともできます。

次のコード例は、"local system" エンティティに対して `SecLookupWellKnownSid` 関数を使用して SID を作成する方法を示しています。

```

{
    UCHAR sidBuffer[128];
    PISID localSid = (PISID) sidBuffer;
    SIZE_T sidSize;
    status = SecLookupWellKnownSid(WinLocalSid,
                                   &localSid,
                                   sizeof(sidBuffer),
                                   &sidSize);

    if (!NT_SUCCESS(status)) {
        //
        // error handling
        //
    }
}

```

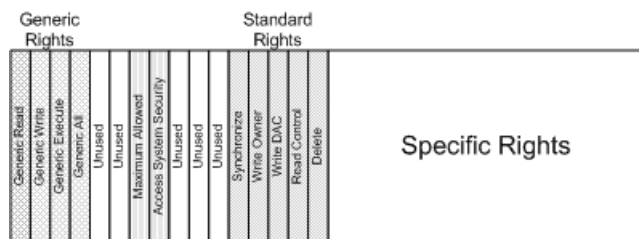
どちらの方法も有効ですが、後者のコードはお勧めします。これらのコード例では、SID を格納するためにローカルバッファを使用することに注意してください。これらのバッファは、現在の呼び出しコンテキストの外部では使用できません。SID バッファを永続化する必要がある場合は、バッファをプールメモリから割り当てる必要があります。

アクセスマスク

2020/04/24 • [Edit Online](#)

アクセスマスクの機能は、アクセス権をコンパクトな形式で記述することです。アクセスの管理を簡略化するために、アクセスマスクには4つのビットのセットが含まれています。汎用的な権限は、関数 `Rtlmapgenericmask` を使用してより詳細な権限のセットに変換されます。

次の図は、アクセスマスクを示しています。



一般権限は、次のいずれかになります。

- 汎用_読み取り: オブジェクトによって保持されている情報を読み取る権限。
- 汎用_書き込み—オブジェクトによって保持されている情報を書き込む権限。
- 汎用_EXECUTE: オブジェクトを実行する権限、またはオブジェクトを検索する権限です。
- GENERIC_ALL: オブジェクトの読み取り、書き込み、および実行の権限。

これらの権限を組み合わせることができることに注意してください (汎用_読み取りと汎用_書き込みの両方を要求することができます)。結果のマッピングでは、各汎用権限に必要な権限の和集合が要求されます。このパラダイムは、unix のリソースへのアクセスを制御するために使用される UNIX の "rwx" アクセスビットを模倣します。アクセスマスクの汎用的な権限により、Windows でのアプリケーション開発が簡略化されます。これらの権限は、さまざまな種類のオブジェクトに対して異なるセキュリティ権限をマスクするためです。

次の一連の標準権限は、すべてのオブジェクトの種類に適用できます。

- DELETE —特定のオブジェクトを削除する権限。
- 読み取り_コントロール-オブジェクトのコントロール (セキュリティ) 情報を読み取る権限。
- 書き込み_DAC: オブジェクトのコントロール (セキュリティ) 情報を変更する権限。
- 書き込み_所有者—オブジェクトの所有者 SID を変更する権限。所有者は常にオブジェクトを変更する権限を持っていることを思い出してください。
- SYNCHRONIZE: 指定されたオブジェクトを待機する権限です (オブジェクトの有効な概念であることを前提としています)。

アクセスマスクの下位16ビットは、特定の権限を表します。これらの特定の権限の意味は、対象のオブジェクトに固有のものであります。ファイルシステムの場合は、ファイルオブジェクトに対する特定の権限が主な対象となります。ファイルオブジェクトの場合、通常、特定の権限は、ファイルオブジェクトがファイルまたはディレクトリを表しているかどうかによって異なる方法で解釈されます。ファイルの場合、通常、解釈は次のようになります。

- ファイル_読み取り_データ-指定したファイルからデータを読み取る権限。
- ファイル_ は、指定されたファイル (ファイルの既存の範囲内) にデータを書き込む権限である_データを書き込みます。
- ファイル_ は、指定されたファイルを拡張する権限_データを追加します。

- **ファイル_読み取り_EA**-ファイルの拡張属性を読み取る権限。
- **ファイル__EA**-ファイルの拡張属性を変更する権限。
- **ファイル_実行**-指定されたファイルをローカルで実行する権限。リモート共有に格納されているファイルを実行するには、読み取りアクセス許可が必要です。これは、ファイルがサーバーから読み取られ、クライアントで実行されるためです。
- **ファイル_読み取り_属性**-ファイルの属性情報を読み取る権限。
- **ファイル_** は、ファイルの属性情報を変更する権限を持つ、_**の属性を書き込みます。**

ディレクトリの場合、同じビット値が使用されますが、その解釈は次のような場合に異なります。

- **[ファイル_一覧_ディレクトリ]** -ディレクトリの内容を一覧表示する権限です。
- **ファイル__ファイルを追加します。**これは、ディレクトリ内に新しいファイルを作成する権限です。
- **ファイル__サブディレクトリ** (ディレクトリ内に新しいディレクトリ (サブディレクトリ) を作成する権限) を追加します。
- **ファイル_読み取り_EA**-指定されたディレクトリの拡張属性を読み取る権限。
- **ファイル_**は、指定されたディレクトリの拡張属性を書き込む権利である**EA_書き込みます。**
- **ファイル_走査**-ディレクトリ内のオブジェクトにアクセスする権限。ファイル_走査アクセス権は、ファイル_リスト_ディレクトリアクセス権限とは異なります。ファイル_一覧_ディレクトリアクセス権を保持すると、エンティティはディレクトリの内容の一覧を取得できます。一方、ファイル_走査アクセス権限では、オブジェクトにアクセスする権限がエンティティに与えられます。ファイル_リスト_ディレクトリアクセス権のない呼び出し元は、既に存在していたファイルを開くことができますが、ディレクトリの内容の一覧を取得することはできません。
- **ファイル__子を削除する:** 現在のディレクトリ内のファイルまたはディレクトリを削除する権限です。
- **ファイル_読み取り_属性:** ディレクトリの属性情報を読み取る権限。
- **ファイル_** は、ディレクトリの属性情報を変更する権限である、_**の属性を書き込みます。**

一般権限とファイルオブジェクトの特定の権限の実際のマッピングは、i/o マネージャーによって定義されます。このマッピングは、ファイルシステムによって **IoGetFileObjectGenericMapping** を使用して取得できます。通常、このマッピングは、ファイルシステムを呼び出す前に i/o マネージャーによる処理を作成_IRP_MJ_中に行われます。ただし、これは特定の操作 (特化された FSCTL 操作など) のセキュリティをチェックするファイルシステムによって行われる可能性があります。

アクセス制御エントリ

2020/04/24 • [Edit Online](#)

アクセス制御エントリ (ACE) は、特定の SID に関連付けられているアクセス権を記述します。アクセス制御エントリは、資格情報に基づいて特定のプログラムに与えられた有効なアクセスを計算するために、オペレーティングシステムによって評価されます。たとえば、ユーザーがコンピューターにログオンしてからプログラムを実行すると、その特定のユーザーのアカウントに関連付けられている資格情報がプログラムによって使用されます。

このため、プログラムがオブジェクトを開こうとすると、Windows は、そのプログラムに関連付けられている資格情報を、オブジェクトに関連付けられているセキュリティコントロールと比較します。次に、セキュリティ参照モジュールが ACE 情報を使用して、指定されたオブジェクトへのアクセスを許可するか拒否するかを判断します。このため、ACE によってセキュリティサブシステムの動作が決定されます。

次の図は、アクセス制御エントリを示しています。

ACE Size	ACE Flags	ACE Type
ACCESS_MASK		
Security Identifier		

セキュリティサブシステムでは、5種類の Ace が使用されています。Ace 構造体の型メンバーは、ace の解釈を制御します。定義されている型は次のとおりです。

- **アクセス_許可_ace_の種類**: この種類は、ace が特定の SID に付与されるアクセス権を指定することを示します。
- **アクセス_拒否_ace_の種類**: この種類は、ace が特定の SID に対して拒否するアクセス権を指定していることを示します。
- **SYSTEM_AUDIT_ace_type**—この型は、ace によって監査の動作が指定されていることを示します。
- **システム_アラーム_ace_の種類**—この種類は、ace がアラーム動作を指定することを示します。
- **アクセス_、複合_ace_type_許可**されます。この型は、ace が特定のサーバーと、それが偽装しているエンティティに関連付けられていることを示します。

そのため、オブジェクトへのプログラムによるアクセスを制御するために、3つの型が使用されています。一方、オブジェクトにアクセスしたときに、セキュリティサブシステムの監査とアラームの動作を制御するために使用されます。セキュリティサブシステムの実際の動作は、オブジェクトに関連付けられている一部またはすべての Ace の情報を組み合わせて計算されることに注意してください。

ドライバーは、ルーチン [Rtladdaccessallowedace](#) を使用して ACE_型のアクセス_許可されるアクセス制御エントリを作成できます。他の種類の ACE エントリを追加するには、WDK に他のサポートルーチンが用意されていないため、ドライバーの作成者は独自の機能を構築する必要があります。

アクセス制御リスト

2020/04/24 • [Edit Online](#)

アクセス制御リスト (ACL) は、何らかの (保護された) 特定のオブジェクトに関連付けられているセキュリティ動作を制御するオペレーティング システムによって作成された Ace のリストです。Windows では、Acl の 2 つの種類があります。

- **随意 ACL**-これは、保護されているオブジェクトのアクセス権を表す 0 個以上の Ace のリスト。アクセスが、所有者または適切な権限を持つすべてのユーザーの判断では許可されているために、随意です。
- **システム ACL**-これは、監査をについて説明し、保護されているオブジェクトのポリシーのアラームを 0 個以上の Ace のリスト。

「任意」という用語は、必須および任意のコントロールの違いを指します。必須のコントロールを使用する環境でオブジェクトの所有者のオブジェクトへのアクセスを許可していない場合があります。任意の環境で、Windows など、オブジェクトの所有者は許可されてこのようなアクセスを許可します。必須のコントロールは、通常、区画を持つセキュリティは、システムが同じシステム上のユーザー間で機密情報の漏えいを防ぐ必要がありますを使用している、非常に強固なセキュリティ環境に関連付けられます。

ACL を作成するドライバーは、いくつかの主要な手順に従ってください。

1. ACL の記憶域を割り当てます。
2. ACL を初期化します。
3. ACL には、0 (または複数) の Ace を追加します。

次のコード例では、ACL を作成する方法を示します。

```
dacl = ExAllocatePool(PagedPool, PAGE_SIZE);
if (!dacl) {
    return;
}
status = RtlCreateAcl(dacl, PAGE_SIZE, ACL_REVISION);
if (!NT_SUCCESS(status)) {
    ExFreePool(dacl);
    return;
}
```

前のコードフラグメントでは、空の ACL を作成します。コード サンプルでは、ACL の必要なサイズがわからないために、大量のメモリが割り当てられます。

この時点では、ACE のエントリがあるないためには、ACL が空です。空の ACL は、このようなアクセス権を付与するエントリがないため、オブジェクトにアクセスしようとしています。すべてのユーザー アクセスを拒否します。次のコード フラグメントでは、この ACL に ACE を追加します。

```
status = RtlAddAccessAllowedAce(dacl, ACL_REVISION, FILE_ALL_ACCESS, SeExports->SeWorldSid);
if (!NT_SUCCESS(status)) {
    ExFreePool(dacl);
    return;
}
```

このエントリは、オブジェクトにアクセスするすべてのエンティティへのアクセスに与えるようになりました。これは、他の Windows システム ユーティリティで "Everyone" アクセスと、通常、表される SID (SeWorldSid) アクセスの目的です。

Acl を作成するときに、順序へのアクセスが拒否 ACE エントリ、ACL の先頭にはことに注意してくださいし、アクセス許可 ACL の末尾に ACE エントリ。これは、セキュリティ参照モニターが、ACL の評価が本物でアクセス拒否 Ace が見つかる前に、アクセスを許可

する ACE が見つかるかどうかがあるためにです。この動作が、Microsoft Windows SDK にも記載されているが、セキュリティ参照モニターを使用してアクセスを許可または拒否するかどうかを決定する特定のメカニズムに関連します。

ファイルシステムでの特権の管理

2020/08/04 • [Edit Online](#)

特権は、特定の操作を制御するためにオペレーティングシステムによって使用される完全に独立したメカニズムです。各特権は、呼び出し元によって特権が保持され、有効になった場合に実行される可能性のある特定の操作に関連付けられています。ここでは、次の2つの条件に注意してください。

- 特権は、呼び出し元によって保持されている必要があります。
- 特権も有効にする必要があります。

ここでは、最小限の特権と呼ばれる原則で、意図していなかった操作をユーザーが意図せずに実行する可能性を最小限に抑えるために、単に想定するのではなく、使用する前に特権を有効にする必要があります。たとえば、**SeRestorePrivilege**は通常、ファイルへの書き込みアクセスの通常のチェックを呼び出し元がバイパスできるようにします。管理者は、ファイルをコピーするとき通常のセキュリティチェックを無効にしたい場合がありますが、バックアップ/復元ユーティリティを使用して同じファイルを復元する場合には、この操作を行うことをお勧めします。

ファイルシステムの場合、システムの通常の動作 (特にセキュリティチェック) を変更するためによく使用される特権が多数あります。これらの特権は次のとおりです。

- **SeBackupPrivilege**は、ファイルのセキュリティ記述子がこのようなアクセスを許可しない場合でも、ファイルコンテンツの取得を許可します。**SeBackupPrivilege**が有効になっている呼び出し元は、ACL ベースのセキュリティチェックをなくなり必要があります。
- **SeRestorePrivilege**では、ファイルのセキュリティ記述子によってアクセスが許可されない場合でも、ファイルの内容を変更できます。この関数を使用して、所有者と保護を変更することもできます。
- **SeChangeNotifyPrivilege**では、走査権限を許可します。パス内のすべてのディレクトリに対してセキュリティチェックを実行する場合のコストは、この特権を保持することによって除外されるため、この特権は Windows での重要な最適化です。
- **SeManageVolumePrivilege**を使用すると、ボリュームレベルの管理操作 (ボリュームのロック、デフラグ、ボリュームのマウント解除、Windows XP 以降での有効なデータの長さの設定など) を行うことができます。この特定の特権は、主に FSCTL 操作に基づいてファイルシステムドライバによって明示的に適用されることに注意してください。この場合、ファイルシステムによって、この特権を適用するポリシーが決定されます。この特権が呼び出し元によって保持されるかどうかは、通常の特権チェックの一部として、セキュリティ参照モニターによって決定されます。

他にも多くの特権がありますが、通常はファイルシステムに対して非透過的であるため、セキュリティ参照モニターによってのみ解釈されます。

ファイルシステム内の特権を管理するための主要な Windows ルーチンは次のとおりです。

- **SePrivilegeCheck**--このルーチンは、必要な特権の特定のセットのチェックを実行します。
- **SeSinglePrivilegeCheck**--このルーチンは、1つの特定の特権のチェックを実行します。これは、**SePrivilegeCheck**の最適化されたバージョンです。
- **SeAccessCheck**--このルーチンは、オブジェクト (通常はファイルシステムのファイルオブジェクト) に対する通常のアクセスチェックを実行します。
- **SeFreePrivileges**--このルーチンは、**seaccesscheck**の前の呼び出しで返された特権ブロックを解放します。
- **SeAppendPrivileges**--このルーチンは、有効な特権を ACCESS_STATE 構造に追加します。通常、ファイルシステムは、IRP_MJ_CREATE 処理中に渡された ACCESS_STATE を使用します。

監査

2020/04/24 • [Edit Online](#)

優れたセキュリティ設計の原則の1つは、セキュリティで保護されたシステムのようなものが存在しないことを認めることです。システムの開発者は、すべてのセキュリティを回避することを意識する必要があります。これは、たとえば、セキュリティサブシステムを調査してシステム内のホールを検出し、悪用することによって、アクティブに実行できます。または、誤って重要なデータを上書きしたり削除したりすることがあります。原因にかかわらず、このような侵害を検出できるシステムを構築することが不可欠です。

Windows 内の監査システムは、特定のセキュリティイベントを追跡するためのメカニズムを提供します。これにより、後でログを分析して、破損または侵害されたシステムの事後分析を実行できます。Windows 深くの監査メカニズムには、ファイルシステムがシステムデータの永続的なストレージを維持する役割を果たすため、ファイルシステムが関係します。多くのシステムでは、セキュリティの必要性が大幅に低下し、その場合は監査が無効になっています。ファイルシステムは、両方の環境の問題に対処できるように実装する必要があります。

監査の主要なルーチンは次のとおりです。

- **SeAuditingFileEvents**--このルーチンは、システムでファイル監査が有効になっているかどうかを判断します。これは、完全な監査チェックを実行する必要があるかどうかを判断するためのグローバルポリシーチェックです。このルーチンは、セキュリティシステムの操作を最適化するために導入されました。
- **SeOpenObjectAuditAlarm**--このルーチンは、Windows システムでプライマリ監査操作を実行します (オブジェクトを開く試行を監査します)。オブジェクトへのアクセスが成功したか失敗したかにかかわらず、監査対象のオブジェクトにアクセスしようとしたことに注意してください。

監査の要件はありません。WDK 実装監査の IFS セクションにあるサンプルファイルシステム (FAT や CDFS など) はありません。ただし、セキュリティの観点からは、管理者がシステムのセキュリティ動作を監視できるようにするため、監査が重要になります。

カーネル拡張属性

2020/04/24 • [Edit Online](#)

カーネル拡張属性 (カーネル EA) は、イメージファイル署名の検証のパフォーマンスを向上させる方法として、Windows 8 の NTFS に追加された機能です。画像署名を検証するためのコストが高い操作です。したがって、以前に検証されたバイナリが変更されたかどうかに関する情報を格納すると、イメージが完全な署名チェックを行う必要があるインスタンスの数が減少します。

概要

名前プレフィックス `$Kernel` を持つ EA は、カーネルモードからのみ変更できます。この文字列で始まるすべての EA は、カーネル EA と見なされます。必要な Update Sequence Number (USN) を取得する前に、最初に発行することを `FSCTL_WRITE_USN_CLOSE_RECORD` お勧めします。これにより、前に発生した可能性があるファイルに対する保留中の USN ジャーナル更新がコミットされます。これを行わないと、カーネル EA の設定直後に `Fileusn` 値が変更される可能性があります。

カーネル EA には、少なくとも次の情報が含まれていることをお勧めします。

- USN UsnJournalID
 - `UsnJournalID` フィールドは、USN ジャーナルファイルの現在の部分を識別する GUID です。USN ジャーナルは、ボリュームごとにユーザーモードから削除および作成できます。USN ジャーナルが作成されるたびに、新しい `UsnJournalID` GUID が生成されます。このフィールドでは、USN ジャーナルが無効になっていて再検証できる期間があるかどうかを確認できます。
 - この値は `FSCTL_QUERY_USN_JOURNAL` を使用して取得できます。
- USN FileUSN
 - `Fileusn` 値には、ファイルに対して行われた最後の変更の USN ID が含まれており、指定されたファイルのマスターファイルテーブル (MFT) レコード内で追跡されます。
 - USN ジャーナルが削除されると、`Fileusn` は 0 にリセットされます。

この情報と、特定の使用量が必要になる場合があります。その後、ファイルにカーネル EA として設定します。

カーネル拡張属性の設定

カーネル EA を設定するには、まずプレフィックス `"$Kernel."` で開始し、有効な EA 名の文字列で trailed する必要があります。ユーザーモードからカーネル EA を設定しようとすると、警告なしで無視されます。要求は `STATUS_SUCCESS` を返しますが、EA の実際の変更は行われません。カーネルモードから `Zwseteafile` いる API を呼び出すカーネル EA を設定するには、カーネルモードでは十分ではありません。これは、SMB がネットワーク経由での EA の設定をサポートし、これらの要求がサーバーのカーネルモードから発行されるためです。

カーネル EA を設定するには、呼び出し元は、IRP (i/o 要求パケット) の `MinorFunction` フィールドの `IRP_MN_KERNEL_CALL` 値も設定する必要があります。このフィールドを設定する唯一の方法は、カスタムの IRP を生成することであるため、ルーチン `FsRtlSetKernelEaFile` は、カーネル EA を設定するためのサポート関数として `FsRtl` パッケージからエクスポートされています。

`FsRtlSetKernelEaFile` の同じ呼び出しで、normal と kernel EA の設定を混在させることはできません。この操作を実行すると、`STATUS_INTERMIXED_KERNEL_EA_OPERATION` で操作が失敗します。カーネル EA を設定しても、USN ジャーナルに `USN_REASON_EA_CHANGE` レコードは生成されません。その結果、カーネル EA と通常の EA を同じ操作で使用することはできません。

拡張属性のクエリ

ユーザーモードから EA のファイルを照会すると、通常の EA とカーネル EA の両方が返されます。アプリケーションの互換性の問題を最小限に抑えるために、ユーザーモードに戻ります。通常の [Zwquer](#) の操作では、ユーザーモードとカーネルモードの両方から、通常とカーネルの両方の EA が返されます。

使用できるのは、FileObject だけである場合、[FsRtlQueryKernelEaFile](#) を使用すると、カーネル EA のカーネルモードのクエリを実行する方が便利な場合があります。

更新シーケンス番号のジャーナル情報を照会しています

[FSCTL_QUERY_USN_JOURNAL](#) 操作では、IRP の MinorFunction フィールドに `IRP_MN_KERNEL_CALL` 値が設定されていない限り、カーネルモードから発行された場合でも `SE_MANAGE_VOLUME_PRIVILEGE` が必要です。ルーチン `Fsrtlkernel Fscontrolfile` は、カーネルモードコンポーネントがこの USN 要求を簡単に発行できるように、カーネルの `FsRtl` パッケージからエクスポートされています。

メモ Windows 10 以降のバージョン 1703 以降では、この操作を `SE_MANAGE_VOLUME_PRIVILEGE` する必要がなくなりました。

カーネル拡張属性の自動削除

ファイルの USN ID が原因でファイルを再スキャンするだけで、問題が発生しているため、USN 更新プログラムがファイルに投稿される可能性が非常に高くなります。これを簡単にするために、カーネル EA の機能の自動削除が NTFS に追加されました。

このシナリオでは、すべてのカーネル EA が削除されるとは限りません。そのため、拡張 EA プレフィックス名が使用されます。カーネル EA が以下で始まる場合: `"$Kernel.Purge."` 次の USN の理由のいずれかが USN ジャーナルに書き込まれると、NTFS は、そのファイルに存在する、指定された名前付け構文に準拠しているすべてのカーネル EAs を削除します。

- `USN_REASON_DATA_OVERWRITE`
- `USN_REASON_DATA_EXTEND`
- `USN_REASON_DATA_TRUNCATION`
- `USN_REASON_REPARSE_POINT_CHANGE`

このカーネル EA の削除は、メモリ不足の状況でも成功します。

注釈

- カーネル EA は、ユーザーモードのコンポーネントによって改ざんされることはありません。
- カーネル EA は、通常の EA と同じファイルに存在できます。

参照

[Fltquer](#) ごみ箱

[FltSetEaFile](#) セット

[FSCTL_QUERY_USN_JOURNAL](#)

[FsRtlQueryKernelEaFile](#)

[FsRtlSetKernelEaFile](#)

[Zwquer](#) ごみ箱

[ZwSetEaFile](#) セット

ファイル システムのセキュリティに関する問題

2020/04/24 • [Edit Online](#)

前のセクションの説明セキュリティに関する考慮事項一般用語。すべてのドライバーへの関心のある一般的なセキュリティ問題、ファイル システムに関連する特定のセキュリティの問題があります。このセクションでは、これらのファイル システム、ドライバー内での Windows スタイルのセキュリティを実装するためのガイドを提供しようとしています。このセクションでは、この特定の問題と対処するファイル システム内でどのようにについて説明します。このセクションでは、リファレンス ガイドではありません。またこれはすべての可能なセキュリティ脅威の完全な一覧と、軽減する方法。既知の脅威とセキュリティが必要なすべてのファイル システムで問題に関連する重要な問題を識別するためにこのセクションの目的は、代わりに、します。セキュリティを自身の領域は十分に広範なため、このドキュメントは、すべての可能なセキュリティの問題または実装含まれません。

ここでは、次のトピックについて説明します。

[セマンティック モデルのチェック](#)

[セキュリティ チェック](#)

セマンティック モデルの確認

2020/04/24 • [Edit Online](#)

ファイル システムとセキュリティについても説明して、共有ファイルへのアクセスまたは特殊なファイル属性、およびこれらのチェックをファイル システム、セキュリティの一部であるなど、セマンティック モデルの一部であるファイル システムは、小切手の識別します。ファイルの情報です。このセクションでは、セマンティック モデルに準拠するために必要なチェックについて説明します。後のセクションでは、ファイル システムのセキュリティ情報ポリシーに固有のセキュリティ チェックを実行するファイル システムに必要な具体的な手順について説明します。

ここでは、次のトピックについて説明します。

[処理を作成します。](#)

[閉じるときに削除します。](#)

[実行可能イメージ](#)

[名前の変更とハード リンクの処理](#)

[ファイル情報の処理を設定します。](#)

[どちらの I/O 操作](#)

[ファイル システムのコントロールの処理](#)

[メディアの検証](#)

処理の作成

2020/04/24 • [Edit Online](#)

ファイルシステムの場合、特に興味深いセキュリティ作業のほとんどは、[IRP_MJ_作成](#) 処理中に行われます。この手順では、受信要求を分析し、呼び出し元が操作を実行するための適切な権限を持っているかどうかを判断し、必要に応じて操作を許可または拒否する必要があります。幸いなことに、ファイルシステムの開発者にとっては、ほとんどの決定メカニズムがセキュリティ参照モニター内に実装されています。したがって、ほとんどの場合、ファイルシステムは適切なセキュリティ参照モニタールーチンを呼び出して、アクセスを適切に判断するだけで十分です。ファイルシステムのリスクは、必要に応じてこれらのルーチンを呼び出すことができず、不適切に呼び出し元へのアクセスを許可しない場合に発生します。

FAT ファイルシステムなどの標準ファイルシステムの場合、IRP_MJ_CREATE の一部として実行されるチェックは、主にセマンティクスチェックです。たとえば、FAT ファイルシステムには、ファイルまたはディレクトリの状態に基づいて、IRP_MJ_CREATE 処理が許可されていることを確認するための多くのチェックがあります。FAT ファイルシステムによって実行されるこれらのチェックには、読み取り専用メディアのチェックが含まれます (たとえば、上書きや置き換える、読み取り専用メディアに対する "作成" 操作を実行しようとした場合など)、共有アクセスチェック、oplock チェックなどがあります。この分析の最も難しい部分の1つは、別のレベルリソース (ボリュームレベルなど) の状態が原因で、1つのレベル (ファイルレベルなど) の操作が実際には許可されないことを認識することです。たとえば、別のプロセスによってボリュームが排他的にロックされている場合、ファイルを開くことはできません。一般的なチェックの例を次に示します。

- ファイルレベルがボリュームレベルの状態と互換性があるかどうか。ボリュームレベルのロックは cache-control である必要があります。したがって、1つのプロセスが排他ボリュームレベルのロックを保持している場合、そのプロセス内のスレッドだけがファイルを開くことができます。他のプロセスのスレッドでは、ファイルを開くことができません。
- ファイルレベルがメディアの状態と互換性があるかどうか。"作成" 操作によって、"作成" 操作の一部としてファイルが変更されます。これには、ファイルに対する最後のアクセス時刻の上書き、置き換え、および更新が含まれます。これらの "作成" 操作は読み取り専用メディアでは許可されておらず、最終アクセス時刻は更新されません。
- ボリュームレベルは、ファイルレベルの状態と互換性があるかどうかを確認できます。ボリュームで開かれている既存のファイルがある場合、排他ボリュームを開くことはできません。これは、新しい開発者がボリュームを開こうとして失敗したことを検出するため、一般的な問題です。このエラーが発生すると、FSCTL_マウント解除_ボリュームを使用して、開いているハンドルを無効にし、マウント解除を強制して、新しくマウントされたボリュームに排他的にアクセスできるようになります。

また、ファイル属性には互換性がある必要があります。読み取り専用属性を持つファイルは、書き込みアクセス用に開くことができません。一般的な権限を展開した後で、目的のアクセス権を確認する必要があることに注意してください。たとえば、FASTFAT ファイルシステム内のこのチェックは `FatCheckFileAccess` 関数内にあります (WDK に含まれる FASTFAT サンプルの `Acchksup` ソースファイルを参照してください)。

次のコード例は、FAT セマンティクスに固有のものです。また、Dacl を実装するファイルシステムでは、セキュリティ参照モニタールーチン ([Seaccesscheck](#) など) を使用した追加のセキュリティチェックが実行されます。

```

//
// check for a read-only Diredt
//

if (FlagOn(DiredtAttributes, FAT_DIRENT_ATTR_READ_ONLY)) {

    //
    // Check the desired access for a read-only Diredt
    // Don't allow
    // WRITE, FILE_APPEND_DATA, FILE_ADD_FILE,
    // FILE_ADD_SUBDIRECTORY, and FILE_DELETE_CHILD
    //

    if (FlagOn(*DesiredAccess, ~(DELETE |
                                READ_CONTROL |
                                WRITE_OWNER |
                                WRITE_DAC |
                                SYNCHRONIZE |
                                ACCESS_SYSTEM_SECURITY |
                                FILE_READ_DATA |
                                FILE_READ_EA |
                                FILE_WRITE_EA |
                                FILE_READ_ATTRIBUTES |
                                FILE_WRITE_ATTRIBUTES |
                                FILE_EXECUTE |
                                FILE_LIST_DIRECTORY |
                                FILE_TRAVERSE))) {

        DebugTrace(0, Dbg, "Cannot open readonly\n", 0);

        try_return( Result = FALSE );
    }
}

```

FASTFAT によって実装されるより微妙なチェックとして、呼び出し元によって要求されるアクセスが、FAT ファイルシステムが認識しているもの (Acchksup の `FatCheckFileAccess` 関数で、WDK に含まれている FASTFAT サンプルから) があることを確認します。

次のコード例は、ファイルシステムのセキュリティに関する重要な概念を示しています。ファイルシステムに渡された内容が予期した範囲外にならないことを確認します。セキュリティの観点から見た控えめで適切なアプローチは、アクセス要求を把握していない場合、その要求を拒否する必要があることです。

```

//
// Check the desired access for the object.
// Reject what we do not understand.
// The model of file systems using ACLs is that
// they do not type the ACL to the object that the
// ACL is on.
// Permissions are not checked for consistency vs.
// the object type - dir/file.
//

if (FlagOn(*DesiredAccess, ~(DELETE |
                                READ_CONTROL |
                                WRITE_OWNER |
                                WRITE_DAC |
                                SYNCHRONIZE |
                                ACCESS_SYSTEM_SECURITY |
                                FILE_WRITE_DATA |
                                FILE_READ_EA |
                                FILE_WRITE_EA |
                                FILE_READ_ATTRIBUTES |
                                FILE_WRITE_ATTRIBUTES |
                                FILE_LIST_DIRECTORY |
                                FILE_TRAVERSE |
                                FILE_DELETE_CHILD |
                                FILE_APPEND_DATA))) {

    DebugTrace(0, Dbg, "Cannot open object\n", 0);

    try_return( Result = FALSE );
}

```

幸いなことに、ファイルシステムでは、最初の作成処理中にセキュリティチェックが行われると、その後のセキュリティチェックが i/o マネージャーによって実行されます。したがって、たとえば、i/o マネージャーを使用すると、ユーザーモードアプリケーションが読み取りアクセス専用で開かれたファイルに対して書き込み操作を実行しないようにすることができます。実際、ファイルシステムでは、ファイルオブジェクトに対して読み取り専用のセマンティクスを強制しようとししないでください。これは、IRP_MJ_書き込みディスパッチルーチンの実行中に、読み取りアクセスのみで開かれた場合でも同様です。これは、メモリマネージャーが特定のファイルオブジェクトを特定のセクションオブジェクトに関連付ける方法によるものです。ファイルが読み取り専用で開かれている場合でも、そのセクションを使用した後続の書き込みは、ファイルオブジェクトに対する書き込み操作_IRP_MJ として送信されます。つまり、ファイルハンドルが [Obreferenceobjectbyhandle](#) によって Nt システムサービスのエントリポイントで対応するファイルオブジェクトに変換されると、アクセスの適用が行われます。

ファイルシステム内には次の2つの場所があり、セマンティックセキュリティチェックは "作成" 処理と同様に行う必要があります。

- 名前変更またはハードリンクの処理中。
- ファイルシステムの制御操作を処理する場合。

名前の変更処理とファイルシステムの制御処理については、以降のセクションで説明します。

これは、"作成" 処理に関連するセマンティック問題の完全な一覧ではないことに注意してください。このセクションの目的は、ファイルシステムの開発者にとって、これらの問題に注意することです。すべてのセマンティック問題は、特定のセマンティクスを満たすように実装されている特定のファイルシステムに対して特定し、実装がさまざまなケースを処理するようにテストする必要があります。

閉じるときに削除

2020/04/24 • [Edit Online](#)

呼び出し元を指定すると、`ファイル_削除_ON_閉じる` オプションが、呼び出し元がファイルの削除権限を持っていることを確認するか、子権限を削除するファイル システムのチェックのために必要な親ディレクトリです。いずれかのアクセス許可は、ファイルを削除するのには不十分です。これは、処理するためにファイル システムの重要なケースです。閉じているときに、ファイルを削除する操作のセマンティクスは、I/O マネージャーではなく、ファイル システムには適用されません。

ファイル システムは、ボリュームが書き込み禁止ではありませんし、この操作は、この操作が許可されていないディレクトリに適用されないことを確認する必要もあります。FASTFAT ファイル システム コードの書き込みで保護されたボリュームのチェックはやファイルを、使用して削除するルート ディレクトリを許可しないなどの `_削除_ON_閉じる`。これらのチェックの例が記載されて、`FatCommonCreate` WDK を含む `fastfat` サンプルから `Create.c` ソース ファイル内の関数。

実行可能イメージ

2020/04/24 • [Edit Online](#)

メモリの割り当て済みのイメージファイルを使用してプロセスのアドレス空間には、実行可能ファイルが読み込まれます。ファイル自体を開く必要も、ハンドルはセクションを使用して、マッピングが行われるため、作成する必要があります。メモリマップ ファイルをサポートするいと仮定すると、これらの特殊なセマンティクスを適用するファイル システムを確認する必要があります。たとえば、このケースを確認するには、FASTFAT ファイル システム コードで見つかる、FatOpenExistingFCB WDK を含む fastfat サンプルから Create.c ソース ファイル内の関数。

```
//
// If the user wants write access to the file, make sure there
// is not a process mapping this file as an image. Any attempt to
// delete the file will be stopped in fileinfo.c
//
// If the user wants to delete on close, check at this
// point though.
//

if (FlagOn(*DesiredAccess, FILE_WRITE_DATA) || DeleteOnClose) {

    Fcb->OpenCount += 1;
    DecrementFcbOpenCount = TRUE;

    if (!MmFlushImageSection( &Fcb->NonPaged->SectionObjectPointers,
                             MmFlushForWrite )) {

        Iosb.Status = DeleteOnClose ? STATUS_CANNOT_DELETE : STATUS_SHARING_VIOLATION;
        try_return( Iosb );
    }
}
```

そのため、ファイル システムにより、ファイルが開いていない場合でも、実行可能イメージを含む、メモリ マップ ファイルを削除することはできません。

名前変更とハードリンク処理

2020/04/24 • [Edit Online](#)

ファイルシステムの特定の問題の領域は、名前変更操作の適切な処理です。問題のような領域は、ハードリンクをサポートするファイルシステムのハードリンクの作成です。名前の変更操作の場合は、ファイルシステム、ファイルシステムで追加のセキュリティチェックを必要とするファイル(名前の変更操作のターゲット)を削除する可能性があります。

構造体のフィールドの1つは、名前の変更操作の制御構造を調べるときに、**ReplaceIfExists**オプション。

```
typedef struct _FILE_RENAME_INFORMATION {
    BOOLEAN ReplaceIfExists;
    HANDLE RootDirectory;
    ULONG FileNameLength;
    WCHAR FileName[1];
} FILE_RENAME_INFORMATION, *PFILE_RENAME_INFORMATION;
```

同様に、ハードリンク操作のコントロール構造体で構造体のフィールドのいずれかは、**ReplaceIfExists**オプション。

```
typedef struct _FILE_LINK_INFORMATION {
    BOOLEAN ReplaceIfExists;
    HANDLE RootDirectory;
    ULONG FileNameLength;
    WCHAR FileName[1];
} FILE_LINK_INFORMATION, *PFILE_LINK_INFORMATION;
```

どちらの場合も、オプションは存在する場合、操作の対象を置き換えるには。FASTFAT ファイルシステムがハードリンクをサポートしていないときに、名前の変更操作をサポートします。これらのセマンティクスと動作 FASTFAT ファイルシステムのソースコード内で認識される、**FatSetRenameInfo**関数(を参照してください、*Fileinfo.c* WDK を含む fastfat サンプルからソースファイル)。

名前の変更操作を処理するためには、次のコード例では、ファイルを削除するためのファイルシステムのチェックを模倣します。堅牢なセキュリティを備えたファイルシステムのモデル(NTFS の例)は、このチェックは、セキュリティ(呼び出し元には、削除に必要な適切なアクセス許可が必要がある)、指定されたファイルを削除する、呼び出し元が許可されていることを確認するチェックも必要です。

```
//
// The name already exists. Check if the user wants
// to overwrite the name and has access to do the overwrite.
// We cannot overwrite a directory.
//

if ((!ReplaceIfExists) ||
    (FlagOn(TargetDirent->Attributes, FAT_DIRENT_ATTR_DIRECTORY)) ||
    (FlagOn(TargetDirent->Attributes, FAT_DIRENT_ATTR_READ_ONLY))) {

    try_return( Status = STATUS_OBJECT_NAME_COLLISION );
}

//
// Check that the file has no open user handles; otherwise,
// access will be denied. To do the check, search
// the list of FCBs opened under the parent Dcb, and make
// sure that none of the matching FCBs have a nonzero unclean count or
// outstanding image sections.
//

for (Links = TargetDcb->Specific.Dcb.ParentDcbQueue.Flink;
```

```

        Links != &TargetDcb->Specific.Dcb.ParentDcbQueue; ) {

    TempFcb = CONTAINING_RECORD( Links, FCB, ParentDcbLinks );

    //
    // Advance now. The image section flush may cause the final
    // close, which will recursively happen underneath of us here.
    // It would be unfortunate if we looked through free memory.
    //

    Links = Links->Flink;

    if ((TempFcb->DirentOffsetWithinDirectory == TargetDirentOffset) &&
        ((TempFcb->UncleanCount != 0) ||
         !MmFlushImageSection( &TempFcb->NonPaged->SectionObjectPointers,
                               MmFlushForDelete))) {

        //
        // If there are batch oplocks on this file, then break the
        // oplocks before failing the rename.
        //

        Status = STATUS_ACCESS_DENIED;

        if ((NodeType(TempFcb) == FAT_NTC_FCB) &&
            FsRtlCurrentBatchOplock( &TempFcb->Specific.Fcb.Oplock )) {

            //
            // Do all of the cleanup now since the IrpContext
            // could go away when this request is posted.
            //

            FatUnpinBcb( IrpContext, TargetDirentBcb );

            Status = FsRtlCheckOplock( &TempFcb->Specific.Fcb.Oplock,
                                       Irp,
                                       IrpContext,
                                       FatOplockComplete,
                                       NULL );

            if (Status != STATUS_PENDING) {

                Status = STATUS_ACCESS_DENIED;
            }
        }

        try_return( NOTHING );
    }
}

//
// OK, this target is finished. Remember the Lfn offset.
//

TargetLfnOffset = TargetDirentOffset -
    FAT_LFN_DIRENTS_NEEDED(&TargetLfn) * sizeof(DIRENT);

DeleteTarget = TRUE;
}

```


ファイル情報設定処理

2020/04/24 • [Edit Online](#)

サポートされている情報クラスのサブセットのいくつか追加のチェックを実行する I/O マネージャー [IRP_MJ_設定_情報](#)します。具体的には、FileRenameInformation、FileLinkInformation および FileMoveClusterInformation、I/O マネージャー問題のオープン送信するには、その親の下に子を作成するアクセスをユーザーが持っていることを確認する対象の名前の親ディレクトリに、IRP_MJ_設定_ファイル システムに情報の要求。

I/O のどちらでもない操作

2020/04/24 • [Edit Online](#)

ファイル システムでは、通常ユーザー バッファを直接操作に関連する操作を処理する必要があります。このような操作は、ユーザーのアドレスが無効になるため、本質的に危険です。ファイル システムは、このような操作を特に意識し、適切に保護することを確認する必要があります。次の操作に依存して、フラグ I/O マネージャーのユーザーとカーネル アドレス スペース間でデータを転送する方法を指定するファイル システムのデバイス オブジェクトのメンバー。

- [IRP_MJ_ディレクトリ_コントロール](#)
- [IRP_MJ_クエリ_EA](#)
- [IRP_MJ_クエリ_クォータ](#)
- [IRP_MJ_READ](#)
- [IRP_MJ_SET_EA](#)
- [IRP_MJ_SET_QUOTA](#)
- [IRP_MJ_WRITE](#)

通常、ファイル システムがどちらの I/O に暗黙的にどちらの操作を設定して_直接_IO も DO_バッファに格納された_で IO、フラグボリューム デバイスのメンバー オブジェクトを作成します。

次の操作は、無視、フラグファイル システムのメンバーのデバイス オブジェクトと、どちらの I/O を使用して、ユーザーとカーネル アドレス スペース間でデータを転送します。

- [IRP_MJ_QUERY_SECURITY](#)

どちらの I/O を使用して、ファイル システムは、独自のデータ転送操作を処理します。これにより、アプリケーションのユーザー スペースのバッファにデータを直接配置することで、操作を満たすためにファイル システムです。ファイル システムは、操作が開始され、操作が進行中に無効になるバッファを適切に処理、ユーザーのバッファが有効であることを確認したがって必要があります。また、高速の I/O には、生のポインターが渡されます。開発者が認識操作の開始位置のバッファの有効性のチェックが不十分であることが、操作全体で有効にすることを確認する必要があります。たとえば、悪意のあるアプリケーションでした (たとえばセクション) から、メモリ ブロックにマップ、発行、I/O 操作および I/O 操作が進行中にメモリ ブロックを解除します。

このような状況を処理するためにファイル システムのいくつかの方法があります。1 つのメカニズムはユーザーのアドレスに対応する物理メモリをロックダウンする、オペレーティング システムのアドレス空間に 2 つ目のマッピングを作成します。これにより、ファイル システムがそれによって制御される仮想アドレスを使用します。その場合でも、ユーザーのアドレスが無効になると、ファイル システムによって作成されたアドレスは有効のようになります。FASTFAT のファイル システムのコードでは、これを実現するために 2 つの異なる関数を使用します。ユーザーのバッファを最初の関数のロック。

```

VOID
FatLockUserBuffer (
    IN PIRP_CONTEXT IrpContext,
    IN OUT PIRP Irp,
    IN LOCK_OPERATION Operation,
    IN ULONG BufferLength
)

/*++
Routine Description:

    This routine locks the specified buffer for the specified type of
    access. The file system requires this routine because it does not
    ask the I/O system to lock its buffers for direct I/O. This routine
    can only be called from the file system driver (FSD) while still in the user context.

    Note that this is the *input/output* buffer.

Arguments:
    Irp - Pointer to the Irp for which the buffer will be locked.
    Operation - IoWriteAccess for read operations, or IoReadAccess for
        write operations.
    BufferLength - Length of user buffer.

Return Value:
    None
--*/

{
    PMDL Mdl = NULL;

    if (Irp->MdlAddress == NULL) {
        //
        // Allocate the Mdl and Raise if the allocation fails.
        //
        Mdl = IoAllocateMdl( Irp->UserBuffer, BufferLength, FALSE, FALSE, Irp );
        if (Mdl == NULL) {
            FatRaiseStatus( IrpContext, STATUS_INSUFFICIENT_RESOURCES );
        }

        //
        // now probe the buffer described by the Irp. If there is an exception,
        // deallocate the Mdl and return the appropriate "expected" status.
        //
        try {
            MmProbeAndLockPages( Mdl,
                                Irp->RequestorMode,
                                Operation );
        } except(EXCEPTION_EXECUTE_HANDLER) {
            NTSTATUS Status;
            Status = GetExceptionCode();
            IoFreeMdl( Mdl );
            Irp->MdlAddress = NULL;
            FatRaiseStatus( IrpContext,
                            FsRtlIsNtstatusExpected(Status) ? Status : STATUS_INVALID_USER_BUFFER );
        }
    }

    UNREFERENCED_PARAMETER( IrpContext );
}

```

このルーチンにより、ユーザーのアドレスをサポートする物理メモリは再利用できません、他の目的、操作が進行中によりになります。ファイル システムは、基になるボリューム管理またはディスク クラスのレイヤーにキャッシュされていないユーザー I/O を満たすために、I/O 操作を送信するためにこれを実行します。このような場合は、ファイル システムにバッファに独自の仮想アドレスは必要ありません。2 番目の関数は、カーネルのアドレス空間に、ファイル システムのマッピングを作成します。

```

PVOID
FatMapUserBuffer (
    IN PIRP_CONTEXT IrpContext,
    IN OUT PIRP Irp
)
/*++
Routine Description:
    This routine conditionally maps the user buffer for the current I/O
    request in the specified mode. If the buffer is already mapped, it
    just returns its address.

    Note that this is the *input/output* buffer.

Arguments:
    Irp - Pointer to the Irp for the request.

Return Value:
    Mapped address
--*/
{
    UNREFERENCED_PARAMETER( IrpContext );

    //
    // If there is no Mdl, then we must be in the FSD, and can simply
    // return the UserBuffer field from the Irp.
    //
    if (Irp->MdlAddress == NULL) {
        return Irp->UserBuffer;
    } else {
        PVOID Address = MmGetSystemAddressForMdlSafe( Irp->MdlAddress, NormalPagePriority );
        if (Address == NULL) {
            ExRaiseStatus( STATUS_INSUFFICIENT_RESOURCES );
        }
        return Address;
    }
}

```

FASTFAT 実装では、2 つ目のルーチンもユーザー レベルのアドレスを許可します。これにはFAT ファイル システムの返されるアドレス (ユーザーまたはカーネル) が有効でなければならないことを確認する必要があります。同様に、これを行うには __try と__except キーワードを使用して保護されたブロックを作成します。

これらのルーチンは、WDK を含む fastfat サンプルから deviosup.c ソース ファイルには。

もう 1 つの重要な問題では、呼び出し元のコンテキストでは、要求が満たされていない場合に発生します。ファイル システムでは、要求、ワーカー スレッドを投稿記事、した場合、ドライバーをことのない見失う MDL バッファーをロックする必要があります。

FatPrePostIrp fastfat サンプルから workque.c ソース ファイル内の関数は、FASTFAT ファイル システムでこの問題を処理する方法の例を示します。

FASTFAT ファイル システムは、これらのルーチンを使用して、まで、広範囲のない単に無効なユーザーのバッファーの障害から保護します。これは、非常に強力な手法でも、すべての保護されたコード ブロックが正しくが保持しているすべてのリソースを解放することを確認が必要です。リソース解放するにはには、メモリ、同期オブジェクト、またはファイル システム自体の他のリソースが含まれます。そのためには、障害は、リソースを使い果たしてにオペレーティング システムに多くを繰り返し呼び出すことで、リソースの枯渇を引き起こすことが可能に志望攻撃者になります。

ファイル システム制御処理

2020/04/24 • [Edit Online](#)

IRP_MJ_ファイル_システム_制御 操作は、ファイルシステム内の他の操作に必要なデータバッファ処理とは異なります。これは、各操作によって、CTL_コードマクロを使用して制御コードの一部として、i/o マネージャーに固有のデータ転送機構が確立されるためです。また、コントロールコードでは、呼び出し元に必要なファイルアクセスを指定します。このアクセスは i/o マネージャーによって適用されるため、コントロールコードを定義するときは、ファイルシステムが特にこの問題を cognizant する必要があります。一部の i/o 制御コード (FSCTL_MOVE_ファイルなど) では、ファイル_特別な_アクセスを指定します。これは、ファイルシステムによって操作のセキュリティがファイルシステムによって直接チェックされることをファイルシステムが示すことを許可するメカニズムです。ファイル_特別な_アクセスは、ファイル_すべての_アクセスに相当します。そのため、i/o マネージャーでは特定のセキュリティチェックが行われず、ファイルシステムではなく遅延されます。ファイル_特別な_アクセスは、主に、ファイルシステムによって追加のチェックが行われるドキュメントを提供します。

いくつかのファイルシステム操作では、ファイル_特別な_アクセスが指定されています。FSCTL_MOVE_FILE 操作は、ファイルシステムの最適化インターフェイスの一部として使用され、ファイル_特別な_アクセスを指定します。読み取りと書き込みが行われている開いているファイルを最適化できるようにするため、使用するハンドルには、共有アクセスの競合を防ぐためにアクセスを許可されたファイル_読み取り_属性のみがあります。ただし、ディスクが低いレベルで変更されているため、この操作は特権操作である必要があります。この問題を解決するには、FSCTL_MOVE_ファイルの発行に使用されるハンドルが、特権ハンドルであるダイレクトアクセスストレージデバイス (DASD) ユーザーボリュームを開いていることを確認します。開いているユーザーボリュームに対してこの操作が実行されていることを確認する FASTFAT ファイルシステムコードは、FatMoveFile関数にあります (WDK に含まれる FASTFAT サンプルの fsctrl ソースファイルを参照してください)。

```
//
// extract and decode the file object and check for type of open
//

if (FatDecodeFileObject( IrpSp->FileObject, &Vcb, &FcbOrDcb, &Ccb ) != UserVolumeOpen) {

    FatCompleteRequest( IrpContext, Irp, STATUS_INVALID_PARAMETER );

    DebugTrace(-1, Dbg, "FatMoveFile -> %08lx\n", STATUS_INVALID_PARAMETER);
    return STATUS_INVALID_PARAMETER;
}
```

FSCTL_MOVE_FILE 操作で使用される構造体は、移動するファイルを指定します。

```
typedef struct {
    HANDLE FileHandle;
    LARGE_INTEGER StartingVcn;
    LARGE_INTEGER StartingLcn;
    ULONG ClusterCount;
} MOVE_FILE_DATA, *PMOVE_FILE_DATA;
```

前述のように、FSCTL_MOVE_ファイルを発行するために使用されるハンドルはボリューム全体の "オープン" 操作であり、実際には、移動_ファイル_データ入力バッファに指定されたファイルハンドルに適用されます。これにより、この操作のセキュリティチェックがやや複雑になります。たとえば、このインターフェイスでは、ファイルハンドルを移動するファイルを表すファイルオブジェクトに変換する必要があります。これには、ドライバーの一部を慎重に検討する必要があります。FASTFAT は、WDK に含まれる FASTFAT サンプルの fsctrl ソースファイルのFatMoveFile関数で、保護された方法で**obreferenceobject**を使用します。

```
//
// Try to get a pointer to the file object from the handle passed in.
//

Status = ObReferenceObjectByHandle( InputBuffer->FileHandle,
                                    0,
                                    *IoFileObjectType,
                                    Irp->RequestorMode,
                                    &FileObject,
                                    NULL );

if (!NT_SUCCESS(Status)) {

    FatCompleteRequest( IrpContext, Irp, Status );

    DebugTrace(-1, Dbg, "FatMoveFile -> %08lx\n", Status);
    return Status;
}
// Complete the following steps to ensure that this is not an invalid attempt
//
// - check that the file object is opened on the same volume as the
//   DASD handle used to call this routine.
//
// - extract and decode the file object and check for type of open.
//
// - if this is a directory, verify that it's not the root and that
//   you are not trying to move the first cluster. You cannot move the
//   first cluster because sub-directories have this cluster number
//   in them and there is no safe way to simultaneously update them
//   all.
//
// Allow movefile on the root directory if it's FAT32, since the root dir
// is a real chained file.
//  //
```

Irp->Irp->requestormode を使用して、呼び出し元がユーザーモードアプリケーションである場合、ハンドルをカーネルハンドルにすることはできません。ファイルがアクティブにアクセスされている間に移動できるように、必要なアクセス権は0です。最後に、呼び出しがユーザーモードで開始された場合、この呼び出しは正しいプロセスコンテキストで行う必要があることに注意してください。FASTFAT ファイルシステムのソースコードでは、fsctrl の**FatMoveFile**関数にもこれが適用されます。

```
//
// Force WAIT to true. There is a handle in the input buffer that can only
// be referenced within the originating process.
//

SetFlag( IrpContext->Flags, IRP_CONTEXT_FLAG_WAIT );
```

FAT ファイルシステムによって実行されるこれらのセマンティックセキュリティチェックは、ハンドルを渡すすべての操作について、ファイルシステムによって必要とされる一般的なものです。また、FAT ファイルシステムも、操作に固有の正常性チェックを実行する必要があります。これらの正常性チェックでは、呼び出し元が許可されていない場合に特権操作を実行しないようにするために、異なるパラメーターに互換性があることを確認します (移動するファイルは、開かれたボリューム上にあります)。

ファイルシステムによっては、次のようなファイルシステム制御操作において適切なセキュリティが不可欠です。

- ユーザーハンドルを適切に検証します。
- ユーザーバッファアクセスの保護。
- 特定の操作のセマンティクスを検証しています。

多くの場合、適切な検証とセキュリティを実行するために必要なコードは、指定された関数内のコードの大部分を構成することができます。

メディアの検証

2020/04/24 • [Edit Online](#)

リムーバブルメディア (たとえば FASTFAT) をサポートするファイル システムを開発するときに、大きな問題は、「死のディスク」攻撃から保護するため。リムーバブルディスクを挿入できるすべてのユーザーからファイル システムを実装する場合、ドライバーが悪意のある形式が正しくない構造体に対して防ぐ必要があります (CD-ROM、DVD-ROM、または USB メモリのディスクをフラッシュなど)、システムにします。

ファイルシステムでのセキュリティチェック

2020/08/04 • [Edit Online](#)

ファイルシステムのセキュリティに関する責任は、セキュリティチェックの領域にあります。これらは、実際にオブジェクトを "所有" する Windows の一部であるため、ファイルシステム内に実装されます。セキュリティを実装する目的は、オブジェクトを保護するために (ファイルシステムによって実装される) ポリシーと、アクセスを決定するための機構 (セキュリティ参照モニターによって実装される) を分離することです。

つまり、ファイルシステムの開発者は、ファイルシステムリソースへの正しいアクセスを検証するために、適切なタイミングでセキュリティ参照モニターを呼び出す必要があります。ファイルシステムでは、セキュリティの参照の監視によってセキュリティの決定がどのように行われるかについて理解する必要はありません。ここでは、ファイルシステムがセキュリティチェックの追加を検討する可能性がある点について説明します。

ここでは、次のトピックについて説明します。

[デバイス オブジェクトに対するセキュリティ記述子の摘要](#)

[IRP_MJ_CREATE のセキュリティチェック](#)

[IRP_MJ_QUERY_SECURITY と IRP_MJ_SET_SECURITY のセキュリティチェック](#)

[IRP_MJ_DIRECTORY_CONTROL のセキュリティチェック](#)

[IRP_MJ_FILE_SYSTEM_CONTROL のセキュリティチェック](#)

[IRP_MJ_SET_INFORMATION のセキュリティチェック](#)

[権限借用](#)

[処理とスレッド終了に関する問題](#)

デバイス オブジェクトに対するセキュリティ記述子の摘要

2020/04/24 • [Edit Online](#)

ほとんどのドライバーは、デバイスオブジェクトに対して i/o マネージャーによって適用されたアクセス制御を使用して、不適切なアクセスから保護します。ほとんどのドライバーの最も簡単な方法は、ドライバーのインストール時に明示的なセキュリティ記述子を適用することです。INF ファイルでは、このようなセキュリティ記述子は、AddReg セクションの "Security" エントリによって記述されます。セキュリティ記述子の記述に使用される完全な言語の詳細については、Microsoft Windows SDK のドキュメントの「セキュリティ記述子の定義言語」を参照してください。

セキュリティ記述子定義言語 (SDDL) を使用したセキュリティ記述子の基本形式には、標準セキュリティ記述子の次のような部分が含まれます。

- 所有者 SID。
- グループ SID。
- 随意アクセス制御リスト (DACL)。
- システムアクセス制御リスト (SACL)。

そのため、セキュリティ記述子の INF スクリプト内の説明は次のようになります。

```
O:owner-sidG:group-sidD:dac1-flags(ace)(ace)S:sac1-flags(ace)(ace)
```

個々のアクセス制御エントリは、セキュリティ識別子または SID によって指定された特定のグループまたはユーザーに対して許可または拒否されるアクセスを記述します。たとえば、INF ファイルには次のような行が含まれている場合があります。

```
"D:P(A;CI;GR;;;BU)(A;CI;GR;;;PU)(A;CI;GA;;;BA)(A;CI;GA;;;SY)(A;CI;GA;;;NS)(A;CI;GA;;;LS)
(A;CI;CCDCLCSWRPSDRC;;;S-1-5-32-556)"
```

上記の例は、NETTCPIP からのものです。Microsoft Windows XP Service Pack 1 (SP1) システムの INF ファイル。

このインスタンスでは、所有者またはグループが指定されていないため、既定値または既定値が既定値になります。Dは、これが DACL であることを示します。Pは、これが保護された ACL であり、親オブジェクトのセキュリティ記述子からの権限を継承していないことを示します。保護された ACL により、親の継承が継承されるのを防ぐことができます。かっこで囲まれた式は、単一のアクセス制御エントリ (ACE) を示します。SDDL を使用するアクセス制御エントリは、セミコロンで区切られた複数のコンポーネントで構成されています。順序は次のとおりです。

- ACE の型インジケーター。Dac1 には4つの一意の型があり、Sacl には4種類あります。
- 子オブジェクトに対するこの ACE の継承、または Sacl の監査とアラームポリシーを説明するために使用されるフラグフィールド。
- 権限フィールドは、ACE によって許可または拒否される権限を示します。このフィールドには、この ACE に適用される汎用、標準、および特定の権限を示す特定の数値を指定することも、一般的なアクセス権の文字列の説明を使用することもできます。
- DACL がオブジェクト固有の ACE 構造体である場合は、オブジェクト GUID。
- DACL がオブジェクト固有の ACE 構造体である場合は、継承されたオブジェクト GUID
- この ACE が適用されるセキュリティエンティティを示す SID。

このため、サンプルのセキュリティ記述子を解釈すると、ACE の先頭の "A" は、これが "アクセス許可" エントリであることを示します。別の方法としては、"アクセスが拒否されました" エントリがあります。このエントリは、使用頻度が低いため、先頭の "D" 文字で示されます。Flags フィールドは、この ACE がサブオブジェクトによって継承されることを示す Container INHERIT (CI) を指定します。

権限フィールド値は、一般的な権限や標準権限を含む特定の権限をエンコードします。たとえば、"GR" は "generic read" アクセスを示し、"GA" は "汎用的な権限" アクセスを示します。多くの特別な権限は、これらの汎用的な権限に従います。上記のサンプルでは、"CC" は、ファイルとディレクトリの権限に固有の [子アクセスの作成] を示しています。上記のサンプルには、子アクセスの削除のための "DC" を含む "CC" 文字列の後にあるその他の標準の権限、リストの子アクセス用の "LC"、自己右アクセス用の "RP"、読み取りプロパティアクセス用の "SD"、読み取り制御アクセス用の "RC" が含まれています。

上記のサンプルの SID エントリ文字列には、パワーユーザー用の "PU"、組み込みユーザーの場合は "BU"、ローカルサービスアカウントの場合は "BA"、ローカルシステムの場合は "SY"、network service アカウントの場合は "NS" が含まれます。このため、上の例では、ユーザーにはオブジェクトに対する汎用読み取りアクセス権が与えられています。これに対して、組み込みの管理者、ローカルサービスアカウント、ローカルシステム、およびネットワークサービスには、汎用のすべてのアクセス (読み取り、書き込み、および実行) が与えられます。使用可能なすべての権限と標準 SID 文字列の完全なセットについては、Windows SDK をご覧ください。

これらの Acl は、特定のドライバーによって作成されるすべてのデバイスオブジェクトに適用されます。また、ドライバーは、名前付きデバイスオブジェクトを作成するときに、[locreateデバイス](#)の新しい関数を使用して、特定のオブジェクトのセキュリティ設定を制御することもできます。locreateデバイスの機能は、Windows XP Service Pack 1 および windows Server 2003 以降で使用できます。locreateデバイスを使用すると、デバイスオブジェクトに適用されるセキュリティ記述子は、デバイスオブジェクトに適した完全なセキュリティ記述子定義言語のサブセットを使用して記述されます。

デバイスオブジェクトに特定のセキュリティ記述子を適用する目的は、アプリケーションがデバイス自体にアクセスしようとするたびに、適切なセキュリティチェックがデバイスに対して実行されるようにすることです。名前構造 (ファイルシステムの名前空間など) を含むデバイスオブジェクトの場合、このデバイス名前空間へのアクセスの管理の詳細は、i/o マネージャーではなく、ドライバーに属しています。

このような場合の興味深い問題は、ドライバーデバイスオブジェクトとデバイスドライバーへのアクセスを確認する i/o マネージャー間の境界でセキュリティを処理する方法です。この場合、ドライバーに適したセキュリティポリシーがすべて実装されます。従来、開いているオブジェクトがデバイス自体の名前である場合、i/o マネージャーは、そのセキュリティ記述子を使用して、デバイスオブジェクトに対して完全なアクセスチェックを直接実行します。ただし、開いているオブジェクトがドライバー自体の内部のパスを示している場合、i/o マネージャーは、デバイスオブジェクトに対して走査アクセスが許可されているかどうかのみを確認します。通常、この走査権限は、ほとんどのスレッドに `SeChangeNotifyPrivilege` が付与されているために付与されます。これは、ディレクトリへの走査権限の付与に対応します。通常、名前の構造をサポートしていないデバイスは、i/o マネージャーが完全なセキュリティチェックを実行するように要求します。これを行うには、[デバイスの特性] フィールドで [ファイル_デバイス_安全_オープンビット] を設定します。このようなデバイスオブジェクトが混在するドライバーでは、名前の構造をサポートしていないデバイスに対して、この特性を設定する必要があります。たとえば、ファイルシステムは、名前付きデバイスオブジェクト (名前付け構造体をサポートしていません) でこのオプションを設定しますが、名前のないデバイスオブジェクト (たとえば、ボリューム) では名前付け構造をサポートしているので、このオプションを設定しません。このビットを正しく設定できないと、ドライバーに共通のバグが発生し、デバイスへの不適切なアクセスが許可される可能性があります。添付ファイルインターフェイスを使用するドライバー ([loattachdevicetodevicestacksafe](#) など) の場合、ドライバーをアタッチするデバイスでこのフィールドを設定すると、`ファイル_デバイス_セキュリティで保護された_オープンビット` が設定されます。そのため、フィルタードライバーでは、セキュリティチェックの特定の側面について心配する必要はありません。

IRP_MJ_ディスパッチ ルーチンの作成

2020/04/24 • [Edit Online](#)

内の Windows セキュリティ チェックの主要な部分で発生した、**IRP_MJ_作成** ルーチンをディスパッチします。Windows のセキュリティ モデルの大部分が検証へのアクセスに関連するためです。アクセスの検証結果は、この操作の結果として作成されたハンドルの一部として格納されます。後続の操作は、この時点で計算権限に対して検証されます。

元がアクセス権を IRP の中に指定である場合は、アクセスは、ファイルまたはディレクトリを開いた後、ファイルの変更に対する権限、_MJ_を有効にする作成操作を続行します。これらのアクセス権は、ハンドルに関連付けられた、後続の操作を管理、下で、アクセスが与えられたハンドルが解決しない限りためです。

ここでは、次のトピックについて説明します。

[IRP の走査の権限のチェック_MJ_を作成します。](#)

[IRP の他の特殊なケースについてチェック_MJ_を作成します。](#)

[IRP の監査を追加する_MJ_を作成します。](#)

[IRP 上へのアクセス制御の管理を一覧表示_MJ_を作成します。](#)

[IRP 上の新しいファイルへのセキュリティの割り当て_MJ_を作成します。](#)

[IRP のクォータを処理_MJ_を作成します。](#)

IRP の走査の権限のチェック_MJ_を作成します。

2020/04/24 • [Edit Online](#)

懸念事項の 1 つ **IRP_MJ_作成** チェックは、呼び出し元が走査特権を持っているかどうか (には、呼び出し元がオブジェクトへのパスにアクセスする権限)。ほとんどの呼び出し元に走査特権があるために走査特権ファイル システム内で通常実行される最初のチェックのいずれかの確認しています。

```
BOOLEAN traverseCheck =  
    !(IrpContext->IrpSp->Parameters.Create.SecurityContext->AccessState->Flags  
      & TOKEN_HAS_TRAVERSE_PRIVILEGE);
```

走査の権限の確認が I/O マネージャーから、ファイル システムに渡される状態情報に依存して ことに注意してください。この情報は、呼び出し元が SeChangeNotifyPrivilege を保持するかどうかに基づいています。呼び出し元がこの特権を保持しない場合は、各ディレクトリの走査チェックを実行してください。走査チェック次の例で汎用ルーチンを使用して行う、通常ほとんどのセキュリティ チェックの使用します。

```
{
```

```

SeLockSubjectContext(
    &accessParams.AccessState->SubjectSecurityContext);
//
// Note: AccessParams is passed to us and is normally based on
//       the fields of the same name from the IRP
//

granted = SeAccessCheck( Fcb->SecurityDescriptor,
    &AccessParams.AccessState->SubjectSecurityContext,
    TRUE,
    AccessParams.desiredAccess,
    0,
    &Privileges,
    IoGetFileObjectGenericMapping(),
    AccessParams.AccessMode,
    &AccessParams.GrantedAccess,
    &AccessParams.status );

if (Privileges != NULL) {
    //
    // this changes the AccessState
    //
    (void) SeAppendPrivileges(AccessParams.AccessState, Privileges );
    SeFreePrivileges( Privileges );
    Privileges = NULL;
}

if (granted) {
    //
    // delete granted bits from desired bits
    //
    AccessParams.desiredAccess &=
        ~(AccessParams.GrantedAccess | MAXIMUM_ALLOWED);

    if (!checkOnly) {
        //
        // the caller wants to modify the access state for this
        // request
        //
        AccessParams.AccessState->PreviouslyGrantedAccess |=
            AccessParams.GrantedAccess;

        if (maxDesired) {
            maxDelete =
                (BOOLEAN)(AccessParams.AccessState->PreviouslyGrantedAccess &
                    DELETE);
            maxReadAttr =
                (BOOLEAN)(AccessParams.AccessState->PreviouslyGrantedAccess &
                    FILE_READ_ATTRIBUTES);
        }
        AccessParams.AccessState->RemainingDesiredAccess &=
            ~(AccessParams.GrantedAccess | MAXIMUM_ALLOWED);
    }
    SeUnlockSubjectContext(&accessParams.AccessState->SubjectSecurityContext);
}

```

この関数は、汎用的なセキュリティ チェックを実行します。この関数は、その際に、次の問題を扱う必要があります。

- チェックに使用する適切なセキュリティ記述子を指定する必要がある必要があります。
- (これらは、操作を実行するエンティティの資格情報) のセキュリティ コンテキストに渡す必要があります。
- セキュリティ チェックの結果に基づくアクセスの状態更新する必要があります。
- 最大値を考慮する必要があります_許可されているオプション (詳細については、WDK を含むファイルを含める ntifs.h を参

照してください)。最大_許可オプションは、ファイル システムがファイル システムで許可されている最大の可能なアクセスをアクセスを設定する必要がありますを指定します (読み取り/書き込み/削除など)。ほとんどのアプリケーションでは、最大使用_許可オプション FASTFAT ファイル システムでは、このオプションはサポートされていないためです。最大_許可されているオプションのビットがない、FASTFAT ファイル システムが認識するアクセス ビットのいずれか、指定されたファイルへのアクセス要求を拒否します。最大値を使用して FASTFAT ボリューム上のファイルを開こうとするアプリケーション_許可されているオプション セットは、要求が失敗したことを検索します。詳細については、次を参照してください。、FatCheckFileAccess 、WDK を含む FASTFAT サンプル コードの Acchksup.c ソース ファイル内の関数。

単純な走査チェックの要求のアクセス権があるファイルに注意してください_走査とセキュリティ記述子であるディレクトリのを通過する元の IRPからのアクセスを要求しないため、呼び出し元が試みているを通じて_。MJ_IRP を作成します。

IRP の他の特殊なケースについてチェック_MJ_を作成します。

2020/04/24 • [Edit Online](#)

その他の特殊なケースの処理は IRP の中に発生する必要があります_MJ_呼び出し元が SeChangeNotifyPrivilege を持っていない場合は、ファイル システム内で処理を作成します。たとえば、そのファイルの ID またはオブジェクト ID を使用して開くことができるファイルでは、呼び出し元が走査ディレクトリ ツリーの構造を使用して、オブジェクトを取得するためのアクセス許可がないので、そのファイルへのパスを取得する呼び出し元が許可されて可能性があります。

この場合は、ファイル システムを検討することがあります。

- **ファイル_オープン_BY_ファイル_ID** -ファイルの名前がないように、呼び出し元にします。IRP の中に、ファイル システムで走査のアクセス許可についてを計算する必要がありますので、この情報 (アクセス許可の走査) では、作成操作中にのみ認識、ファイル情報のクエリの呼び出し中に利用できませんが、_MJ_作成します。
- **SL_オープン_ターゲット_ディレクトリ** : ターゲット ファイルが存在しない、ファイルの作成のアクセスには、ディレクトリを確認する必要があります。ターゲットが存在する場合は、削除のアクセス確認が必要です。IRP の中に名前変更操作時に削除のアクセス チェックを実行する通常_MJ_設定_情報の処理。
- **ファイル_優先的とファイル_上書き** -場合でも、呼び出し元が明示的に要求に破壊的な操作は、追加のアクセス権が必要です。たとえば、ファイル システムでは、優先的操作を実行するために削除のアクセスを必要があります。
- **ファイル_作成、ファイル_オープン_場合、およびファイル_上書き_場合** 建設的な操作には、新しいオブジェクトが作成される親ディレクトリへのアクセスが必要です。これは、格納されているディレクトリで、オブジェクト自体、およびそのため、以前に同様のコード サンプルではなく確認できます。
- **SL_FORCE_アクセス_確認** -カーネル モードではなく、ユーザー モードから呼び出す場合と、チェックを実行する必要がありますこの値が、I/O スタックの場所に設定されている場合。そのため、セキュリティの監視ルーチンの呼び出しを指定する必要がありますUserModeカーネル モードのサーバー内で、呼び出しが発生した場合でもです。
- **ファイル/ディレクトリの削除** -ファイルの ACL の状態に基づいてこの可能性があります (たとえば、ファイル_書き込み_データ アクセスが暗黙的に削除を許可; 有効な削除がある場合は、データを削除します。ファイルのアクセス許可)、ディレクトリの ACL の状態と同様 (ファイル_削除_例については、格納されているディレクトリのアクセス許可を子)。

次のコード例は、特殊なケースのファイルの処理を示します_優先的とファイル_上書き、どちらの場合も、追加のアクセスが暗黙的に必要な呼び出し元によって要求されていなかった場合でもです。


```

{
    ULONG NewAccess = Supersede ? DELETE : FILE_WRITE_DATA;
    ACCESS_MASK AddedAccess = 0;
    PACCESS_MASK DesiredAccess =
        &IrpSp->Paramters.Create.SecurityContext->DesiredAccess;

    //
    // If the caller does not have restore privilege, they must have write
    // access to the EA and attributes for overwrite or supersede.
    //
    if (0 == (AccessState->Flags & TOKEN_HAS_RESTORE_PRIVILEGE)) {
        *DesiredAccess |= FILE_WRITE_EA | FILE_WRITE_ATTRIBUTES;

        //
        // Does the caller already have this access?
        //
        if (AccessState->PreviouslyGrantedAccess & NewAccess) {

            //
            // No - they need this as well
            //
            *DesiredAccess |= NewAccess;

        }

        //
        // Now check access using SeAccessCheck (omitted)
        //
    }
}

```

このコード サンプルは、適切なファイル システムのポリシーが優先の例です。呼び出し元が、削除のアクセスまたはファイルに要求しなかった_書き込み_データ アクセスがこのようなアクセスは、ファイル システムのセマンティクスをに基づいて実行される操作に固有です。

IRP の監査を追加する _MJ_ を作成します。

2020/04/24 • [Edit Online](#)

ファイル システム内でセキュリティ チェックのもう 1 つの重要な側面では、必要に応じて、監査を追加します。通常、これは監査の目的は、システムによるセキュリティ上の決定を記録するために、セキュリティ上の決定を行うルーチンの同じセットの一部として行います。たとえば、次のコードは、アクセス チェックの完了後にファイル システム内で監査を実装するために管理される可能性があります。

```
{
UNICODE_STRING FileAuditObjectName;

RtlInitUnicodeString(&FileAuditObjectName, L"File");

if ( SeAuditingFileOrGlobalEvents (AccessGranted,
    &Fcb->SecurityDescriptor,
    &AccessState->SubjectSecurityContext)) {
    //
    // Must pass complete Windows path name, including device name.
    //
    ConstructAuditFileName(Irp, Fcb, &AuditName);

    if (IrpSp->Parameters.Create.SecurityContext->FullCreateOptions
        & FILE_DELETE_ON_CLOSE) {
        SeOpenObjectForDeleteAuditAlarm(&FileAuditObjectName,
            NULL,
            &AuditName,
            &Fcb->SecurityDescriptor,
            AccessState,
            FALSE, // Object not created.
            // Was it successful?
            // Based on SeAccessCheck
            SeAccessCheckAccessGranted,
            // UserMode or KernelMode
            EffectiveMode,
            &AccessState->GenerateOnClose
        );
    } else {
        SeOpenObjectAuditAlarm(&FileAuditObjectName,
            NULL,
            &AuditName,
            &Fcb->SecurityDescriptor,
            AccessState,
            FALSE, // object not created
            // Was it successful?
            // Based on SeAccessCheck
            AccessGranted,
            // UserMode or KernelMode
            EffectiveMode,
            &AccessState->GenerateOnClose
        );
    }

    //
    // Free file name here if needed.
    //
}
```

IRP 上へのアクセス制御の管理を一覧表示_MJ_を作成します。

2020/04/24 • [Edit Online](#)

これは、さまざまなセキュリティに関連するその他の問題、ファイル システム内でアドレスを指定できます。たとえば、ディスク上のアクセス制御リストの管理は、主要なセキュリティの問題です。セキュリティ情報は、何千ものファイルのと同じである可能性があります、こと、セキュリティ記述子の共有のモデルを実装するために、ファイル システムと便利です。したがって、同じセキュリティ記述子を使用するすべてのファイルは、セキュリティ記述子の 1 つのディスク上 (および場合によってメモリ内) のコピーを共有します。NTFS ファイル システムでは、このモデルを使用します。

1 つのオプションは、ファイル システム キャッシュ結果になります。セキュリティに関連が厳密は、セキュリティ操作がかなりのコストをファイルを開くなどの通常の操作に追加できることを認識する必要があります。したがって、以前の操作からのセキュリティの結果をキャッシュは、以前の決定に依存するファイル システムを許可できます。たとえば、同じファイルに同じユーザーに与えられていたアクセスのサブセットを要求する新しい呼び出しを閉じません付与する可能性があります。もちろん、このような任意のメカニズムを追加するリスクとは、不適切なアクセスを許可するバグを追加する可能性があります。任意のセキュリティの実装が期待どおりの方法で動作することを確認するを徹底的にテストすることを確認するには重要です。

IRP 上の新しいファイルへのセキュリティの割り当て_MJ_を作成します。

2020/04/24 • [Edit Online](#)

最後のタスクの作成処理は、新しいファイルにセキュリティを割り当てることができます。Windows のセキュリティ モデルは、継承をサポートしています (個々の ACE のエントリがこのような方法でマークされている新しいファイルまたはディレクトリが作成されたときに継承されること) これは、ファイル システムの外部に実装します。そのため、ファイル システム内のロジックの大部分は、新しいセキュリティ記述子を格納する専用です。ルーチンの例を次に示します。

```
NTSTATUS FsdAssignInitialSecurity( PIRP_CONTEXT IrpContext,
                                PFCB Fcb, PFCB Directory)
{
    NTSTATUS status = STATUS_SUCCESS;
    BOOLEAN CreateDir = ((IrpContext->IrpSp->Parameters.Create.Options
        & FILE_DIRECTORY_FILE)==FILE_DIRECTORY_FILE);
    PACCESS_STATE AccessState =
        IrpContext->IrpSp->Parameters.Create.SecurityContext->AccessState;
    PSECURITY_DESCRIPTOR SecurityDescriptor = NULL;

    //
    // Make sure the parent directory's security descriptor is loaded.
    //
    (void) FsdLoadSecurityDescriptor(IrpContext, Directory);

    //
    // don't care about the return code here, as it is handled later
    //
    if (Directory->SecurityDescriptor == NULL) {

        //
        // If the parent has no security, then we are outside
        // of the normal Windows paradigm.
        //
        // The child (that is, the target of the create) will also have
        // a NULL SD.
        //
        // Note that you can always assign security to the file object
        // explicitly at later on.
        //
        return STATUS_SUCCESS;
    }

    //
    // Now create the security descriptor.
    //
    status = SeAssignSecurity(Directory->SecurityDescriptor,
                             AccessState->SecurityDescriptor,
                             &SecurityDescriptor,
                             CreateDir,
                             &AccessState->SubjectSecurityContext,
                             IoGetFileObjectGenericMapping(),
                             PagedPool);

    if (!NT_SUCCESS(status)) {

        return status;
    }

    //
```

```

// Associate the SD with the file; use our own storage so when
// cleanup occurs it is unnecessary to know if the storage came from the
// security reference monitor.
//
Fcb->SecurityDescriptorLength =
    RtlLengthSecurityDescriptor( SecurityDescriptor );

Fcb->SecurityDescriptor = ExAllocatePoolWithTag(PagedPool,
    Fcb->SecurityDescriptorLength, 'DSyM');

if (!Fcb->SecurityDescriptor) {
    //
    // There is no paged pool.
    //
    SeDeassignSecurity(&SecurityDescriptor);
    Fcb->SecurityDescriptorLength = 0;
    return STATUS_NO_MEMORY;
}

RtlCopyMemory(Fcb->SecurityDescriptor, SecurityDescriptor,
    Fcb->SecurityDescriptorLength);

SeDeassignSecurity(&SecurityDescriptor);

//
// Store the SD persistently (this is file system specific).
//
(void) FsdStoreSecurityDescriptor(IrpContext, Fcb);

return STATUS_SUCCESS;
}

```

初期のセキュリティ記述子 (理解の継承の例) を構築するロジックがファイル システム内で処理されないことに注意してください。これは、合わせて、ファイル システム レイヤー内でのセキュリティ記述子进行处理のための単純なモデルです。

IRP のクォータを処理_MJ_を作成します。

2020/04/24 • [Edit Online](#)

いくつかのロジックは、ファイル システム クォータをサポートしている場合は、クォータの情報を取得する可能性がありますも含まれます。ファイル システムを採用している 1 つの戦略に関するクォータ情報のブロックを取得すること [IRP_MJ_作成](#)を後でチェックありのディスパッチ ルーチンによって更新ファイルのサイズを変更できる他の IRP 要求 (削除および書き込み操作など)。

IRP_MJ_クエリ_セキュリティと IRP_MJ_設定_セキュリティ

2020/04/24 • [Edit Online](#)

さいわいなことに、ファイル システムのセキュリティ記述子の取得と実際のストレージは比較的不透明です。これは、ファイル システムで、記述子の理解を必要としない自己相対形式のセキュリティ記述子の性質です。したがって、クエリ操作の処理は、非常に単純な演習では通常です。ファイル システムの実装の例を次に示します。

```
NTSTATUS FsdCommonQuerySecurity( PIRP_CONTEXT IrpContext)
{
    NTSTATUS status = STATUS_SUCCESS;
    PSECURITY_DESCRIPTOR LocalPointer;

    // Need to add code to lock the FCB here

    status = FsdLoadSecurityDescriptor(IrpContext, IrpContext->Fcb);

    if (NT_SUCCESS(status) ) {

        //
        // copy the SecurityDescriptor into the callers buffer
        // note that this copy can throw an exception that must be handled
        // (code to handle the exception was omitted here for brevity)
        //
        LocalPointer = IrpContext->Fcb->SecurityDescriptor;

        status = SeQuerySecurityDescriptorInfo(
            &IrpContext->IrpSp->Parameters.QuerySecurity.SecurityInformation,
            (PSECURITY_DESCRIPTOR)IrpContext->Irp->UserBuffer,
            &IrpContext->IrpSp->Parameters.QuerySecurity.Length,
            &LocalPointer );

        //
        // CACLS utility expects OVERFLOW
        //
        if (status == STATUS_BUFFER_TOO_SMALL ) {
            status = STATUS_BUFFER_OVERFLOW;
        }
    }

    // Need to add code to unlock the FCB here

    return status;
}
```

このルーチンは、実際のセキュリティ記述子を (この実装では、既に読み込まれていない場合、ルーチンがそのセキュリティ記述子のみ読み込むこと) の永続的な記憶域から読み込む外部関数に依存しているに注意してください。セキュリティ記述子は、ファイル システムに対して非透過的であるために、セキュリティ参照モニターは、記述子をユーザーのバッファーにコピーするために使用する必要があります。ここには、このコード サンプルに関して 2 つの点に注意してください。

1. エラー コードの状態の変換_バッファー_すぎます_警告コードの状態に小さな_バッファー_オーバーフローがいくつかの Windows セキュリティ ツールの正しい動作を提供するために必要な。
2. 両方のクエリし、セキュリティ操作のセットが通常行いますので、バッファーには、ユーザーとは、処理でエラーが発生するユーザーのバッファーを直接使用します。このによって管理されている、フラグデバイスのメンバー_ファイル システムによって作成されたオブジェクト。このコードに基づくファイル システムの実装、呼び出し元の関数を使用する必要は、__無効なユーザー

バッファから保護するためのブロックを実行してください。

ファイル システムがストレージからセキュリティ記述子を読み込む方法の詳細 (、FsdLoadSecurityDescriptorこの例では関数)、ファイル システムのセキュリティ記述子のストレージの実装に完全に異なります。

セキュリティ記述子を格納するは、少し複雑です。ファイル システムは、セキュリティ記述子では、ファイル システムは、セキュリティ記述子の共有をサポートしている場合に、既存のセキュリティ記述子が一致するかどうかを判断する必要があります。セキュリティ記述子の一致しないファイル システムは、この新しいセキュリティ記述子の新しい記憶域を割り当てる必要があります。ファイルのセキュリティ記述子を置換するためのルーチンの例を次に示します。

```
NTSTATUS FsdCommonSetSecurity(PIRP_CONTEXT IrpContext)
{
    NTSTATUS status = STATUS_SUCCESS;
    PSECURITY_DESCRIPTOR SavedDescriptorPtr =
        IrpContext->Fcb->SecurityDescriptor;
    ULONG SavedDescriptorLength =
        IrpContext->Fcb->SecurityDescriptorLength;
    PSECURITY_DESCRIPTOR newSD = NULL;
    POW_FCB Fcb = IrpContext->Fcb;
    ULONG Information = IrpContext->Irp->IoStatus.Information;

    //
    // make sure that the FCB security descriptor is up to date
    //
    status = FsdLoadSecurityDescriptor(IrpContext, Fcb);

    if (!NT_SUCCESS(status)) {
        //
        // Something is seriously wrong
        //
        IrpContext->Irp->IoStatus.Status = status;
        IrpContext->Irp->IoStatus.Information = 0;
        return status;
    }

    status = SeSetSecurityDescriptorInfo(
        NULL,
        &IrpContext->IrpSp->Parameters.SetSecurity.SecurityInformation,
        IrpContext->IrpSp->Parameters.SetSecurity.SecurityDescriptor,
        &Fcb->SecurityDescriptor,
        PagedPool,
        IoGetFileObjectGenericMapping()
    );

    if (!NT_SUCCESS(status)) {

        //
        // restore things and return
        //
        Fcb->SecurityDescriptorLength = SavedDescriptorLength;
        Fcb->SecurityDescriptor = SavedDescriptorPtr;
        IrpContext->Irp->IoStatus.Status = status;
        IrpContext->Irp->IoStatus.Information = 0;

        return status;
    }

    //
    // get the new length
    //
    Fcb->SecurityDescriptorLength =
        RtlLengthSecurityDescriptor(Fcb->SecurityDescriptor);

    //
    // allocate our own private SD to replace the one from
    // SeSetSecurityDescriptorInfo so we can track our memory usage
    //
```



```

//
newSD = ExAllocatePoolWithTag(PagedPool,
    Fcb->SecurityDescriptorLength, 'DSyM');

if (!newSD) {

    //
    // paged pool is empty
    //
    SeDeassignSecurity(&Fcb->SecurityDescriptor);
    status = STATUS_NO_MEMORY;
    Fcb->SecurityDescriptorLength = SavedDescriptorLength;
    Fcb->SecurityDescriptor = SavedDescriptorPtr;

    //
    // make sure FCB security is in a valid state
    //
    IrpContext->Irp->IoStatus.Status = status;
    IrpContext->Irp->IoStatus.Information = 0;

    return status;
}

//
// store the new security on disk
//
status = FsdStoreSecurityDescriptor(IrpContext, Fcb);

if (!NT_SUCCESS(status)) {
    //
    // great- modified the in-core SD but couldn't get it out
    // to disk. undo everything.
    //
    ExFreePool(newSD);
    SeDeassignSecurity(&Fcb->SecurityDescriptor);
    status = STATUS_NO_MEMORY;
    Fcb->SecurityDescriptorLength = SavedDescriptorLength;
    Fcb->SecurityDescriptor = SavedDescriptorPtr;
    IrpContext->Irp->IoStatus.Status = status;
    IrpContext->Irp->IoStatus.Information = 0;

    return status;
}

//
// if we get here everything worked!
//
RtlCopyMemory(newSD, Fcb->SecurityDescriptor,
    Fcb->SecurityDescriptorLength);

//
// deallocate the security descriptor
//
SeDeassignSecurity(&Fcb->SecurityDescriptor);

//
// this either is the new private SD or NULL if
// memory allocation failed
//
Fcb->SecurityDescriptor = newSD;

//
// free the memory from the previous descriptor
//
if (SavedDescriptorPtr) {
    //
    // this must always be from private allocation
    //

```

```
        ExFreePool(SavedDescriptorPtr);

    }

    IrpContext->Irp.IoStatus = status;
    IrpContext->Irp.Information = Information;

    return status;
}
```

これは、実装内の領域がファイル システムにファイル システムからと大幅に異なります。たとえば、セキュリティ記述子の共有をサポートするファイル システムは一致するセキュリティ記述子を検索する明示的なロジックを追加する必要があります。このサンプルは、実装ガイダンスを提供する試みのみです。

IRP_MJ_DIRECTORY_CONTROL のセキュリティ チェック

2020/08/04 • [Edit Online](#)

特定のディレクトリ制御操作 (特に変更通知を扱うもの) を処理する場合は、セキュリティが考慮されます。セキュリティ上の問題は、ディレクトリ変更通知によって、変更された特定のファイルに関する情報が返される可能性があるということです。ディレクトリへのパスを走査する権限がユーザーに与えられていない場合、変更に関する情報をユーザーに返すことはできません。それ以外の場合、ユーザーには、ユーザーが所有してはならないディレクトリに関する追加情報を学習するためのメカニズムが用意されています。

ファイルシステムのランタイムライブラリによるディレクトリ変更通知のサポートにより、ファイルシステムは、ディレクトリ変更通知を返す前に走査チェックを実行するためのコールバック関数を指定できます。このコールバック関数は、多数のパラメーターを受け取ります。セキュリティの考慮事項として、次の3つのパラメーターが重要です。

- *Notifycontext*は、変更通知がアクティブになっているディレクトリのコンテキストです。これは、[Fsrtlnotifyfullchangedirectory](#)への呼び出しに渡される*fscontext*パラメーターになります。
- *Targetcontext*は、変更されたファイルのコンテキストです。これは、[Fsrtlnotifyfilterreportchange](#)を呼び出すときにファイルシステムによって渡される*targetcontext*パラメーターになります。
- *Subjectcontext*は、ディレクトリ変更通知を要求しているスレッドのセキュリティコンテキストです。これは、[Fsrtlnotifyfullchangedirectory](#)へのディレクトリ変更通知の呼び出しが行われたときにファイルシステムによってキャプチャされたサブジェクトセキュリティコンテキストです。

変更が発生すると、ファイルシステムによってファイルシステムのランタイムライブラリに対して指定されます。ファイルシステムランタイムライブラリは、ファイルシステムによって提供されるコールバック関数を呼び出して、変更に関する情報を呼び出し元に与えることができるかどうかを確認します。呼び出し元にチェックが必要な場合にのみ、ファイルシステムでコールバック関数を登録する必要があることに注意してください。これは、呼び出し元のセキュリティトークンの `TOKEN_HAS_TRAVERSE_PRIVILEGE` に示されているように、呼び出し元で `SeChangeNotifyPrivilege` が有効になっていない場合に当てはまります。

コールバック関数内では、ファイルシステムは、*Notifycontext*パラメーターによって指定されたディレクトリから、*targetcontext*パラメーターによって指定された変更されたファイルへの走査チェックを実行する必要があります。次のサンプルルーチンでは、このようなチェックを実行します。

```
BOOLEAN
FsdNotifyTraverseCheck (
    IN PDIRECTORY_CONTEXT OriginalDirectoryContext,
    IN PFILE_CONTEXT ModifiedDirectoryContext,
    IN PSECURITY_SUBJECT_CONTEXT SubjectContext
)
{
    BOOLEAN AccessGranted = TRUE;
    PFILE_CONTEXT CurrentDirectoryContext;
    ACCESS_MASK GrantedAccess;
    NTSTATUS Status;
    PPRIVILEGE_SET Privileges = NULL;
    PFILE_CONTEXT TopDirectory;

    //
    // Nothing to do if there is no file context.
    //
    if (ModifiedDirectoryContext == NULL) {

        return TRUE;
    }

    //
```

```

//
// If the directory that changed is the original directory,
// we can return , since the caller has access.
// Note that the directory context is unique to the specific
// open instance, while the modified directory context
// represents the per-file/directory context.
// How these data structures work in your file system will vary.
//
if (OriginalDirectoryContext->FileContext == ModifiedDirectoryContext) {
    return TRUE;
}

//
// Lock the subject context.
//
SeLockSubjectContext(SubjectContext);

for( TopDirectory = OriginalDirectoryContext->FileContext,
      CurrentDirectoryContext = ModifiedDirectoryContext;
      CurrentDirectoryContext == TopDirectory || !AccessGranted;
      CurrentDirectoryContext = CurrentDirectoryContext->ParentDirectory) {
    //
    // Ensure we have the current security descriptor loaded for
    // this directory.
    //
    FsdLoadSecurity( NULL, CurrentDirectoryContext);

    //
    // Perform traverse check.
    //
    AccessGranted = SeAccessCheck(
        CurrentDirectoryContext->SecurityDescriptor,
        SubjectContext,
        TRUE,
        FILE_TRAVERSE,
        0,
        &Privileges,
        IoGetFileObjectGenericMapping(),
        UserMode,
        &GrantedAccess,
        &Status);

    //
    // At this point, exit the loop if access was not granted,
    // or if the parent directory is the same as where the change
    // notification was made.
    //
}

//
// Unlock subject context.
//
SeUnlockSubjectContext(SubjectContext);

return AccessGranted;
}

```

このルーチンは、セキュリティ情報をキャッシュするファイルシステムや、ファイルやディレクトリ間のリンクを追跡するための構造を使用するファイルなど、ファイルシステムによって大幅に異なる可能性があります。このサンプルでは、この例を簡略化するために、リンクをサポートしているファイルシステムは考慮されていません。

IRP_MJ_FILE_SYSTEM_CONTROL のセキュリティ チェック (IFS)

2020/08/04 • [Edit Online](#)

ファイルシステムコントロールを使用すると、ファイルシステムは基本的に特化された操作を実行できます。既存の Windows ファイルシステムには、いくつかの特殊な制御があります。また、Windows 用に開発されたサードパーティのファイルシステムでも、すべてのパラメーターを積極的に確認することが不可欠です。また、FSCTL 操作には、多くの場合、制限付きのセキュリティ権限があります。これらは、[FASTFAT サンプルコード](#)で確認できます (例については、Fsctrl のFatInvalidateVolumes関数を参照してください)。これは、特権チェックの一例です。この場合、FASTFAT ファイルシステムのポリシーは、指定された特権がシステムで有効になっていることを要求します。

ファイルシステムが CTL_CODE マクロを使用してファイルシステム操作の定義でこれらのビットを設定している場合、i/o マネージャーは、特定の FSCTL 操作に対して FILE_READ_DATA および FILE_WRITE_DATA のアクセス許可を適用します。ファイルシステムのポリシーの場合は、ファイルシステムによって必要なその他すべてのアクセス許可 (FILE_READ_ATTRIBUTES アクセス許可など)を確認する必要があります。

IRP_MJ_SET_INFORMATION のセキュリティチェック

2020/08/04 • [Edit Online](#)

設定情報の名前変更とハードリンクのケースでは、特定の状況下でセキュリティチェックが必要になる場合があります。具体的には、呼び出し元が置換 `Eifexists` フィールドを `TRUE` に設定して、名前の変更またはハードリンクのターゲットを削除する場合、ファイルシステムはセキュリティチェックを実行して、ターゲットを削除するための適切なアクセス許可が呼び出し元に与えられていることを確認する必要があります。さらに、ポリシーに関係なくファイルシステムがこの方法での削除を許可しないファイルの種類が存在する場合があります (レジストリハイブやページングファイルなど)。次のコード例では、呼び出し元に、ファイルを削除するための適切なセキュリティアクセス許可があるかどうかを判断します。

```
NTSTATUS FsdCheckDeleteFileAccess(POW_IRP_CONTEXT IrpContext,
                                PSECURITY_DESCRIPTOR targetSD,
                                PFCB ParentFcb)
{
    SECURITY_SUBJECT_CONTEXT SubjectContext;
    BOOLEAN Granted;
    NTSTATUS status = STATUS_SUCCESS;
    PPRIVILEGE_SET Privileges = NULL;
    ACCESS_MASK GrantedAccess;

    //
    // See if the user has DELETE access to the target.
    //
    SeCaptureSubjectContext( &SubjectContext );

    SeLockSubjectContext( &SubjectContext );

    Granted = SeAccessCheck(targetSD,          // Target's SD.
                            &SubjectContext,  // Captured security context.
                            TRUE,              // Tokens are locked.
                            DELETE,           // we only care about delete
                            0,                 // previously granted access.
                            &Privileges,      // privilege_set
                            IoGetFileObjectGenericMapping(), // Generic mappings.
                            UserMode,         // Mode
                            &GrantedAccess,   // Granted access mask
                            &status );        // Error code

    //
    // Do not need privilege set, so release it.
    //
    if (Privileges != NULL) {
        SeFreePrivileges( Privileges );
        Privileges = NULL;
    }

    if (!Granted) {
        status = STATUS_SUCCESS;

        //
        // The user does not have DELETE access to the target, but
        // could have FILE_DELETE_CHILD access to the parent directory.
        //
        (void) FsdLoadSecurityDescriptor(IrpContext, ParentFcb);
        if (!ParentFcb->SecurityDescriptor) {
            //

```

```

        // fine - no security is fine - the user gets to do what they want
        //
        SeUnlockSubjectContext( &SubjectContext );
        SeReleaseSubjectContext( &SubjectContext );
        return STATUS_SUCCESS;
    }

    Granted = SeAccessCheck(&ParentFcb->SecurityDescriptor,
                            &SubjectContext,    // Captured security context.
                            TRUE,                // Tokens are locked.
                            FILE_DELETE_CHILD,  // we only care about delete
                            0,                   // Previously granted access.
                            &Privileges,        // privilege_set
                            IoGetFileObjectGenericMapping(), // Generic mappings
                            UserMode,           // mode
                            &GrantedAccess,     // Granted access mask
                            &status );          // Error code

    //
    // Release privileges
    //
    if (Privileges != NULL) {
        SeFreePrivileges( Privileges );
        Privileges = NULL;
    }
}
SeUnlockSubjectContext( &SubjectContext );
SeReleaseSubjectContext( &SubjectContext );
return status;
}

```

このコードは、名前変更とハードリンク作成ケースの両方に使用できます。

このドキュメントでは、削除するファイルの種類に基づいてファイルシステムで削除が禁止されていると判断されたポリシーレベルのコードについて説明します。

偽装

2020/04/24 • [Edit Online](#)

ファイルシステムによっては、元の呼び出し元に代わって操作を実行すると便利な場合があります。たとえば、ネットワークファイルシステムでは、ファイルを開いたときに呼び出し元のセキュリティ情報をキャプチャし、適切な資格情報を使用して後続の操作を実行できるようにすることが必要になる場合があります。当然のことですが、この種の機能は、ファイルシステム内と特定のアプリケーションの両方で役に立ちます。

偽装に必要な主なルーチンは次のとおりです。

- **SeImpersonateClientEx**は偽装を開始します。特定のスレッドが指定されていない限り、偽装は現在のスレッドコンテキストで実行されます。
- **Psreverttoself**--現在のスレッドコンテキスト内で偽装を終了します。
- **Psreferenceprimarytoken**--指定されたプロセスのプライマリ (プロセス) トークンに対する参照を保持します。この関数は、システム上の任意のプロセスのトークンをキャプチャするために使用できます。
- **PsDereferencePrimaryToken**--以前に参照されたプライマリトークンで参照を解放します。
- **SeCreateClientSecurityFromSubjectContext**--クライアントのセキュリティコンテキストを返します。サブジェクトコンテキストからの権限借用に役立ちます (IRP_MJ_CREATE処理中に FSD に提供されます)。
- **SeCreateClientSecurity**--システム上の既存のスレッドのセキュリティ資格情報に基づいて、クライアントのセキュリティコンテキストを作成します。
- **ImpersonateSecurityContext**--カーネルセキュリティサービスである ksecdd.sys 内のセキュリティコンテキストを偽装します。
- **RevertSecurityContext**--カーネルセキュリティサービスである ksecdd.sys 内の偽装を終了します。

偽装は、を実装するための簡単な方法です。次のコード例は、基本的な偽装を示しています。


```

NTSTATUS PerformSpecialTask(IN PFSD_CONTEXT Context)
{
    BOOLEAN CopyOnOpen;
    BOOLEAN EffectiveOnly;
    SECURITY_IMPERSONATION_LEVEL ImpersonationLevel;
    NTSTATUS Status;
    PACCESS_TOKEN oldToken;

    //
    // We need to perform a task in the system process context
    //
    if (NULL == Context->SystemProcess) {

        return STATUS_NO_TOKEN;

    }

    //
    // Save the existing token, if any (otherwise NULL)
    //
    oldToken = PsReferenceImpersonationToken(PsGetCurrentThread(),
                                              &CopyOnOpen,
                                              &EffectiveOnly,
                                              &ImpersonationLevel);

    Status = PsImpersonateClient( PsGetCurrentThread(),
                                  Context->SystemProcess,
                                  TRUE,
                                  TRUE,
                                  SecurityImpersonation);

    if (!NT_SUCCESS(Status)) {

        if (oldToken)
            PsDereferenceImpersonationToken(oldToken);
        return Status;

    }

    //
    // Perform task - whatever it is
    //

    //
    // Restore to previous impersonation level
    //
    if (oldToken) {
        Status = PsImpersonateClient(PsGetCurrentThread(),
                                      oldToken,
                                      CopyOnOpen,
                                      EffectiveOnly,
                                      ImpersonationLevel);

        if (!NT_SUCCESS(Status)) {
            //
            // This is bad - we can't restore, we can't leave it this way
            //
            PsRevertToSelf();
        }
        PsDereferenceImpersonationToken(oldToken);
    } else {
        PsRevertToSelf();
    }

    return Status;
}

```

この偽装コードには、ファイルシステムの開発者が使用できるさまざまなバリエーションがありますが、その手法の基本的な例を示します。

プロセスとスレッドの終了に関する問題

2020/04/24 • [Edit Online](#)

特定のユーザーに関連する状態情報を格納するファイルシステムでは、プロセスとスレッド終了の状態を監視する必要がある場合があります。たとえば、特定のユーザーに関連付けられている暗号化キーは、特殊なコントロールアプリケーションの終了 (予定されているかどうかに関係なく) によって破棄される必要があります。これらの条件の処理に使用されるルーチンの詳細については、「[Psssetcreateprocessnotifyroutine](#)」と「[Psssetcreateprocessnotifyroutine](#)」を参照してください。

ファイル システム フィルター ドライバーでのプロキシ操作

2020/04/24 • [Edit Online](#)

ファイルシステムフィルタードライバーは、元の (ユーザーモード) 呼び出し元に代わって操作を頻繁に実行する必要があります。この場合、ファイルシステムフィルタードライバーは、元のユーザーが操作できなかった操作を実行しないようにする必要があります。たとえば、ユーザーがファイル_置き換えるファイルを開こうとしたとします。フィルタードライバーは、[Zwcreatefile](#)を使用して同じ種類のアクセスを指定するファイルを開こうすることはできません。たとえば、ユーザーがファイルを置き換えるアクセス許可を持っていない場合でも、ファイルシステムによって実行される操作は成功します。フィルタードライバー。

ファイルシステムフィルタードライバーによってこのような問題が発生する可能性がある方法は多数あります。ファイルシステムフィルタードライバーが、ユーザーの操作の結果を知らなくても、基になるファイルシステムに対して操作を実行するたびに発生する可能性があります。そのため、ファイルシステムフィルタードライバーは、このようなケースを特定し、操作の結果を判断したか、または実際のユーザー操作でエラーから復旧するためのメカニズムを備えている必要があります。たとえば、ファイルを置き換える要求の場合、フィルタードライバーは、置き換え操作に十分なファイルのセキュリティ権限を示す[Iocreatefile](#)を使用して、元の呼び出し元の代わりにファイルを開く必要があります。フィルタードライバーが IO_ 使用する場合は、_アクセス_チェックオプションを使用します。たとえば、カーネルドライバーからの呼び出しであっても、現在のスレッドの資格情報でセキュリティチェックが行われます。

ファイルシステムフィルタードライバーは、ユーザーレベルの操作によって、またはその結果として、ドライバーが操作を実行しているインスタンスを識別するために不可欠です。このような場合は、適切な操作を確実に識別する方法を明確にする方法があります。

ファイル システム フィルター ドライバーでの偽装

2020/04/24 • [Edit Online](#)

ファイルシステムフィルタードライバーが使用しようとしている可能性のある別の操作は、偽装です。偽装は、他のスレッドの代わりにセキュリティを処理するための非常に強力な手法ですが、コンポーネントの代わりに使用するための適切な注意も必要です。ファイルシステムフィルタードライバーの場合は、偽装を使用して実行する必要がある操作を識別することが重要です。次に、ファイルシステムフィルタードライバーによって実行される他の操作を、偽装を使用して実行しないようにする必要があります。偽装のリスクは、通常、呼び出し元のドライバーよりも呼び出し元の特権が低いことを示します。したがって、偽装を使用して呼び出しが行われた場合、偽装を行わずに成功する可能性があります、失敗する可能性があります。

ハンドルはオブジェクトへの参照を表し、セキュリティチェックが実行されたポイントであるため、新しいハンドルを作成するすべての操作には偽装が必要です。たとえば、ファイルまたは他のオブジェクト ([Zw/ zwcreateevent](#)、[Zwcreateevent](#)、[zwcreatefile](#)など) を開くときには偽装が必要になります。これらの呼び出しでは、これらを呼び出すフィルタードライバーが、渡されたパラメーターが有効であることを確認する必要があります。これは、他のオペレーティングシステム操作では、カーネルモードからの呼び出しに有効なパラメーターがあると想定されるためです。したがって、フィルタードライバーは、偽装している場合でも、これらの関数のいずれかにユーザーバッファアドレスを安全に渡すことができません。

[Zwcreatefile](#)の場合は、対応する i/o Manager 呼び出し[iocreatefile](#)を使用する必要があります。これは、フィルタードライバーが `IO_` 指定して `_アクセス_` チェックを強制することができるためです。このオプションを指定しないと、i/o マネージャーは適切なユーザーレベルのアクセスチェックを強制しません。

他のファイル システム フィルター ドライバーとの共存

2020/04/24 • [Edit Online](#)

ファイル システム フィルター ドライバーによって適切に処理する必要がありますのある最も厄介な問題の 1 つは、他のフィルター ドライバーとの共存です。その他のファイル システム フィルター ドライバーと共存させるには、ファイル システム フィルター ドライバーをビルドする場合は、次の問題を考慮することをお勧めします。

- フィルター ドライバーは、その操作では、他のフィルター ドライバーの存在を考慮してください。フィルター ドライバーによって実行された操作は、同じまたは別の手法を使用して、追加のフィルター ドライバーを維持するのに十分な堅牢な必要があります。
- フィルター ドライバーは、システムの基本動作を変更して他のフィルターの動作の影響を与える可能性があります。
- フィルター ドライバーの数を増やすと、特にスタック領域の不足しているリソースの消費量が増加します。ファイル システム フィルター ドライバーは、このような不足しているリソースの使用を最小限に抑えるよう努める必要があります。それ以外の場合、悪意のあるユーザーのアプリケーションには、システムに障害が発生するには、このような弱点の利点がかかります。開発者は、入力候補のパスとエラー パスについて特に注意必要があります。
- フィルター ドライバーはできるだけ控えめに下位のドライバー (ドライバーのフィルターまたはファイル システム) に送信する必要があります、自由に承諾する必要があります。フィルター ドライバーが基になるドライバーに送信する操作が単純でない複雑なことを確認しようとして可能な限り、(名前の変更を実行しない操作中に、操作を作成します)。
- フィルター ドライバーは、ロックに関する注意が必要である必要があります。ファイル システムの呼び出しの間でロックを保持しない必要があります。システムのさまざまなコンポーネントを推測非常に正確と明示的なロックの順序付けと関数できますをブロックすることはできません。ロックの別のレイヤーを追加することでこの影響を及ぼすことは簡単にデッドロックが発生します。Srv.sys から I/O が特に簡単には、これらの問題を公開しますが、通常のストレス テストも中に、見られることができます。

命令型のファイル システム フィルター ドライバー開発者が設計だけでなく、他のフィルター ドライバーと正常に共存しますが、も、ドライバーが内でセキュリティ上の問題を導入していないことを確認するには、他のフィルター ドライバー、フィルター ドライバーをテストするために実装システム。

ファイル システム フィルター ドライバーのメモリ マップ ファイル

2020/04/24 • [Edit Online](#)

ファイル システム フィルター ドライバー ファイルが読み取りを介してではなく、ファイルの仮想メモリマッピングを使用してアクセスできるし、書き込みパスのファクトの理解している必要があります。ファイル内の変更を監視するファイル システム フィルター ドライバーには、このようなファイルへの変更が不足します。一般にメモリ マップ I/O を処理する必要があるファイル システム フィルター ドライバーをページング I/O のフィルターを適用する必要があります。これに対処するための手法は、Windows ファイル システムの開発者の一覧で説明([NTFS](#))ニュースグループ OSR によってホストされています。

ファイル システム フィルター ドライバーの再解析ポイント

2020/04/24 • [Edit Online](#)

フィルター ドライバーがプロセスに再解析ポイントは、アプリケーション プログラムが無効な再解析ポイントを作成する場合があります。リスクに注意してくださいである必要があります。チェックサムをセキュリティで保護された、暗号化された内容、または無効な再解析ポイントに確実にその他のメカニズムを通じて、検証可能なハンドルの再解析ポイントがデータの内容の再解析ポイント自体ことを確認する必要があることドライバー最も厳格なセキュリティを確保するには特権のないアプリケーションを作成できません。たとえば、フィルター ドライバーが、再解析ポイントがあることを要求する可能性がありますアプリケーション (または、たとえば、ローカル セキュリティ機関) の間で共有されているパスワードを使用して暗号化し、再解析のデータの内容が指していることを確認するには、ドライバーは無効です。

それ以外の場合、悪意のあるアプリケーションが無効な再解析ポイントの情報の再解析ポイントを作成することが可能です。この場合は、自己参照データ (たとえば何らかのオーバーフローを引き起こす可能性のある参照ループを作成するデータ) を含む、無効な再解析ポイントのデータ、データのオーバーフロー問題、および無効なデータを処理するために、ファイル システム フィルター ドライバーを準備する必要があります。内容。

Windows コンテナ用のウイルス対策最適化

2020/08/04 • [Edit Online](#)

このページの情報は、次のものに適用されます。

- Windows 10 バージョン1607以降
- Windows Server 2016 以降のバージョン
- ホストで実行されているウイルス対策 (AV) 製品

このトピックでは、Windows コンテナファイルの冗長スキャンを回避し、コンテナの起動時間を向上させるために AV 製品が利用できる最適化について説明します。

コンテナの概要

Windows コンテナ機能は、アプリケーションの配布と展開を簡略化するように設計されています。詳細については、「[Windows コンテナの概要](#)」を参照してください。

コンテナは、任意の数のパッケージレイヤーから構築されます。Windows ベース OS パッケージは、最初のレイヤーを形成します。

各コンテナには、そのコンテナのシステムボリュームを表す分離ボリュームがあります。コンテナ分離フィルター (*wcifs.sys*) は、このコンテナボリュームにパッケージレイヤーの仮想オーバーレイを提供します。オーバーレイは、プレースホルダー (再解析ポイント) を使用して実現されます。コンテナが最初に *overlay* パスにアクセスする前に、プレースホルダーを使用してボリュームがシード処理されます。プレースホルダーファイルの読み取りは、バックアップパッケージファイルに送られます。このようにして、複数のコンテナボリュームが同じ基になるパッケージファイルデータストリームにアクセスできます。

コンテナによってファイルが変更された場合、分離フィルターでは、書き込み時のコピーが実行され、プレースホルダーがパッケージファイルの内容に置き換えられます。これにより、特定のコンテナのパッケージファイルへの "リンケージ" が中断されます。

リダイレクトの読み取り

プレースホルダーファイルからの読み取りは、分離フィルターによって適切なパッケージレイヤーにリダイレクトされます。リダイレクトは、フィルターのレベルで実行されます。フィルターは AV 範囲を下回るため、AV フィルターでは読み取りリダイレクトが表示されません。また、リダイレクトを設定するために実行されたパッケージファイルが開きません。

AV フィルターには、コンテナシステムボリューム上のすべての操作が完全に表示されます。プレースホルダーファイルに加えて、ファイルの変更や新しいファイルの追加に関する操作も表示されます。

冗長スキャンの問題

多くの場合、同じパッケージレイヤーに応じて多数のコンテナが存在します。指定されたパッケージファイルの同じデータストリームによって、複数のコンテナシステムボリューム上のプレースホルダーのデータが提供されます。その結果、すべてのコンテナの同じデータに対して、重複する AV スキャンが発生する可能性があります。これにより、コンテナのパフォーマンスに不要な悪影響が生じます。これは、コンテナがすぐに開始されることが予想され、有効期間が短くなる可能性があるため、signification コストです。

推奨されるアプローチ

コンテナでの冗長なスキャンを回避するには、以下で説明するように、AV 製品で動作を変更することをお勧めします。このアプローチでは、お客様に対するリスク/報酬特典を決定するために、AV 製品が必要です。詳細については、このページの下部にある「[利点とリスク](#)」を参照してください。

1. パッケージのインストール

パッケージのインストール中に、管理ツールによってパッケージ内のファイルがレイヤールートに配置されます。AV フィルタは、パッケージルートに配置されている間はファイルのスキャンを続行し、通常はファイルのスキャンします。これにより、レイヤー内のすべてのファイルがマルウェアに対して最初にクリーンであることが保証されます。

2. コンテナの開始と実行

コンテナボリュームをリアルタイムでスキャンするには、重複を回避する方法で AVs をスキャンする必要があります。プレースホルダーファイルには特別な考慮が必要です。コンテナによって変更されたファイルまたはコンテナ内に作成された新しいファイルはリダイレクトされないため、冗長スキャンが問題になることはありません。

重複するスキャンを回避するには、まず、そのボリュームのコンテナボリュームとプレースホルダーを、AV フィルターで識別する必要があります。さまざまな理由により、ボリュームがコンテナボリュームであるか、または特定のファイルがプレースホルダーファイルであるかどうかをクエリするための直接の方法はありません。分離フィルターでは、アプリケーションの互換性のためにプレースホルダーの再解析ポイントが非表示になります (再解析ポイントにアクセスしていることが認識されている場合、一部のアプリケーションは正しく動作しません)。また、コンテナが実行されている間、ボリュームはコンテナボリュームにすぎません。コンテナが停止し、ボリュームが再マウントされる可能性があります。代わりに、作成前に、AV フィルターは、ファイルオブジェクトに対してクエリを実行して、コンテナのコンテキストで開かれているかどうかを判断する必要があります。次に、作成完了時に、作成して、プレースホルダー状態を受信します。

AV 製品では、次の変更が必要です。

- コンテナボリュームでの事前作成時に、プレースホルダー情報を受信する [作成] の [送信] データに ECP をアタッチします。これらの作成は、IoGetSiloParametersを使用して、FILEOBJECT からサイロパラメーターを照会することによって識別できます。注フィルターではWCIFS_REDIRECTION_ECP_CONTEXT構造のサイズを指定する必要があります。他のすべてのフィールドは、ECP が確認された場合に、out フィールドに設定されます。
- [作成後] で、ECP が確認された場合は、ECP リダイレクトフラグを確認します。フラグは、開いているがパッケージレイヤーまたはスクラッチルート (新規または変更されたファイル) からサービスを処理したかどうかを示します。フラグは、パッケージレイヤーが登録されているかどうか、およびリモートであるかどうかを示します。

- リモートレイヤーからサービスが提供されている場合、AV はファイルのスキャンをスキップします。これは、リダイレクトフラグによって示されます。

```
WCIFS_REDIRECTION_FLAGS_CREATE_SERVICED_FROM_LAYER &&  
WCIFS_REDIRECTION_FLAGS_CREATE_SERVICED_FROM_REMOTE_LAYER
```

リモート層は、リモートホストでスキャンされていると見なすことができます。Hyper-v コンテナパッケージは、コンテナをホストするユーティリティ VM に対してリモートです。これらのパッケージは、ユーティリティ VM から SMB ループバックを介してアクセスされたときに、Hyper-v ホスト上で正常にスキャンされます。

VolumeGUID と FileId はリモートでは適用されないため、これらのフィールドは設定されません。

- 登録されたレイヤーからサービスが提供される場合、AV はファイルのスキャンをスキップします。これは、リダイレクトフラグによって示されます。

```
WCIFS_REDIRECTION_FLAGS_CREATE_SERVICED_FROM_LAYER &&  
WCIFS_REDIRECTION_FLAGS_CREATE_SERVICED_FROM_REGISTERED_LAYER
```

登録されたレイヤーは、パッケージのインストール中および署名の更新後に、非同期的にスキャンする必要があります。

NOTE

登録済みのレイヤーは、将来システムによって識別されない場合があります。この場合、ローカルレイヤーファイルは、最後の箇条書きで説明されているように、個別に識別する必要があります。

- ローカルパッケージレイヤーからサービスが提供されるオープンの場合、AV では、レイヤーファイルの指定された volumeguid と FileId を使用して、ファイルのスキャンする必要があるかどうかを判断する必要があります。これには、ボリューム GUID と FileId によってインデックス付けされたスキャン済みファイルのキャッシュを作成するために、AV が必要になる可能性があります。これはリダイレクトフラグによって示されます。

WCIFS_REDIRECTION_FLAGS_CREATE_SERVICED_FROM_LAYER

- 新規または変更されたファイルがスクラッチの場所にある場合、AV 製品はファイルをスキャンし、通常の修復を実行します。これはリダイレクトフラグによって示されます。WCIFS_REDIRECTION_FLAGS_CREATE_SERVICED_FROM_SCRATCH

この場合、レイヤーファイルは存在しないため、VolumeGUID と FileId は設定されません。

- "このファイルはレイヤーからサービスされています" という名前のストリームコンテキストで永続的なマーカーとして保存しないでください。最初にレイヤールートから処理されるファイルは、作成後に変更される可能性があります。この場合、同じファイルの後続の作成では、作成がコンテナボリュームからサービスされていることを示している可能性があります。この問題が発生する可能性があることを、AV フィルタが理解している必要があります。

レイヤー Root場所レジストリキーは使用しないでください

以前は、レジストリキーを使用して LayerRootLocations 基本イメージの場所を取得することをお勧めします。AV 製品では、このレジストリキーを使用できなくなります。代わりに、このトピックで推奨されている方法を使用して、重複スキャンを回避してください。

パッケージレイヤーの登録に使用されたレジストリの場所:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Virtualization\LayerRootLocations

利点とリスク

これらの新しい最適化を AV 製品に使用する場合は、次の利点とリスクについて検討してください。

メリット

- コンテナの開始または実行時間には影響しません (最初のコンテナであっても)。
- 複数のコンテナ内の同じコンテンツのスキャンを回避します。
- Windows Server コンテナに対して機能します。Hyper-v コンテナの場合、これはパッケージに対して機能しますが、コンテナを実行するには追加の作業が必要です。

リスク

署名の更新と次にスケジュールされている予防的なマルウェア対策スキャンの間にコンテナが起動した場合、コンテナで実行されたファイルは、最新のマルウェア対策の署名に関してスキャンされません。このリスクを軽減するために、最後のプロアクティブスキャン以降の署名の更新がない場合にのみ、AV 製品でリダイレクトされたファイルのスキャンをスキップできます。これにより、最新の署名でプロアクティブスキャンが完了するまで、コンテナのパフォーマンスの低下が制限されます。必要に応じて、AV 製品はこの状況でプロアクティブスキャンをトリガーできます。これにより、以降のコンテナの起動が効率的になります。

オフロード データ転送

2020/04/24 • [Edit Online](#)

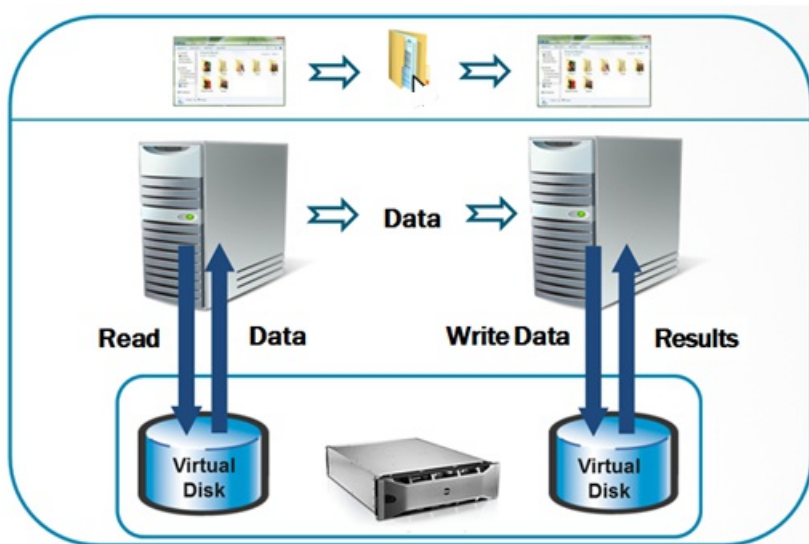
コンピューター間または同じコンピューター内でのデータ転送は、ファイルシステムの頻繁な動作です。標準のReadFile関数とWriteFile関数を使用することは、機能的な観点からは適切に機能しますが、システムのあらゆるレベルでの大量のデータ移動と、ネットワーク経由の可能性があります。これは、転送に関係するシステムとシステムを接続するネットワークの可用性に影響を与える可能性があります。多くのストレージサブシステムで利用できる高度な機能を使用すると、データ移動タスクをより効率的に実行できます。

Windows 8以降では、アプリケーションはこれらの機能を利用して、記憶域サブシステムへのデータ移動プロセスをオフロードすることができます。通常、ファイルシステムフィルターは、ボリュームに対する読み取りおよび書き込み要求をインターセプトすることによって、これらのアクションを監視できます。フィルターでオフロードデータ転送を認識するには、追加のアクションが必要です。

標準的なデータ転送

現在、アプリケーションのシナリオでデータを移動するのは非常に簡単です。ローカルメモリにデータを読み取ってから、新しい場所に書き戻す必要があります。次の図は、このシナリオを示しています。

このシナリオでは、2つの異なるファイルサーバー上の2つの場所の間でファイルをコピーします。それぞれのファイルサーバーには、インテリジェントストレージアレイ (ISA) を介して公開される独自の仮想ディスクが含まれます。開始システムは、まず、ソース仮想ディスクからローカルバッファにデータを読み取る必要があります。次に、何らかのトランスポートとプロトコル (1 Gbe を超える SMB など) を使用してデータをパッケージ化し、2番目のシステムに送信します。このシステムは、データを受信してローカルバッファに出力します。次に、ターゲットシステムによって、データが宛先仮想ディスクに書き込まれます。このシナリオでは、毎日多くの異なるアプリケーションによって複数回実行されるデータ転送の非常に一般的な読み取り/書き込み方法について説明します。



標準的な読み取りと書き込みは、ほとんどのシナリオで適切に動作しますが、コピーされるデータは、同じインテリジェントストレージアレイによって管理される仮想ディスクに配置される場合があります。これは、データが配列からサーバーに移動され、ネットワークトランスポート経由で別のサーバーに移動し、もう一度同じ配列に戻されることを意味します。サーバー内およびネットワークトランスポート間でデータを移動することは、これらのシステムの可用性に大きな影響を与える可能性があります。データ移動のスループットは、ネットワークのスループットと可用性によって制限されるという事実には触れません。

オフロードデータ転送 (ODX)

データ転送のオフロード

Windows 8では、データ転送のオフロード方法を容易にする2つの新しい FSCTLs が導入されています。これにより、サーバーから

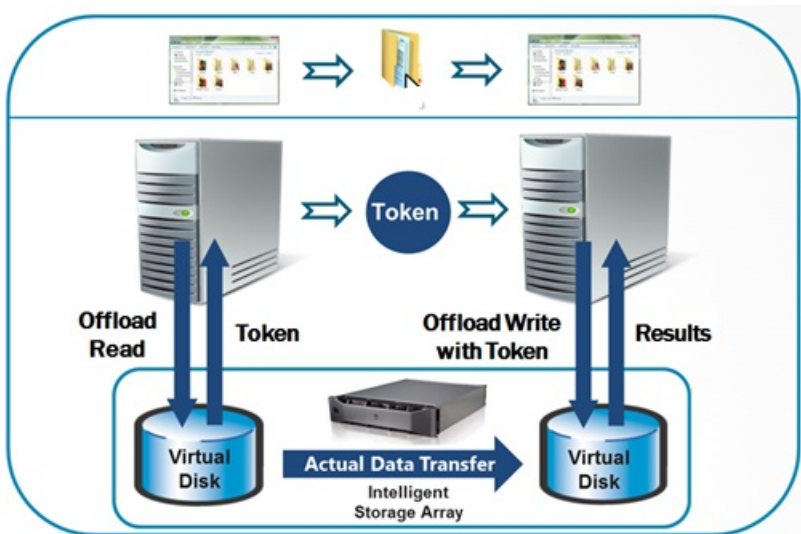
のビット移動の負担が、記憶域サブシステム内でインテリジェントに発生するように変化します。コマンドのセマンティクスを視覚化する最良の方法は、バッファなしの読み取りとバッファなしの書き込みに似ていると考えることです。

FSCTL_オフロード_読み取り

このコントロール要求は、読み取り対象のファイル内のオフセットと、**FSCTL_オフロード_読み取り_入力**構造体の必要な長さを受け取ります。サポートされている場合、ファイルをホストする記憶域サブシステムは、関連付けられているオフロード読み取りストレージコマンドを受け取り、トークンを生成します。これは、オフロード読み取りコマンドの時点で読み取られることを意図したデータの論理表現です。このトークン文字列は、**FSCTL_オフロード_読み取り_出力**構造で呼び出し元に返されます。

FSCTL_オフロード_書き込み

この制御要求では、書き込み先のファイル内のオフセット、書き込みに必要な長さ、および書き込まれるデータの論理的な表現であるトークンが使用されます。サポートされている場合、書き込まれるファイルをホストする記憶域サブシステムは、関連付けられているオフロード書き込みストレージコマンドを受け取ります。まず、指定されたトークンを認識し、次に可能な場合は書き込み操作を実行します。書き込み操作は Windows の下で完了します。そのため、ファイルシステムとストレージスタック上のコンポーネントにデータ移動は表示されません。データの移動が完了すると、書き込まれたバイト数が呼び出し元に返されます。



最初の図と同様に、2つの異なるサーバー上の2つの仮想ディスク間の単純なファイルコピーが示されています。通常の読み取りと書き込みを行うのではなく、ストレージレイへのビット移動の大量の処理をオフロードします。最初のシステムはオフロード読み取り操作を実行し、配列は、最初の仮想ディスクの領域内で読み取るデータの特定の時点のビューを表すトークンを生成するように要求します。その後、最初のシステムはトークンを2番目のシステムに送信します。その後、トークンを使用して2番目の仮想ディスクにオフロード書き込み操作が発行されます。次に、配列はトークンを解釈し、仮想ディスク間のデータ移動を実行しようとします。実際のデータ転送はインテリジェントストレージレイ内で行われ、2つのホスト間では発生しないことがわかります。これにより、システム間のネットワークトラフィックを実質的に排除しながら、2つのシステムの可用性を大幅に向上させることができます。

コピーエンジンとの統合

Windows のコアコピーエンジンは、CopyFileおよび関連する関数によって使用されます。Windows 8 以降では、コピーエンジンは、従来のコピーファイルコードパスの前に、オフロードデータ転送を透過的に使用しようとします。コピー Api はほとんどのアプリケーション、ユーティリティ、およびシェルによって使用されるため、これらの呼び出し元は、既定でオフロードデータ転送機能を使用することができます (コードの変更やユーザーの介入はほとんどありません)。

次の手順は、コピーエンジンがオフロードデータ転送を試行する方法をまとめたものです。

1. コピーエンジンは、読み取りトークンを取得するために、ソースファイルに対する**読み取り_FSCTL_オフロード**を発行します。
2. 読み取りトークンの取得でエラーが発生した場合、コピーエンジンは従来の読み取りと書き込み (従来のコピーファイルのコードパス) にフォールバックします。障害が、ソースボリュームがオフロードをサポートしていないことを示している場合、コピーエンジンは、プロセスごとのキャッシュのボリュームをマークします。コピーエンジンは、プロセスごとのキャッシュ内のボリュームに対して、オフロードを試行しません。
3. トークンが正常に取得された場合、コピーエンジンは**FSCTL_OFFLOAD**を発行しようとします。これは、トークンによって論理的に表されるすべてのデータがオフロードされるまで、サイズが大きいチャンクのターゲットファイルに対して書き込みコマンドを実行します。

4. オフロードの読み取りまたは書き込みを実行中にエラーが発生した場合は、コピーエンジンが読み取りと書き込みの従来のコードパスに戻り、オフロードコードパスが終了した場所から開始されます (読み取りまたは書き込みが切り捨てられた場所)。移行先ボリュームがオフロードをサポートしていないか、ソースボリュームがターゲットボリュームに接続できないことがエラーによって示された場合、コピーエンジンは同じプロセスごとのキャッシュを更新し、これらのボリュームでオフロードを試行しないようにします。このプロセスごとのキャッシュは定期的にリセットされます。

次の関数は、オフロードデータ転送をサポートしています。

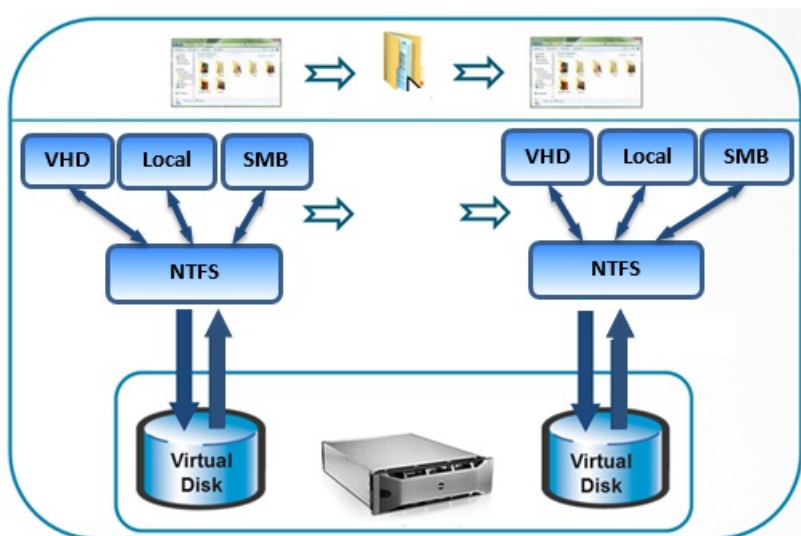
- CopyFile
- CopyFileEx
- My.computer.filesystem.movefile
- MoveFileEx
- CopyFile2

オフロードデータ転送は、次の関数ではサポートされていません。

- CopyFileTransacted
- MoveFileTransacted

サポートされているオフロードデータ転送シナリオ

オフロード操作のサポートは、Hyper-v 記憶域スタックと Windows SMB ファイルサーバーに用意されています。バックアップ物理記憶域が ODX 操作をサポートする場合、呼び出し元は **FSCTL_オフロード** を発行できます。また、読み取りと **FSCTL_オフロード_vhd** またはリモートファイル共有に存在するファイルへの書き込みを、仮想マシン内から実行することもできます。コンピューターまたは物理ハードウェア上。次の図は、オフロードデータ転送のために最も基本的なサポートされているソースとターゲットを示しています。



ファイルシステムフィルターのオプトインモデルとアプリケーションへの影響

Windows 8以降では、フィルターマネージャーを使用して、サポートされている機能としてオフロード機能を指定できます。ボリュームにアタッチされたファイルシステムフィルターは、特定のオフロード操作がサポートされているかどうかをまとめて判断できます。そうでない場合は、適切なエラーコードで操作が失敗します。

フィルターは、**FSCTL_オフロード_読み取り**と**FSCTL_オフロード**をサポートしていることを示す必要があります。また、driver Service 定義にあるSupportedfeaturesというレジストリDWORD値を使用して書き込みを行います。レジストリの HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\<フィルタードライバー名>にあります。この値にはビットフィールドが含まれており、どの機能がオプトインされているかは、フィルターのインストール時に設定する必要があります。

現在、定義されているビットは次のとおりです。

FLAG	II
サポートされている_FS_機能_オフロード_読み取り0x00000001	フィルターは、FSCTL_オフロード_読み取りをサポートします
サポートされている_FS_機能_オフロード_書き込み0x00000002	フィルターでは、FSCTL_オフロード_書き込みがサポートされています

フィルターオプトインモデルを有効または無効にするには、
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\FileSystem\Filtersupported機能モードレジストリに存在する値に基づいて有効または無効にします。キー。次の値が含まれます。

FILTERSUPPORTEDIIIII	II
0 (既定値)	通常のオプトイン処理を実行します。
1	オプトインしない (すべてのフィルターがアタッチされている場合に SupportedFeatures を0に設定することに相当)

調べる

スタックのサポートされている機能を確認するには、fltmc ユーティリティ内に更新されたコマンドがあります。Fltmc instances – v [volume]: 管理者特権を持つユーザーとして実行し、SprtFtrs列を確認します。フィルターのSprtFtrs値が0に設定されている場合は、フィルターがこのボリュームのオフロードをブロックしていることを意味します。SprtFtrsフィールドが3に設定されている場合は、オフロード操作の両方がサポートされます。

IRP 処理で機能のサポートを確認しています

IRP 処理の一部として、Fsrtlgetsupportedfeaturesルーチンは、特定のボリュームスタックにアタッチされているすべてのフィルターについて、集計されたsupportedfeaturesの状態を取得します。I/O マネージャーや SRV (SMB) などのコンポーネントは、このルーチンを呼び出して、スタック上のすべてのフィルターについてSupportedfeaturesの状態を検証します。独自のオフロードIrpをロールするコンポーネントは、その操作のオプトインサポートを検証するために、この関数を呼び出す必要があります。

フィルタードライバーに関する考慮事項

オフロードデータ転送は、データセンター内でデータを移動する新しい方法です。コアコピーエンジンにはオフロードロジックが統合されているため、既定では、多くのアプリケーションは、明示的にオプトインせずにオフロードデータ移動を実行できます。そのため、フィルター開発者は、これらの新しい操作がフィルターに与える影響を理解する必要があります。これらの操作を十分に理解していない場合、または新しいデータフローを評価しない場合、データが不整合になったり破損したりする可能性があります。次の一覧は、フィルター開発者がオフロードでメモする一連のアクション項目をまとめたものです。

- 新しいデータフロー、フィルターへの影響、およびこれらのオフロード操作をサポートするフィルターの機能について理解します。
- フィルターインストーラーを更新して、Supportedfeaturesの REG_DWORD 値を HKLM\システム \CurrentControlSet\Services\[filter] サブキーに追加します。これを初期化してオフロード機能を指定します。
- オフロード操作に対処するフィルターについては、登録をIRP_MJ_ファイル_システム_制御に更新して、FSCTL_オフロード_読み取りとFSCTL_オフロードを処理_書き込み。
- オフロード操作をブロックする必要があるフィルターの場合は、ステータスコードの状態を返します。フィルター内からサポートされていない状態_サポートされていません。エンドユーザーが変更できるように、ブロッキングオフロード操作を強制するためにレジストリ値に依存しないでください。フィルターでは、オフロード操作を明示的に許可または禁止する必要があります。

トークンのコピー

オフロード操作では、ファイルデータは i/o スタックに表示されません。代わりに、データの論理プロキシである512バイトのトークンとして表示されます。このトークンは、ストレージサブシステムによって生成されるベンダー固有の形式の非透過的で一意の文字列であるか、またはデータのパターンを表す既知の型 (論理的にはゼロに相当するデータ範囲) であることができます。トークンがプロキシになっているデータを変更すると、トークンが無効になるか、またはストレージサブシステムがベンダー固有の手段 (スナップショットメカニズムなど) によって元のデータを保持するようになります。ファイル内の指定された範囲に対するその後のオフロード要求では、一意のトークンが生成されます。

明確に定義されたデータのパターンを表すトークンのクラスがあります。よく知られている最も一般的なトークンは、ゼロに相当するゼロトークンです。トークンが既知のトークンとして定義されている場合、ストレージ_オフロード_トークン構造内のTokenTypeメンバーは、_既知_既知の種類__オフロードトークンに設定されます。このフィールドが設定されている場合、WellKnownPatternメンバーによって、トークンが持つデータのパターンが決まります。

- WellKnownPatternフィールドが 記憶域_オフロード_パターン_0 またはストレージ_オフロード_ゼロ_保護_情報に設定されている場合は、ゼロトークンを示します。このトークンがFSCTL_オフロード_読み取り操作によって返された場合は、目的のファイル範囲に含まれるデータが論理的に0と等しいことを示します。このトークンがFSCTL_オフロード_書き込み操作に提供された場合、書き込まれるファイルの目的の範囲を論理的にゼロにする必要があることを示します。
- ゼロトークン以外に、現在定義されている他の既知のトークンパターンはありません。ユーザーが独自の既知のトークンパターンを定義することは推奨されません。

示さ

Windows が通信する基になる記憶域サブシステムでは、オフロード操作で必要だったデータを処理することができません。これを切り捨てと呼びます。オフロード読み取りでは、返されたトークンが、要求されたデータの範囲を下回ることを意味します。これは、FSCTL_オフロード_読み取り_出力構造のtransferlengthメンバーによって示されます。これは、読み取るファイルの範囲の先頭からのバイト数です。オフロード書き込みの場合、切り捨ては、意図したよりも少ないデータが書き込まれたことを示します。この値は、書き込まれるファイルの範囲の先頭からのバイト数である、FSCTL_OFFLOAD_の書き込み_出力構造体によって示されます。コマンド処理のエラー、または大きな範囲のスタック内の制限により、切り捨てが発生します。

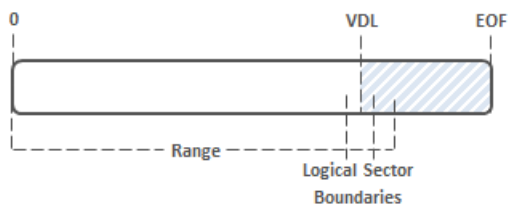
次の2つのシナリオでは、読み取りまたは書き込みのオフロードする範囲が NTFS によって切り捨てられます。

1. VDL がファイルの末尾 (EOF) より前にある場合、コピー範囲は有効なデータ長 (VDL) に切り捨てられます。これは、VDL が論理セクターの境界に合わせて調整されていることを前提としています。



FSCTL_オフロード_読み取り操作の実行中は、フラグオフロード_フラグの読み取りフラグ_現在の_の範囲を超える_0_すべての_は、FSCTL_オフロードで設定され_ファイルの残りの部分にゼロが含まれており、transferlengthメンバーが VDL に切り捨てられていることを示す読み取り_出力構造体。

2. シナリオ1に似ていますが、VDL が論理セクターの境界に揃っていない場合、必要な範囲は NTFS によって次の論理セクターの境界に切り捨てられます。



制限事項

- オフロード操作は、NTFS ボリュームでのみサポートされます。
- リモート共有が NTFS ボリュームであり、サーバーで Windows Server 2012 が実行されている場合は、リモートファイルサーバーを使用してオフロード操作がサポートされます (リモートスタックでオフロード操作もサポートされている場合)。
- NTFS は、Bitlocker または NTFS 暗号化 (EFS) で暗号化されたファイル、重複しないファイル、圧縮ファイル、常駐ファイル、スパーファイル、または TxF トランザクションに参加しているファイルに対して実行されるオフロード FSCTLs をサポートしていません。
- NTFS では、volsnap スナップショット内のファイルに対して実行されるオフロード FSCTLs はサポートされていません。
- 目的のファイル範囲がソースデバイスの論理セクターサイズに固定されていない場合、または目的のファイル範囲がターゲットデ

バイスの論理セクターサイズに固定されていない場合、NTFS はオフロード FSCTL を失敗させます。これは、キャッシュされていない IO と同じセマンティクスに従います。

- **FSCTL_オフロード_書き込み**の前に、コピー先のファイルが事前に割り当てられている (setendoffileとsetallocationではない) 必要があります。
- 負荷分散の読み取りとオフロードの書き込みでは、NTFS はまず**CcCoherencyFlushAndPurgeCache**を呼び出して、変更されたデータをシステムキャッシュにコミットします。これは、非キャッシュ IO と同じセマンティクスです。

ディスク イメージの正しい場所を見つける

2020/04/24 • [Edit Online](#)

ディスクのイメージングとボリュームの管理ツールは、Volsnap によって作成されたスナップショットを使用する場合は、ディスク上のイメージのデータ領域とユーザー データへの影響を配置する場所の対応である必要があります。スナップショットにはディスク イメージが正しく配置されているときに破損したデータが含まれます。

イメージング製品の一部では、ディスク上の BIOS パラメーター ブロック (BPB) は、実際には、ディスクの最初の 8 K バイトに拡張は常と 4 K バイトのみを間違った想定を確認します。その結果、クラスター ボリュームのサイズに応じてイメージング ツール、BPB の最後の 4 K バイト部分内のユーザー データを検索する場合しまうデータが失われる Volsnap スナップショットがボリュームの作成後。これは、原因という事実にイメージング ツールは、ユーザー データをディスクの予約領域に配置されます。

8 K バイトでは常に、論理セクター サイズが 512 バイトまたは 4 K (バイト単位) など、デバイスの設定は、BPB のサイズを特定できません。

次の表には、セクター レイアウトとバイトオフセットの要件、データ領域のディスク イメージがについて説明します。

BCB 予約領域イメージ データ領域 (512 バイト セクター) 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 + 0 (バイトオフセット)
4096 8192

重複するボリューム名がある場合のボリュームの 列挙について

2020/04/24 • [Edit Online](#)

ボリュームを列挙するときに、結果のボリューム情報の一覧に重複するボリューム名が表示される可能性があります。

これが発生する理由を理解するために、次のシナリオを考えてみます。ボリューム列挙ルーチ

ン [FltEnumerateVolumeInformation](#) は、すべてのシステムボリュームを列挙するために使用されます。これにより、フィルターマネージャーが認識しているボリュームごとに1つずつ、ボリューム情報構造を格納したバッファーが生成されます。このバッファーでは、各ボリューム情報構造は、[フィルター_ボリューム_基本的な_情報](#) または [フィルター_ボリューム_標準_情報](#) ですが、両方を指定することはできません。

このボリューム情報構造の一覧を指定すると、複数のリスト要素に同じボリューム名を含めることができます。つまり、2つ以上のリスト要素の `filtervolumename` メンバーが同一である可能性があります。これが可能なのは、[FltEnumerateVolumes](#) などのすべてのフィルターマネージャー列挙ルーチンが、マウント解除されていても破棄されていないボリューム (開いているファイルがまだボリューム上に存在しているため) を含むボリュームを列挙するためです。このため、ボリュームのマウントが解除されると、ボリューム情報リスト (現在のマウントされた状態の場合は1回、前のマウント解除された状態の場合は1回、最も単純な場合は非破棄状態) で、ボリュームの名前が複数回表示されることがあります。

ボリューム情報の一覧に重複するボリューム名が表示される場合は、同じ名前の各グループが前述の説明によって説明されています。ただし、上記のシナリオを確認するには、次の手順を実行します。

- フィルター_ボリューム_標準_情報の構造を使用してリストに値が設定されている場合は、`filtervolumename` メンバーが等しい構造体のグループを識別します。このグループ内の1つまたは複数の構造体が、`フラグメンバー` で設定されている_ボリュームフラグを_している場合は、そのグループに関連付けられているボリュームがマウント解除されていますが、非破棄状態になっています。これにより、重複するボリューム名が存在することが確認できます。該当する場合は、残りのすべてのグループに対してこの手順を繰り返します。
- リストに `type of FILTER_VOLUME_BASIC_INFORMATION` の構造が設定されている場合は、この一覧を同等のフィルター_ボリューム_標準_情報構造体の形式に変換して、前の例のように処理します。プレットポイント。

フィルター_ボリューム_標準_情報の構造は、Windows Vista 以降でのみ使用でき ことに注意してください。

このトピックの影響を受けるルーチンと構造は次のとおりです。

[_ボリューム_基本的な_情報にフィルターを適用する](#)

[_ボリューム_標準_情報をフィルター処理する](#)

[FilterVolumeFindFirst](#)

[FilterVolumeFindNext](#)

[FltEnumerateVolumeInformation](#)

[FltEnumerateVolumes](#)

[FltGetVolumeInformation](#)

その他の参照情報

2020/04/24 • [Edit Online](#)

ファイルシステム関連の開発に関連するほとんどの参照ページは、『[Windows Driver Kit \(WDK\) 用プログラミングリファレンス](#)』の「[ファイルシステム](#)」にあります。

このセクションには、追加の参照ページが含まれています。最終的には、上記のリンクにある [ファイルシステム] セクションの下に移行されます。

IRP_MJ_ACQUIRE_FOR_MOD_WRITE 共用体の FLT_PARAMETERS

2020/08/04 • [Edit Online](#)

操作の [FLT_IO_PARAMETER_BLOCK](#) 構造体の `MajorFunction` フィールドが `IRP_MJ_ACQUIRE_FOR_MOD_WRITE` 場合は、次の共用体コンポーネントが使用されます。

構文

```
typedef union _FLT_PARAMETERS {
    ... ;
    struct {
        PLARGE_INTEGER EndingOffset;
        PERESOURCE *ResourceToRelease;
    } AcquireForModifiedPageWriter;
    ... ;
} FLT_PARAMETERS, *PFLT_PARAMETERS;
```

メンバー

`AcquireForModifiedPageWriter`

次のメンバーを含む構造体。

`EndingOffset`

書き込まれた最後のバイトのオフセットに1を加えた値を格納する変数へのポインター。

`ResourceToRelease`

取得する [リソース \(リソース\)](#) へのポインターへのポインター。

注釈

`IRP_MJ_ACQUIRE_FOR_MOD_WRITE` 操作の [FLT_PARAMETERS](#) 構造体には、コールバックデータ ([FLT_CALLBACK_DATA](#)) 構造体によって表される `AcquireForModifiedPageWriter` 操作のパラメーターが含まれています。これは [FLT_IO_PARAMETER_BLOCK](#) 構造体に含まれています。

`IRP_MJ_ACQUIRE_FOR_MOD_WRITE` は、ファイルシステム (FSFilter) コールバック操作です。この操作では、*ResourceToRelease* は取得 (操作前) または取得された (操作後の) リソースへのポインターへのポインターです。リソースは `IRP_MJ_RELEASE_FOR_MOD_WRITE` コールバック操作で解放されます。

FSFilter のコールバック操作の詳細については、[Fsrtlregisterfilesystemfiltercallbacks](#) のリファレンスエントリを参照してください。

要件

ヘッダー: *fltkernel.H* (*fltkernel.h* を含む)

関連項目

[FLT_CALLBACK_DATA](#)

FLT_IO_PARAMETER_BLOCK

FLT_IS_FASTIO_OPERATION

FLT_IS_FS_FILTER_OPERATION

FLT_IS_IRP_OPERATION

FLT_PARAMETERS

FsRtlRegisterFileSystemFilterCallbacks バック

IRP_MJ_ACQUIRE_FOR_SECTION_SYNCHRONIZATION 共用体の FLT_PARAMETERS

2020/08/04 • [Edit Online](#)

操作の [FLT_IO_PARAMETER_BLOCK](#) 構造体の MajorFunction フィールドが
IRP_MJ_ACQUIRE_FOR_SECTION_SYNCHRONIZATION 場合は、次の共用体コンポーネントが使用されます。

構文

```
typedef union _FLT_PARAMETERS {  
    ...    ;  
    struct {  
        FS_FILTER_SECTION_SYNC_TYPE SyncType;  
        ULONG POINTER_ALIGNMENT PageProtection;  
        PFS_FILTER_SECTION_SYNC_OUTPUT OutputInformation;  
    } AcquireForSectionSynchronization;  
    ...    ;  
} FLT_PARAMETERS, *PFLT_PARAMETERS;
```

メンバー

SyncType

セクションに要求された同期の種類。セクションが作成されている場合、このパラメーターは `SyncTypeCreateSection` に設定されます。それ以外の場合は、`SyncTypeOther` に設定されます。

PageProtection

セクションに要求されたページ保護の種類。`SyncType` が `SyncTypeOther` の場合、は0である必要があります。それ以外の場合、このパラメーターは、定義されている [メモリ保護定数値](#) のいずれかである必要があります。

OutputInformation

作成されるセクションの属性を記述する情報を指定する [FS_FILTER_SECTION_SYNC_OUTPUT](#) 構造体。

注釈

IRP_MJ_ACQUIRE_FOR_SECTION_SYNCHRONIZATION 操作の [FLT_PARAMETERS](#) 構造体には、コールバックデータ ([FLT_CALLBACK_DATA](#)) 構造体によって表される `AcquireForSectionSynchronization` 操作のパラメーターが含まれています。これは `FLT_IO_PARAMETER_BLOCK` 構造体に含まれています。

IRP_MJ_ACQUIRE_FOR_SECTION_SYNCHRONIZATION は、ファイルシステム (FSFilter) コールバック操作です。

`SyncType` メンバーの列挙値が `SyncTypeOther` に設定されている場合、ファイルシステムミニフィルタードライバーまたはレガシフィルタードライバーがこの操作を失敗させることはできません。`SyncType` が `SyncTypeCreateSection` に設定されている場合は、セクションを作成するのに十分なメモリがない場合、ファイルシステムミニフィルタードライバーまたはレガシフィルタードライバーが失敗し、`STATUS_INSUFFICIENT_RESOURCES` エラーが発生します。

FSFilter のコールバック操作の詳細については、[Fsrtlregisterfilesystemfiltercallbacks](#) のリファレンスエントリを参照してください。

要件

バージョン: windows XP 以降のバージョンの windows オペレーティングシステムで使用できます。

ヘッダー: fltkernel .H (fltkernel .h を含む)

関連項目

[FLT_CALLBACK_DATA](#)

[FLT_IO_PARAMETER_BLOCK](#)

[FLT_PARAMETERS](#)

[FsRtlRegisterFileSystemFilterCallbacks](#) バック

IRP_MJ_CREATE 共用体の FLT_PARAMETERS

2020/08/04 • [Edit Online](#)

操作の [FLT_IO_PARAMETER_BLOCK](#) 構造体の `MajorFunction` フィールドが [IRP_MJ_CREATE](#) 場合は、次の共用体コンポーネントが使用されます。

構文

```
typedef union _FLT_PARAMETERS {
    ... ;
    struct {
        PIO_SECURITY_CONTEXT    SecurityContext;
        ULONG                   Options;
        USHORT POINTER_ALIGNMENT FileAttributes;
        USHORT                  ShareAccess;
        USHORT POINTER_ALIGNMENT EaLength;
        PVOID                   EaBuffer;
        LARGE_INTEGER           AllocationSize;
    } Create;
    ... ;
} FLT_PARAMETERS, *PFLT_PARAMETERS;
```

メンバー

`FLT_PARAMETERS` の `Create` 構造には、次のメンバーが含まれます。

SecurityContext

`IRP_MJ_CREATE` 要求のセキュリティコンテキストを表す [IO_SECURITY_CONTEXT](#) 構造体へのポインター。この場合、次のようになります。

- > `SecurityContext AccessState` は、オブジェクトのサブジェクトコンテキスト、アクセスの種類、およびその他の必要なアクセスの種類を含む [ACCESS_STATE](#) 構造体へのポインターです。
- > `SecurityContext DesiredAccess` は、ファイルに要求されたアクセス権を指定する [ACCESS_MASK](#) 構造です。詳細については、[Fltcreatefile](#) の `DesiredAccess` パラメーターを参照してください。

[オプション]

ファイルを作成または開くときに適用するオプション、およびファイルが既に存在する場合に実行するアクションを指定するフラグのビットマスク。このメンバーの下位24ビットは、[Fltcreatefile](#) の `createoptions` パラメーターに対応しています。上位8ビットは [Fltcreatefile](#) の `CreateDisposition` パラメーターに対応します。

FileAttributes

ファイルを作成または開くときに適用される属性のビットマスク。詳細については、[Fltcreatefile](#) の `fileattributes` パラメーターを参照してください。

ShareAccess

ファイルに要求された共有アクセス権のビットマスク。このパラメーターがゼロの場合は、排他アクセスが要求されます。詳細については、[Fltcreatefile](#) の `"/` アクセスパラメーターを参照してください。

EaLength

`EaBuffer` メンバーが指すバッファの長さ (バイト単位)。詳細については、[Fltcreatefile](#) の `EaLength` パラメーターを参照してください。

EaBuffer

ファイルに適用される拡張属性 (EA) 情報を格納する、呼び出し元が提供する、[FILE_FULL_EA_INFORMATION](#) 構造化されたバッファーへのポインター。詳細については、[Ftcreatefile](#) の *EaBuffer* パラメーターを参照してください。

AllocationSize

必要に応じて、ファイルの初期割り当てサイズをバイト単位で指定します。0 以外の値は、ファイルが作成、上書き、または置き換えられない限り、無効です。詳細については、[Ftcreatefile](#) の割り当て *サイズ* パラメーターを参照してください。

注釈

[IRP_MJ_CREATE](#) 操作の [FLT_PARAMETERS](#) 構造体には、コールバックデータ ([FLT_CALLBACK_DATA](#)) 構造体によって表される IRP ベースの作成操作のパラメーターが含まれています。これは [FLT_IO_PARAMETER_BLOCK](#) 構造体に含まれています。

IRP_MJ_CREATE は、IRP ベースの操作です。

要件

|Header |*fltkernel.h* (*fltkernel.h* を含む)

関連項目

[ACCESS_MASK](#)

[ACCESS_STATE](#)

[FILE_FULL_EA_INFORMATION](#)

[FLT_CALLBACK_DATA](#)

[FLT_IO_PARAMETER_BLOCK](#)

[FLT_IS_FASTIO_OPERATION](#)

[FLT_PARAMETERS](#)

[FtCreateFile](#)

[IRP_MJ_CREATE](#)

IRP_MJ_CREATE_MAILSLLOT 共用体の FLT_PARAMETERS

2020/08/04 • [Edit Online](#)

FLT_PARAMETERS union 内の次の構造は、FLT_IO_PARAMETER_BLOCK構造体のMajorFunctionフィールドがIRP_MJ_CREATE_MAILSLLOT場合に使用されます。

構文

```
typedef union _FLT_PARAMETERS {
    ... ;
    struct {
        PIO_SECURITY_CONTEXT    SecurityContext;
        ULONG                   Options;
        USHORT POINTER_ALIGNMENT Reserved;
        USHORT                   ShareAccess;
        PVOID                   Parameters;
    } CreateMailslot;
    ... ;
} FLT_PARAMETERS, *PFLT_PARAMETERS;
```

メンバー

FLT_PARAMETERS のCreatemailslot構造には、次のメンバーが含まれています。

SecurityContext

IRP_MJ_CREATE_MAILSLLOT 要求のセキュリティコンテキストを表すIO_SECURITY_CONTEXT構造体へのポインター。この場合、次のようになります。

- >SecurityContext AccessStateは、オブジェクトのサブジェクトコンテキスト、アクセスの種類、およびその他の必要なアクセスの種類を含むACCESS_STATE構造体へのポインターです。
- >SecurityContext DesiredAccessは、メールスロットに要求されたアクセス権を指定するACCESS_MASK構造です。詳細については、FltCreateMailslotFileのDesiredAccessパラメーターを参照してください。

[オプション]

メールスロットを作成または開くときに適用されるオプションを指定するフラグのビットマスク。また、メールスロットが既に存在する場合に実行されるアクションも指定します。このメンバーの下位24ビットは、FltCreateMailslotFileのcreateoptionsパラメーターに対応しています。上位8ビットは、FltCreateMailslotFileのCreateDispositionパラメーターに対応します。

予約されています。

確保使用しないでください。

ShareAccess

メールスロットファイルに要求された共有アクセス権のビットマスク。このパラメーターがゼロの場合は、排他アクセスが要求されます。詳細については、FltCreateMailslotFileへの"/" アクセスパラメーターの説明を参照してください。

パラメーター

作成または開いているメールスロットに関する情報を格納しているMAILSLLOT_CREATE_PARAMETERS構造体へのポインター。

注釈

I/O 操作が IRP_MJ_CREATE_MAILSLOT 場合、[FLT_PARAMETERS](#)には`createmailslot`構造体が含まれます。I/O 操作は[FLT_CALLBACK_DATA](#)構造体によって表されます。操作パラメーターは、コールバックデータの`IoPb`パラメーターが指す[FLT_IO_PARAMETER_BLOCK](#)構造内に含まれます。

IRP_MJ_CREATE_MAILSLOT 操作のコールバックルーチンが登録されているファイルシステムミニフィルタードライバーは、必要な処理を実行して返す必要があります。

最後の `longword` フィールド以外に、`Createmailslot`構造体内のフィールドが`Create structure`のフィールドと一致する必要があります。

IRP_MJ_CREATE_MAILSLOT は、IRP ベースの操作です。

要件

ヘッダー: `fltkernel.h` (`fltkernel.h` を含む)

関連項目

[ACCESS_MASK](#)

[ACCESS_STATE](#)

[FLT_CALLBACK_DATA](#)

[FLT_IO_PARAMETER_BLOCK](#)

[FLT_PARAMETERS](#)

[FltCreateMailslotFile](#)

[IRP_MJ_CREATE_MAILSLOT](#)

[MAILSLOT_CREATE_PARAMETERS](#)

IRP_MJ_CREATE_NAMED_PIPE 共用体の FLT_PARAMETERS

2020/08/04 • [Edit Online](#)

FLT_PARAMETERS union 内の次の構造は、FLT_IO_PARAMETER_BLOCK構造体のMajorFunctionフィールドがIRP_MJ_CREATE_NAMED_PIPE場合に使用されます。

構文

```
typedef union _FLT_PARAMETERS {
    ... ;
    struct {
        PIO_SECURITY_CONTEXT    SecurityContext;
        ULONG                   Options;
        USHORT POINTER_ALIGNMENT Reserved;
        USHORT                   ShareAccess;
        PVOID                    Parameters;
    } CreatePipe;
    ... ;
} FLT_PARAMETERS, *PFLT_PARAMETERS;
```

メンバー

FLT_PARAMETERS のCreatepipe構造体には、次のメンバーが含まれています。

SecurityContext

IRP_MJ_CREATE_NAMED_PIPE 要求のセキュリティコンテキストを表すIO_SECURITY_CONTEXT構造体へのポインター。この場合、次のようになります。

- >SecurityContext AccessStateは、オブジェクトのサブジェクトコンテキスト、アクセスの種類、およびその他の必要なアクセスの種類を含むACCESS_STATE構造体へのポインターです。
- >SecurityContext DesiredAccessは、名前付きパイプに対して要求されるアクセス権を指定するACCESS_MASK構造です。詳細については、FltCreateNamedPipeFileのDesiredAccess/パラメーターを参照してください。

[オプション]

名前付きパイプを作成または開くときに適用されるオプションを指定するフラグのビットマスク。パイプが既に存在する場合に実行されるアクションも指定します。このメンバーの下位24ビットは、FltCreateNamedPipeFileのcreateoptions/パラメーターに対応しています。上位8ビットは、FltCreateNamedPipeFileのCreateDisposition/パラメーターに対応します。

予約されています。

確保使用しないでください。

ShareAccess

名前付きパイプファイルに要求された共有アクセス権のビットマスク。このパラメーターがゼロの場合は、排他アクセスが要求されます。詳細については、FltCreateNamedPipeFileへの"/" アクセスパラメーターの説明を参照してください。

パラメーター

作成または開いている名前付きパイプに関する情報を格納しているNAMED_PIPE_CREATE_PARAMETERS構造体へのポインター。

注釈

I/O 操作が IRP_MJ_CREATE_NAMED_PIPE 場合、[FLT_PARAMETERS](#)にはcreatepipe構造体が含まれます。I/O 操作は[FLT_CALLBACK_DATA](#)構造体によって表されます。操作/パラメーターは、コールバックデータの*IoPb*パラメーターが指す[FLT_IO_PARAMETER_BLOCK](#)構造内に含まれます。

IRP_MJ_CREATE_NAMED_PIPE 操作のコールバックルーチンが登録されているファイルシステムミニフィルタードライバーは、必要な処理を実行して返す必要があります。

最後の longword フィールド以外は、**Createpipe**構造体のフィールドがCreate structure のフィールドと一致している必要があることに注意してください。

IRP_MJ_CREATE_NAMED_PIPE は、IRP ベースの操作です。

要件

ヘッダー: fltkernel .H (fltkernel .h を含む)

関連項目

[ACCESS_MASK](#)

[ACCESS_STATE](#)

[FLT_CALLBACK_DATA](#)

[FLT_IO_PARAMETER_BLOCK](#)

[FLT_PARAMETERS](#)

[FltCreateNamedPipeFile](#)

[IRP_MJ_CREATE_NAMED_PIPE](#)

[NAMED_PIPE_CREATE_PARAMETERS](#)

_IRP_MJ_デバイス_コントロールおよび IRP_MJ_内部_デバイス_コントロール共用体

2020/04/24 • [Edit Online](#)

FLT_IO_パラメーターのMajorFunctionフィールド_操作のブロック構造体がirp_MJ_デバイス_CONTROLまたはirp_MJ_の場合に使用される共用体コンポーネント_内部_デバイス_制御。

構文

```
typedef union _FLT_PARAMETERS {
    ... ;
    union {
        struct {
            ULONG                OutputBufferLength;
            ULONG POINTER_ALIGNMENT InputBufferLength;
            ULONG POINTER_ALIGNMENT IoControlCode;
        } Common;
        struct {
            ULONG                OutputBufferLength;
            ULONG POINTER_ALIGNMENT InputBufferLength;
            ULONG POINTER_ALIGNMENT IoControlCode;
            PVOID                InputBuffer;
            PVOID                OutputBuffer;
            PMDL                 OutputMdlAddress;
        } Neither;
        struct {
            ULONG                OutputBufferLength;
            ULONG POINTER_ALIGNMENT InputBufferLength;
            ULONG POINTER_ALIGNMENT IoControlCode;
            PVOID                SystemBuffer;
        } Buffered;
        struct {
            ULONG                OutputBufferLength;
            ULONG POINTER_ALIGNMENT InputBufferLength;
            ULONG POINTER_ALIGNMENT IoControlCode;
            PVOID                InputSystemBuffer;
            PVOID                OutputBuffer;
            PMDL                 OutputMdlAddress;
        } Direct;
        struct {
            ULONG                OutputBufferLength;
            ULONG POINTER_ALIGNMENT InputBufferLength;
            ULONG POINTER_ALIGNMENT IoControlCode;
            PVOID                InputBuffer;
            PVOID                OutputBuffer;
        } FastIo;
    } DeviceIoControl;
    ... ;
} FLT_PARAMETERS, *PFLT_PARAMETERS;
```

Members

DeviceIoControl

的

すべてのバッファリングメソッドに使用される共用体コンポーネント。

OutputBufferLength

出力バッファの長さ (バイト単位)。outputbuffer、DirectBuffer、またはfadirectoryのメンバーはを指します。

InputBufferLength

InputBuffer、バッファに格納されていない、InputBuffer、またはfaのメンバーが指すバッファの長さ (バイト単位)。

IoControlCode

ターゲットデバイスのデバイスドライバに渡される IOCTL 関数コード。

IOCTL 要求の詳細については、「カーネルモードアーキテクチャガイド」の「[i/o 制御コードの使用](#)」および Microsoft Windows SDK のドキュメントの「デバイスの入力と出力の制御コード」を参照してください。(このリソースは、一部の言語および国では使用できません。)

両者

バッファリングメソッドがメソッド_場合に使用される共用体コンポーネントではありません。バッファリングメソッドの詳細については、「カーネルモードアーキテクチャガイド」の「[i/o 制御コードの定義](#)」を参照してください。

InputBuffer

操作の元の要求元が指定した入力バッファのユーザーモード仮想アドレス。I/O マネージャーとフィルタマネージャーでは、これらのアドレスは検証されません。ユーザー領域のアドレスが有効であることを確認するには、[ProbeForRead](#)、[ProbeForWrite](#)、[FltLockUserBuffer](#)などのルーチンを使用して、try/exceptブロック内のすべてのバッファ参照を含むようにする必要があります。詳細については、「カーネルモードアーキテクチャガイド」の「[バッファを使用したユーザー領域のアドレスの参照](#)」を参照してください。(このリソースは、一部の言語および国では使用できません。)

OutputBuffer

操作の元の要求元が指定した出力バッファのユーザーモード仮想アドレス。I/O マネージャーとフィルタマネージャーでは、これらのアドレスは検証されません。ユーザー領域のアドレスが有効であることを確認するには、[ProbeForRead](#)、[ProbeForWrite](#)、[FltLockUserBuffer](#)などのルーチンを使用して、try/exceptブロック内のすべてのバッファ参照を含むようにする必要があります。詳細については、「カーネルモードアーキテクチャガイド」の「[バッファを使用したユーザー領域のアドレスの参照](#)」を参照してください。

OutputMdlAddress

Outputbufferメンバーが指しているバッファを記述するメモリ記述子リスト (MDL) のアドレス。このメンバーは省略可能であり、NULLにすることができます。

付き

バッファリングメソッドがメソッド_バッファリングされるときに使用される共用体コンポーネント。バッファリングメソッドの詳細については、「カーネルモードアーキテクチャガイド」の「[i/o 制御コードの定義](#)」を参照してください。(このリソースは、一部の言語および国では使用できません。)

SystemBuffer

操作に対してシステムが割り当てたバッファのアドレス。メソッド_バッファ i/o では、このバッファは入力と出力の両方に使用されます。詳細については、「カーネルモードアーキテクチャガイド」の「[データバッファにアクセスするためのメソッド](#)」を参照してください。(このリソースは、一部の言語および国では使用できません。)

接続

バッファリングメソッドが_ダイレクトまたはメソッドで_メソッドである場合に使用される共用体コンポーネントは_ダイレクト_OUT ます。バッファリングメソッドの詳細については、「カーネルモードアーキテクチャガイド」の「[i/o 制御コードの定義](#)」を参照してください。(このリソースは、一部の言語および国では使用できません。)

InputSystemBuffer

操作の入力バッファのアドレス。このバッファは、カーネルモードから安全にアクセスできるように、オペレーティングシステムによってロックされています。詳細については、「カーネルモードアーキテクチャガイド」の「[データバッファにアクセスするためのメソッド](#)」を参照してください。(このリソースは、一部の言語および国では使用できません。)

OutputBuffer

操作の元の要求元が指定した出力バッファのユーザーモード仮想アドレス。ダイレクト i/o では、i/o 操作の元の要求元と同じブ

ロセスコンテキストでミニフィルターが使用されている限り、オペレーティングシステムは i/o ではない_方法で、このバッファーをロックダウンして、カーネルモードから安全にアクセスできるようにします。(それ以外の場合は、[MmGetSystemAddressForMdlSafe](#)を呼び出して、[Outputmdladdress](#)メンバーが指すメモリ記述子リスト (MDL) からシステムアドレスを取得する必要があります)。
詳細については、「[カーネルモードアーキテクチャガイド](#)」の「[direct I/O とエラーをダイレクト I/O で使用する](#)」を参照してください。(このリソースは、一部の言語および国では使用できません。)

OutputMdlAddress

直接の [outputbuffer](#)メンバーが指すバッファーを記述する MDL のアドレス。このメンバーは必須であり、NULLにすることはできません。

高速

[FLT_CALLBACK_データ](#)構造体が高速 i/o [IRP_MJ_デバイス_制御](#)操作を表す場合に使用される共用体コンポーネント。

InputBuffer

操作の元の要求元が指定した入力バッファーのユーザーモード仮想アドレス。I/O マネージャーとフィルターマネージャーでは、これらのアドレスは検証されません。ユーザー領域のアドレスが有効であることを確認するには、[ProbeForRead](#)、[ProbeForWrite](#)、[FltLockUserBuffer](#)などのルーチンを使用して、try/exceptブロック内のすべてのバッファー参照を含むようにする必要があります。詳細については、「[カーネルモードアーキテクチャガイド](#)」の「[ユーザー空間の参照](#)」の[エラー](#)を参照してください。(このリソースは、一部の言語および国では使用できません。)

OutputBuffer

操作の元の要求元が指定した出力バッファーのユーザーモード仮想アドレス。I/O マネージャーとフィルターマネージャーでは、これらのアドレスは検証されません。ユーザー領域のアドレスが有効であることを確認するには、[ProbeForRead](#)、[ProbeForWrite](#)、[FltLockUserBuffer](#)などのルーチンを使用して、try/exceptブロック内のすべてのバッファー参照を含むようにする必要があります。詳細については、「[カーネルモードアーキテクチャガイド](#)」の「[ユーザー空間の参照](#)」の[エラー](#)を参照してください。(このリソースは、一部の言語および国では使用できません。)

注釈

[Irp_MJ_デバイス_コントロール](#)と[Irp_MJ_内部_デバイス](#)の[FLT_parameters](#)構造体には、irp ベースのパラメーターが含まれています。コールバックデータ ([FLT_callback_data](#)) 構造によって表されるデバイス-i/o 制御情報操作。これは、[FLT_IO_パラメーター_ブロック](#)構造体に含まれています。

[IRP_MJ_デバイス_制御](#)は、IRP ベースの操作または高速 i/o 操作にすることができます。

[IRP_MJ_内部_デバイス_制御](#)は、IRP ベースの i/o 操作です。

要件

Header	Fltkernl.h (Fltkernl.h を含む)

関連項目

[FLT_コールバック_データ](#)

[FLT_IO_パラメーター_ブロック](#)

[FLTは_高速な操作_](#)

[FLTは_FS_フィルターの_操作です。](#)

[FLTは_IRP_操作です](#)

[FLT_パラメーター](#)

[FltDeviceIoControlFile](#)

FltLockUserBuffer

IoBuildDeviceIoControlRequest

IRP_MJ_デバイス_コントロール

IRP_MJ_内部_デバイス_コントロール

MmGetSystemAddressForMdlSafe

MmProbeAndLockPages

ProbeForRead

ProbeForWrite

ZwDeviceIoControlFile

IRP_MJ_DIRECTORY_CONTROL 共用体の FLT_PARAMETERS

2020/08/04 • [Edit Online](#)

操作の [FLT_IO_PARAMETER_BLOCK](#) 構造体の MajorFunction フィールドが [IRP_MJ_DIRECTORY_CONTROL](#) 場合に使用される共用体コンポーネント。

構文

```
typedef union _FLT_PARAMETERS {
    ... ;
    union {
        struct {
            ULONG                Length;
            PUNICODE_STRING      FileName;
            FILE_INFORMATION_CLASS FileInformationClass;
            ULONG_PTR            FileIndex;
            PVOID                DirectoryBuffer;
            PMDL                 MdlAddress;
        } QueryDirectory;
        struct {
            ULONG                Length;
            ULONG_PTR            CompletionFilter;
            ULONG                Spare1;
            ULONG_PTR            Spare2;
            PVOID                DirectoryBuffer;
            PMDL                 MdlAddress;
        } NotifyDirectory;
    } DirectoryControl;
    ... ;
} FLT_PARAMETERS, *PFLT_PARAMETERS;
```

メンバー

DirectoryControl

次のメンバーを含む構造体。

QueryDirectory

IRP_MN_QUERY_DIRECTORY 操作に使用される共用体コンポーネントです。

[データ型]

Querydirectory. DirectoryBuffer メンバーが指すバッファの長さ (バイト単位)。

FileName

指定したディレクトリ内のファイルの名前を格納している [UNICODE_STRING](#) 構造体へのポインター。

FileInformationClass

次に示す値のいずれかを指定します。

I	II
ファイルのディレクトリ情報 (& i)	各ファイルの FILE_BOTH_DIR_INFORMATION 構造体を返します。

「	”
FileDirectoryInformation	各ファイルのFILE_DIRECTORY_INFORMATION構造体を返します。
FileFullDirectoryInformation	各ファイルのFILE_FULL_DIR_INFORMATION構造体を返します。
FileIdBothDirectoryInformation	各ファイルのFILE_ID_BOTH_DIR_INFORMATION構造体を返します。
FileIdFullDirectoryInformation	各ファイルのFILE_ID_FULL_DIR_INFORMATION構造体を返します。
FileNamesInformation	各ファイルのFILE_NAMES_INFORMATION構造体を返します。
FileObjectIdInformation	各ファイルのFILE_OBJECTID_INFORMATION構造体を返します。
FileReparsePointInformation	ディレクトリの1つのFILE_REPARSE_POINT_INFORMATION構造体を返します。

FileIndex

ディレクトリスキャンが開始されるファイルのインデックス。SL_INDEX_SPECIFIED フラグが設定されていない場合は無視されます。このパラメーターは、どの Win32 関数またはカーネルモードサポーターンでも指定できません。現時点では、32ビットの NT ベースのオペレーティングシステムのみが存在する NT 仮想 DOS コンピューター (NTVDM) によってのみ使用されています。ファイルインデックスは NTFS などのファイルシステムに対して定義されていないことに注意してください。これは、親ディレクトリ内のファイルの位置が固定されておらず、並べ替え順序を維持するためにいつでも変更できることに注意してください。

DirectoryBuffer

ディレクトリの内容に関する要求された情報を受け取る、呼び出し元から提供される出力バッファーへのポインター。このメンバーは省略可能であり、MDL がQuerydirectory. MdlAddressに指定されている場合は NULL にすることができます。「解説」を参照してください。

MdlAddress

Querydirectory. DirectoryBufferメンバーが指すバッファーを記述するメモリ記述子リスト (MDL) のアドレス。このメンバーは省略可能であり、Querydirectory. directorybufferにバッファーが指定されている場合はNULLにすることができます。「解説」を参照してください。

NotifyDirectory

IRP_MN_NOTIFY_CHANGE_DIRECTORY 操作に使用される共用体コンポーネントです。

[データ型]

Notifydirectory. DirectoryBufferメンバーが指すバッファーの長さ (バイト単位)。

補完フィルター

通知リストの Irp を完了させる必要があるファイルまたはディレクトリに対する変更の種類を指定するフラグのビットマスク。有効なフラグ値については、次に説明します。

'''	”
FILE_NOTIFY_CHANGE_FILE_NAME	このディレクトリでファイルが追加、削除、または名前が変更されました。
FILE_NOTIFY_CHANGE_DIR_NAME	サブディレクトリの作成、削除、または名前の変更が行われました。

III	II
FILE_NOTIFY_CHANGE_NAME	このディレクトリの名前は変更されました。
FILE_NOTIFY_CHANGE_ATTRIBUTES	最終アクセス時刻など、このファイルの属性の値が変更されました。
FILE_NOTIFY_CHANGE_SIZE	このファイルのサイズが変更されました。
FILE_NOTIFY_CHANGE_LAST_WRITE	このファイルの最終更新時刻が変更されました。
FILE_NOTIFY_CHANGE_LAST_ACCESS	このファイルの最終アクセス時刻が変更されました。
FILE_NOTIFY_CHANGE_CREATION	このファイルの作成時刻が変更されました。
FILE_NOTIFY_CHANGE_EA	このファイルの拡張属性は変更されています。
FILE_NOTIFY_CHANGE_SECURITY	このファイルのセキュリティ情報は変更されています。
FILE_NOTIFY_CHANGE_STREAM_NAME	ファイルストリームがこのディレクトリで追加、削除、または名前変更されました。
FILE_NOTIFY_CHANGE_STREAM_SIZE	このファイルストリームのサイズが変更されました。
FILE_NOTIFY_CHANGE_STREAM_WRITE	このファイルストリームのデータは変更されています。

Spare1
現在は使用しません。

Spare2
現在は使用しません。

DirectoryBuffer
ディレクトリの内容に関する要求された情報を受け取る、呼び出し元から提供される出力バッファーへのポインター。このメンバーは省略可能であり、MDL がNotifydirectory. MdlAddressに指定されている場合は NULL にすることができます。「解説」を参照してください。

MdlAddress
Notifydirectory. directorybufferメンバーが指すバッファーを記述する MDL のアドレス。このメンバーは省略可能であり、Notifydirectory. directorybufferにバッファーが指定されている場合はNULLにすることができます。「解説」を参照してください。

注釈

IRP_MJ_DIRECTORY_CONTROL 操作のFLT_PARAMETERS 構造体には、コールバックデータ (FLT_CALLBACK_DATA) 構造体によって表される、IRP ベースのディレクトリ制御情報操作のパラメーターが含まれています。これは FLT_IO_PARAMETER_BLOCK 構造体に含まれています。

Directorybufferとmdladdressの両方のバッファーを指定する場合は、ミニフィルターで MDL を使用することをお勧めします。Directorybufferが指すメモリは、呼び出しプロセスのコンテキスト内でアクセスされているユーザーモードアドレスである場合、またはカーネルモードアドレスの場合に有効です。

ミニフィルターによってmdladdressの値が変更された場合、事後操作コールバックの後、フィルターマネージャーは現在mdladdressに格納されている MDL を解放し、mdladdressの前の値を復元します。

IRP_MJ_DIRECTORY_CONTROL は、IRP ベースの操作です。

要件

ヘッダー: fltkernel .H (fltkernel .h を含む)

関連項目

[FILE_BOTH_DIR_INFORMATION](#)

[FILE_DIRECTORY_INFORMATION](#)

[FILE_FULL_DIR_INFORMATION](#)

[FILE_ID_BOTH_DIR_INFORMATION](#)

[FILE_ID_FULL_DIR_INFORMATION](#)

[FILE_NAMES_INFORMATION](#)

[FILE_OBJECTID_INFORMATION](#)

[FILE_REPARSE_POINT_INFORMATION](#)

[FLT_CALLBACK_DATA](#)

[FLT_IO_PARAMETER_BLOCK](#)

[FLT_IS_FASTIO_OPERATION](#)

[FLT_IS_FS_FILTER_OPERATION](#)

[FLT_IS_IRP_OPERATION](#)

[FLT_PARAMETERS](#)

[FltNotifyFilterChangeDirectory](#)

[FsRtlNotifyFilterChangeDirectory](#)

[FsRtlNotifyFilterReportChange](#)

[FsRtlNotifyFullChangeDirectory](#)

[FsRtlNotifyFullReportChange](#)

[IRP_MJ_DIRECTORY_CONTROL](#)

[ZwQueryDirectoryFile](#)

FLT_の IRP_MJ_高速_IO_IF union が可能な場合_チェック_

2020/04/24 • [Edit Online](#)

次の共用体コンポーネントは、**FLT_IO_パラメーター_のブロック**構造のMajorFunctionフィールドが、操作に対する IRP_MJ_高速_IO_の場合に使用され得る。

構文

```
typedef union _FLT_PARAMETERS {
    ... ;
    struct {
        LARGE_INTEGER      FileOffset;
        ULONG               Length;
        ULONG_PTR_ALIGNMENT LockKey;
        BOOLEAN_PTR_ALIGNMENT CheckForReadOperation;
    } FastIoCheckIfPossible;
    ... ;
} FLT_PARAMETERS, *PFLT_PARAMETERS;
```

Members

可能な場合

次のメンバーを含む構造体。

FileOffset

キャッシュされたファイル内の開始バイトオフセット。

長さ

読み取りまたは書き込みを行うデータの長さ (バイト単位)。

LockKey

ターゲットファイルのバイト範囲ロックに関連付けられたキー値。読み取りまたは書き込みを行う範囲がファイル内の排他的にロックされている範囲の場合、このパラメーターはその共有ロックのキーである必要があります。共有ロックは、呼び出し元のスレッドの親プロセスによって保持されている必要があります。それ以外の場合、このパラメーターは無視されます。

CheckForReadOperation

この操作で読み取りまたは書き込み操作を確認するかどうかを指定します。読み取り操作の場合はTRUEに、書き込み操作の場合はFALSEに設定されます。

注釈

IRP_MJ の**FLT_PARAMETERS** 構造体_高速_IO_チェック_(可能な操作_に、コールバックによって表される迅速な操作のパラメーターが含まれている場合)データ (**FLT_CALLBACK_data**) 構造体。これは、FLT_IO_パラメーター_ブロック構造体に含まれています。

IRP_MJ_高速_IO_チェック_可能であれば高速な i/o 操作であることを確認します。

要件

Header	Fltkernel.h (Fltkernel.h を含む)

関連項目

[FLT_コールバック_データ](#)

[FLT_IO_パラメーター_ブロック](#)

[FLT_は_高速な操作_](#)

[FLT_は_FS_フィルターの_操作です。](#)

[FLT_は_IRP_操作です](#)

[FLT_パラメーター](#)

[FsRtlAreThereCurrentFileLocks](#)

[FsRtlCopyRead](#)

[FsRtlCopyWrite](#)

IRP_MJ_FILE_SYSTEM_CONTROL 共用体の FLT_PARAMETERS

2020/08/04 • [Edit Online](#)

操作の **FLT_IO_PARAMETER_BLOCK** 構造体の **MajorFunction** フィールドが **IRP_MJ_FILE_SYSTEM_CONTROL** 場合に使用される共用体コンポーネント。

構文

```
typedef union _FLT_PARAMETERS {
    ... ;
    union {
        struct {
            PVPB          Vpb;
            PDEVICE_OBJECT DeviceObject;
        } VerifyVolume;
        struct {
            ULONG          OutputBufferLength;
            ULONG POINTER_ALIGNMENT InputBufferLength;
            ULONG POINTER_ALIGNMENT FsControlCode;
        } Common;
        struct {
            ULONG          OutputBufferLength;
            ULONG POINTER_ALIGNMENT InputBufferLength;
            ULONG POINTER_ALIGNMENT FsControlCode;
            PVOID          InputBuffer;
            PVOID          OutputBuffer;
            PMDL           OutputMdlAddress;
        } Neither;
        struct {
            ULONG          OutputBufferLength;
            ULONG POINTER_ALIGNMENT InputBufferLength;
            ULONG POINTER_ALIGNMENT FsControlCode;
            PVOID          SystemBuffer;
        } Buffered;
        struct {
            ULONG          OutputBufferLength;
            ULONG POINTER_ALIGNMENT InputBufferLength;
            ULONG POINTER_ALIGNMENT FsControlCode;
            PVOID          InputSystemBuffer;
            PVOID          OutputBuffer;
            PMDL           OutputMdlAddress;
        } Direct;
    } FileSystemControl;
    ... ;
} FLT_PARAMETERS, *PFLT_PARAMETERS;
```

メンバー

FileSystemControl

次のメンバーを含む構造体。

VerifyVolume

IRP_MN_VERIFY_VOLUME 操作に使用される共用体コンポーネントです。

Vpb

検証するボリュームのボリュームパラメーターブロック (VPB) へのポインター。

DeviceObject

検証するボリュームのデバイスオブジェクトへのポインター。

共通

IRP_MN_KERNEL_CALL および IRP_MN_USER_FS_REQUEST 操作のすべてのバッファリングメソッドに使用される共用体コンポーネント。

OutputBufferLength

OutputbufferまたはDirectによって参照されていないバッファの長さ (バイト単位)。

InputBufferLength

InputBuffer、Buffered.Systembuffer、または直接の inputsystembufferメンバーが指すバッファの長さ (バイト単位)。

FsControlCode

ターゲットデバイスのファイルシステム、ファイルシステムフィルター、またはミニフィルタードライバに渡される FSCTL 関数コード。

IOCTL 要求と FSCTL 要求の詳細については、「カーネルモードアーキテクチャガイド」の「[i/o 制御コードの使用](#)」および Microsoft Windows SDK のドキュメントの「デバイスの入力と出力の制御コード」を参照してください。(このリソースは、一部の言語および国では使用できません。)

どちらもオフ

バッファリングメソッドが METHOD_NEITHER 場合に IRP_MN_KERNEL_CALL および IRP_MN_USER_FS_REQUEST 操作に使用される共用体コンポーネント。バッファリングメソッドの詳細については、「カーネルモードアーキテクチャガイド」の「[i/o 制御コードの定義](#)」を参照してください。(このリソースは、一部の言語および国では使用できません。)

InputBuffer

操作の元の要求元が指定した入力バッファのユーザーモード仮想アドレス。I/o マネージャーとフィルターマネージャーでは、これらのアドレスは検証されません。ユーザー領域のアドレスが有効であることを確認するには、[ProbeForRead](#)、[ProbeForWrite](#)、[MmProbeAndLockPages](#)などのルーチンを使用して、try/exceptブロック内のすべてのバッファ参照を含むようにする必要があります。詳細については、「カーネルモードアーキテクチャガイド」の「[バッファを使用したユーザー領域のアドレスの参照](#)」を参照してください。(このリソースは、一部の言語および国では使用できません。)

OutputBuffer

操作の元の要求元が指定した出力バッファのユーザーモード仮想アドレス。I/o マネージャーとフィルターマネージャーでは、これらのアドレスは検証されません。ユーザー領域のアドレスが有効であることを確認するには、[ProbeForRead](#)、[ProbeForWrite](#)、[MmProbeAndLockPages](#)などのルーチンを使用して、try/exceptブロック内のすべてのバッファ参照を含むようにする必要があります。詳細については、「カーネルモードアーキテクチャガイド」の「[バッファを使用したユーザー領域のアドレスの参照](#)」を参照してください。(このリソースは、一部の言語および国では使用できません。)

OutputBufferは省略可能であり、MDL が指定されていない場合は NULL にすることができます。OutputMdlAddress。「[解説](#)」を参照してください。

OutputMdlAddress

Outputbufferメンバーが指しているバッファを記述するメモリ記述子リスト (MDL) のアドレス。このメンバーは省略可能であり、バッファがOutputBuffer のいずれにも指定されていない場合はNULLにすることができます。

付き

バッファリングメソッドが METHOD_BUFFERED 場合に IRP_MN_KERNEL_CALL および IRP_MN_USER_FS_REQUEST 操作に使用される共用体コンポーネント。バッファリングメソッドの詳細については、「カーネルモードアーキテクチャガイド」の「[i/o 制御コードの定義](#)」を参照してください。(このリソースは、一部の言語および国では使用できません。)

SystemBuffer

操作に対してシステムが割り当てたバッファのアドレス。METHOD_BUFFERED i/o では、このバッファは入力と出力の両方に使用されます。詳細については、「カーネルモードアーキテクチャガイド」の「[データバッファにアクセスするためのメソッド](#)」を参照してください。(このリソースは、一部の言語および国では使用できません。)

直接

バッファリングメソッドが METHOD_IN_DIRECT または METHOD_OUT_DIRECT 場合に、IRP_MN_KERNEL_CALL 操作および IRP_MN_USER_FS_REQUEST 操作に使用される共用体コンポーネント。バッファリングメソッドの詳細については、「カーネルモードアーキテクチャガイド」の「[i/o 制御コードの定義](#)」を参照してください。

InputSystemBuffer

操作の入力バッファのアドレス。このバッファは、カーネルモードから安全にアクセスできるように、オペレーティングシステムによってロックされています。詳細については、「カーネルモードアーキテクチャガイド」の「[データバッファにアクセスするためのメソッド](#)」を参照してください。(このリソースは、一部の言語および国では使用できません。)

OutputBuffer

操作の元の要求元が指定した出力バッファのユーザーモード仮想アドレス。ダイレクト i/o では、METHOD_NEITHER i/o とは異なり、オペレーティングシステムはこのバッファをロックダウンして、ミニパスが i/o 操作の元の要求元と同じプロセスコンテキストにある限り、カーネルモードから安全にアクセスできるようにします。(それ以外の場合は、

[MmGetSystemAddressForMdlSafe](#) を呼び出して、Outputmdladdress メンバーが指す MDL からシステムアドレスを取得する必要があります)。詳細については、「カーネルモードアーキテクチャガイド」の「[direct I/O とエラーをダイレクト I/O で使用する](#)」を参照してください。(このリソースは、一部の言語および国では使用できません。)

OutputMdlAddress

直接の outputbuffer メンバーが指すバッファを記述するメモリ記述子リスト (MDL) のアドレス。このメンバーは必須であり、NULL にすることはできません。

注釈

[IRP_MJ_FILE_SYSTEM_CONTROL](#) 操作の [FLT_PARAMETERS](#) 構造体には、コールバックデータ ([FLT_CALLBACK_DATA](#)) 構造体によって表されるファイルシステムコントロール情報操作のパラメーターが含まれています。これは [FLT_IO_PARAMETER_BLOCK](#) 構造体に含まれています。

Outputbuffer と mdladdress バッファの両方が指定されていない場合は、ミニフィルターで MDL を使用することをお勧めします。

ミニフィルターによって、どちらの値も変更されない場合は、フィルターマネージャーによって、現在格納されている MDLのうち mdladdress が解放されます。mdladdress の前の値は復元されません。

IRP_MJ_FILE_SYSTEM_CONTROL は、IRP ベースの操作です。

要件

ヘッダー: fltkernel.h (fltkernel.h を含む)

関連項目

[FLT_CALLBACK_DATA](#)

[FLT_IO_PARAMETER_BLOCK](#)

[FLT_IS_FASTIO_OPERATION](#)

[FLT_IS_FS_FILTER_OPERATION](#)

[FLT_IS_IRP_OPERATION](#)

[FLT_PARAMETERS](#)

[FltFsControlFile](#)

[IoBuildAsynchronousFsdRequest](#)

[IoBuildSynchronousFsdRequest](#)

IoVerifyVolume

IRP_MJ_FILE_SYSTEM_CONTROL

MmGetSystemAddressForMdlSafe

MmProbeAndLockPages

ProbeForRead

ProbeForWrite

ZwFsControlFile

IRP_MJ_LOCK_CONTROL union の FLT_パラメーター

2020/04/24 • [Edit Online](#)

次の共用体コンポーネントは、[FLT_IO_パラメーター_のブロック](#)構造のMajorFunctionフィールドが、操作のブロック構造体である場合に使用されます。これは、[MJ_ロック_コントロール](#)です。

構文

```
typedef union _FLT_PARAMETERS {
    ... ;
    struct {
        PLARGE_INTEGER          Length;
        ULONG_PTR_ALIGNMENT Key;
        LARGE_INTEGER          ByteOffset;
        PEPROCESS              ProcessId;
        BOOLEAN                 FailImmediately;
        BOOLEAN                 ExclusiveLock;
    } LockControl;
    ... ;
} FLT_PARAMETERS, *PFLT_PARAMETERS;
```

Members

LockControl

次のメンバーを含む構造体。

長さ

ロックする範囲の長さ (バイト単位) を指定する変数へのポインター。

Key

バイト範囲ロックに割り当てられるキー値。

ByteOffset

ロックする範囲のファイル内での開始バイトオフセット。

ProcessId

バイト範囲ロックを要求したプロセスのプロセス ID への不透明なポインター。

FailImmediately ちに

ロックを直ちに許可できない場合に、ロック要求が失敗するかどうかを指定するブール値。要求が許可されるまで待機状態になるか、またはできない場合はTRUEになるように、このメンバーはFALSEに設定されます。

ExclusiveLock

排他ロックが要求されるかどうかを指定するブール値。排他ロックが要求された場合、このメンバーはTRUEに設定され、共有ロックが要求された場合はFALSEに設定されます。

注釈

[MJ_ロック_制御操作_](#)のFLT_PARAMETERS構造体は、コールバックデータ ([FLT_callback_data](#)) 構造体によって表されます。これは、[FLT_IO_パラメーター_ブロック](#)構造体に含まれています。

IRP_MJ_LOCK_CONTROL は、IRP ベースの i/o 操作でも、高速な i/o 操作でもかまいません。

要件

Header	Fltkernel.h (Ftkernel.h を含む)

関連項目

[アクセス_マスク](#)

[_の状態へのアクセス](#)

[FLT_コールバック_データ](#)

[FLT_IO_パラメーター_ブロック](#)

[FLT_は_高速な操作_](#)

[FLT_は_FS_フィルターの_操作です。](#)

[FLT_は_IRP_操作です](#)

[FLT_パラメーター](#)

[FltAllocateFileLock](#)

[FltCheckLockForReadAccess](#)

[FltCheckLockForWriteAccess](#)

[FltFreeFileLock](#)

[FltInitializeFileLock](#)

[FltProcessFileLock](#)

[FltUninitializeFileLock](#)

[IRP_MJ_ロック_コントロール](#)

[PFLT__ロック_コールバック_データ_ルーチンの完了](#)

[PUNLOCK_ルーチン](#)

IRP_MJ_MDL_読み取り共用体の FLT_パラメーター

2020/04/24 • [Edit Online](#)

次の共用体コンポーネントは、[FLT_IO_パラメーターの_ブロック](#)構造体の MJ が IRP_MDL_READである場合に使用されます。

構文

```
typedef union _FLT_PARAMETERS {
    ... ;
    struct {
        LARGE_INTEGER          FileOffset;
        ULONG_PTR_ALIGNMENT Length;
        ULONG_PTR_ALIGNMENT Key;
        PMDL                    *MdlChain;
    } MdlRead;
    ... ;
} FLT_PARAMETERS, *PFLT_PARAMETERS;
```

Members

MdlRead

次のメンバーを含む構造体。

FileOffset

キャッシュされたファイル内の開始バイト。

長さ

キャッシュされたファイルから読み取るデータの長さ (バイト単位)。

Key

ターゲットファイルのバイト範囲ロックに関連付けられたキー値。読み取り対象の範囲がファイル内で排他的にロックされている範囲と重複している場合、このパラメーターはその排他ロックのキーである必要があります。排他ロックは、呼び出し元のスレッドの親プロセスによって保持されている必要があります。それ以外の場合、このパラメーターは無視されます。

MdlChain

読み取るデータを含むページを記述する1つ以上のメモリ記述子リスト (MDL) のチェーンへのポインターを受け取る変数へのポインター。

注釈

IRP_MJ_MDL_読み取り操作の [FLT_parameters](#) 構造体には、コールバックデータ ([FLT_callback_data](#)) によって表される高速 i/o mdlread 操作のパラメーターが含まれています。データ. これは、FLT_IO_パラメーター_ブロック構造体に含まれています。

高速 i/o IRP_MJ_MDL_読み取り要求が失敗した場合、i/o の発行者によって、要求を再発行する方法が決まります。ミニフィルターでは、必ずしも IRP ベースの IRP_MJ_MDL_読み取ることはできません。たとえば、irp 要求は、IRP_MJ_READ/IRP_MDL として再発行される可能性があります。

IRP_MJ_MDL_READ は、高速な i/o 操作です。

要件

Header	Fltkernel.h (Fltkernel.h を含む)

関連項目

[FLT_コールバック_データ](#)

[FLT_IO_パラメーター_ブロック](#)

[FLT_は_高速な操作_](#)

[FLT_は_FS_フィルターの_操作です。](#)

[FLT_は_IRP_操作です](#)

[FLT_パラメーター](#)

IRP_MJ_MDL_読み取り_完全な共用体の FLT_パラメーター

2020/04/24 • [Edit Online](#)

次の共用体コンポーネントは、[FLT_IO_パラメーター_ブロック](#)構造のMajorFunctionフィールドが、操作に対する IRP_MJ_MDL_READ_完了したときに使用されます。

構文

```
typedef union _FLT_PARAMETERS {
    ... ;
    struct {
        PMDL MdlChain;
    } MdlReadComplete;
    ... ;
} FLT_PARAMETERS, *PFLT_PARAMETERS;
```

Members

MdlReadComplete

次のメンバーを含む構造体。

MdlChain

キャッシュされたファイルから読み取られるデータを含むページを記述する1つ以上のメモリ記述子リスト (MDL) のチェーンへのポインターを受け取る変数へのポインター。

注釈

IRP_MJ_MDL の[FLT_PARAMETERS](#) 構造体_READ_COMPLETE 操作には、コールバックデータで表される高速 I/O Mdlreadcomplete操作のパラメーターが含まれてい [_コールバック_データ](#)) 構造体。これは、FLT_IO_パラメーター_ブロック構造体に含まれています。

IRP_MJ_MDL_READ_COMPLETE は、高速な i/o 操作です。

要件

Header	Fltkernel.h (Fltkernel.h を含む)

関連項目

[FLT_コールバック_データ](#)

[FLT_IO_パラメーター_ブロック](#)

[FLT_は_高速な操作_](#)

[FLT_は_FS_フィルターの_操作です。](#)

[FLT_は_IRP_操作です](#)

IRP_MJ_MDL_書き込み_完全な共用体の FLT_パラメーター

2020/04/24 • [Edit Online](#)

次の共用体コンポーネントは、[FLT_IO_パラメーター_のブロック](#)構造のMajorFunctionフィールドが、操作に対する IRP_MJ_MDL_書き込み_完了したときに使用されます。

構文

```
typedef union _FLT_PARAMETERS {
    ...    ;
    struct {
        LARGE_INTEGER FileOffset;
        PMDL          MdlChain;
    } MdlWriteComplete;
    ...    ;
} FLT_PARAMETERS, *PFLT_PARAMETERS;
```

Members

MdlWriteComplete

次のメンバーを含む構造体。

FileOffset

キャッシュされたファイル内の開始バイト。

MdlChain

キャッシュされたファイルに書き込まれるデータを含むページを記述する1つ以上のメモリ記述子リスト (MDL) のチェーンへのポインターを受け取る変数へのポインター。

注釈

IRP_MJ_MDL の[FLT_PARAMETERS](#) 構造体_書き込み_完了操作には、コールバックデータによって表される高速 i/o [MdlWriteComplete](#) 操作のパラメーターが含まれてい [_コールバック_データ](#)) 構造体。これは、FLT_IO_パラメーター_ブロック構造体に含まれています。

IRP_MJ_MDL_書き込み_完了は、高速な i/o 操作です。

要件

Header	Fltkernel .h (Fltkernel. h を含む)

関連項目

[FLT_コールバック_データ](#)

[FLT_IO_パラメーター_ブロック](#)

[FLT_は_高速な操作_](#)

FLT_は_FS_フィルターの_操作です。

FLT_は_IRP_操作です

FLT_パラメーター

FLT _ PARAMETERS IRP _ MJ _ NETWORK _ QUERY _ OPEN union

2020/08/04 • [Edit Online](#)

操作のFLT _ IO _ パラメーター _ ブロック構造のMajorFunctionフィールドが IRP _ MJ NETWORK QUERY OPEN の場合は、次の共用体コンポーネントが使用されます。

構文

```
typedef union _FLT_PARAMETERS {
    ... ;
    struct {
        PIRP Irp;
        PFILE_NETWORK_OPEN_INFORMATION NetworkInformation;
    } NetworkQueryOpen;
    ... ;
} FLT_PARAMETERS, *PFLT_PARAMETERS;
```

メンバー

NetworkQueryOpen

次のメンバーを含む構造体。

Irp

このオープン操作を表す create IRP へのポインター。この IRP は、一般的な open/create コードにファイルシステムによって使用されますが、実際には完了していません。

NetworkInformation

ファイルネットワークへのポインターは、ファイルに関する要求された情報を受け取るために、** _ _ _ 情報を開い**ています。

コメント

IRP MJ NETWORK QUERY open 操作のFLT _ parameters構造体には、_ _ _ _ コールバックデータ (FLT _ callback _ Data) 構造体によって表される networkqueryopen 操作のパラメーターが含まれています。FLT _ PARAMETERS構造体は、FLT _ IO _ パラメーター _ ブロック構造体に含まれています。

NOTE

IRP MJ NETWORK QUERY OPEN に関連付けられているファイルオブジェクト _ _ は、_ _ スタックベースのオブジェクトです。NetworkQueryOpen コールバックに登録されているフィルターは、このオブジェクトを参照できません。つまり、このスタックベースのファイルオブジェクトでは、ObReferenceObject または ObDereferenceObject を呼び出さないでください。また、オブジェクトへのポインターを保存しないでください。

IRP _ MJ _ NETWORK _ QUERY _ OPEN は、高速な i/o 操作です。これは、fa/Oqueryopen (Fa/Faoquerynetworkopeninfo ではない) 操作に相当します。この操作を行うには、フィルターを登録する必要があります。

必要条件

Header	Fltkernel.h (Fltkernel.h を含む)

関連項目

[ファイル _ ネットワークの _ オープン _ 情報](#)

[FLT _ CALLBACK _ データ](#)

[FLT _ IO _ パラメーター _ ブロック](#)

[FLT _ の _ _ 操作](#)

[FLT _ IS _ FS _ FILTER _ 操作](#)

[FLT _ は _ IRP _ 操作です](#)

[FLT _ パラメーター](#)

[FltQueryInformationFile](#)

[IRP _ MJ の _ クエリ _ 情報](#)

[ZwQueryInformationFile](#)

IRP_MJ_PNP 共用体の FLT_パラメーター

2020/04/24 • [Edit Online](#)

操作の [FLT_IO_パラメーター_ブロック](#) 構造の MajorFunction フィールドが [IRP_MJ_PNP](#) である場合に使用される共用体コンポーネント。

構文

```
typedef union _FLT_PARAMETERS {
    ... ;
    union {
        struct StartDevice;
        struct QueryDeviceRelations;
        struct QueryInterface;
        struct DeviceCapabilities;
        struct FilterResourceRequirements;
        struct ReadWriteConfig;
        struct SetLock;
        struct QueryId;
        struct QueryDeviceText;
        struct UsageNotification;
    } Pnp;
    ... ;
} FLT_PARAMETERS, *PFLT_PARAMETERS;
```

Members

全て

StartDevice

IRP_ に使用されるユニオンコンポーネントで、デバイス操作を開始_。この操作のパラメーターの詳細については、[_デバイスを起動_](#) には、IRP_ の参照エントリを参照してください。

QueryDeviceRelations

IRP_ クエリ_デバイス_リレーション操作に使用される共用体コンポーネント。この操作のパラメーターの詳細については、「[IRP_ クエリ_デバイス_関係](#)」の参照エントリを参照してください。

QueryInterface

IRP_ クエリ_インターフェイス操作に使用される共用体コンポーネントです。この操作のパラメーターの詳細については、「[IRP_ クエリ_インターフェイス](#)」の参照エントリを参照してください。

DeviceCapabilities

IRP_ クエリ_機能操作に使用される共用体コンポーネントです。この操作のパラメーターの詳細については、次を参照してください。[IRP_ は、_クエリ_機能](#) の参照エントリです。

FilterResourceRequirements

IRP_ フィルター_リソース_要件操作に使用される共用体コンポーネント。この操作のパラメーターの詳細については、「[IRP_ フィルター_リソース_要件](#)」の参照エントリを参照してください。

ReadWriteConfig

IRP_ に使用される共用体コンポーネント。[_読み取り_構成](#)と [IRP_、_構成](#)操作を書き込むことができます。この操作のパラメーターの詳細については、次を参照してください。[irp_ は、_読み取り_構成](#)と [irp_ の参照エントリ_書き込み_構成](#)。

SetLock

IRP_ 設定_ロック操作に使用される共用体コンポーネント。この操作のパラメーターの詳細については、次を参照してください。

[IRP_が_ロック_設定](#)されています。

QueryId

IRP__クエリ_ID 操作に使用される共用体コンポーネントです。この操作のパラメーターの詳細については、「[IRP__クエリ_ID](#)の参照エントリ」を参照してください。

QueryDeviceText

IRP__クエリ_デバイス_テキスト操作に使用される共用体コンポーネント。この操作のパラメーターの詳細については、「[IRP__クエリ_デバイス_テキスト](#)」の参照エントリを参照してください。

UsageNotification

IRP__デバイス_使用状況_通知操作に使用される共用体コンポーネント。この操作のパラメーターの詳細については、「[IRP__デバイス_使用状況_通知](#)」の参照エントリを参照してください。

注釈

[Irp_MJ_の pnp](#)操作の[FLT_parameters](#)構造体には、コールバックデータ ([FLT_callback_data](#)) によって表される irp ベース プラグアンドプレイ (pnp) 操作のパラメーターが含まれています。データ.これは、[FLT_IO_パラメーター_ブロック](#)構造体に含まれています。

IRP_MJ_PNP 操作は、IRP ベースの操作です。

要件

Header	Fltkernel.h (Fltkernel.h を含む)

関連項目

[FLT_コールバック_データ](#)

[FLT_IO_パラメーター_ブロック](#)

[FLT_は_高速な操作_](#)

[FLT_は_FS_フィルターの_操作です。](#)

[FLT_は_IRP_操作です](#)

[FLT_パラメーター](#)

[IRP_MJ_PNP](#)

[IRP_MJ_PNP \(WDK カーネルモードドライバアーキテクチャリファレンス\)](#)

[IRP__デバイス_使用状況_通知](#)

[IRP__フィルター_リソース_の要件](#)

[IRP__クエリ_機能](#)

[IRP__クエリ_デバイス_関係](#)

[IRP__クエリ_デバイス_テキスト](#)

[IRP__クエリ_ID](#)

[IRP__クエリ_インターフェイス](#)

[IRP__読み取り_構成](#)

IRP__設定_ロック

_デバイス起動_IRP_

IRP__書き込み_構成

IRP_MJ の FLT_パラメーター_準備_MDL_書き込み 共用体

2020/04/24 • [Edit Online](#)

次の共用体コンポーネントは、[FLT_IO_パラメーター_ブロック](#)構造のMAJORFUNCTIONフィールドが
IRP_MJ_PREPARE_MDL_WRITE である場合に使用されます。

構文

```
typedef union _FLT_PARAMETERS {  
    ... ;  
    struct {  
        LARGE_INTEGER      FileOffset;  
        ULONG POINTER_ALIGNMENT Length;  
        ULONG POINTER_ALIGNMENT Key;  
        PMDL                *MdlChain;  
    } PrepareMdlWrite;  
    ... ;  
} FLT_PARAMETERS, *PFLT_PARAMETERS;
```

Members

作成した Mdlwrite

次のメンバーを含む構造体。

FileOffset

キャッシュされたファイル内の開始バイト。

長さ

キャッシュされたファイルに書き込まれるデータの長さ (バイト単位)。

Key

ターゲットファイルのバイト範囲ロックに関連付けられたキー値。書き込まれる範囲がファイル内の排他的にロックされている範囲の場合、このパラメーターはその排他ロックのキーである必要があります。排他ロックは、呼び出し元のスレッドの親プロセスによって保持されている必要があります。それ以外の場合、このパラメーターは無視されます。

MdlChain

書き込むデータを含むページを記述する1つ以上のメモリ記述子リスト (MDL) のチェーンへのポインターを受け取る変数へのポインター。

注釈

IRP_MJ_PREPARE_MDL_書き込み操作の[FLT_PARAMETERS](#)構造体には、コールバックデータによって表される高速な I/O/書き込み操作のパラメーターが含まれてい [コールバック_データ](#) 構造体。これは、FLT_IO_パラメーター_ブロック構造体に含まれています。

高速 i/o IRP_MJ_PREPARE_MDL_書き込み要求が失敗した場合は、i/o の発行者が要求を再発行する方法を決定します。ミニフィルターでは、必ずしも IRP ベースの IRP_MJ_MDL_書き込むことはできません。たとえば、irp 要求は、IRP_MJ_WRITE/IRP_MJ_MDLとして再発行される可能性があります。

IRP_MJ_PREPARE_MDL_WRITE は高速な i/o 操作です。

要件

Header	Fltkernel .h (Fltkernel. h を含む)

関連項目

[FLT_コールバック_データ](#)

[FLT_IO_パラメーター_ブロック](#)

[FLT_は_高速な操作_](#)

[FLT_は_FS_フィルターの_操作です。](#)

[FLT_は_IRP_操作です](#)

[FLT_パラメーター](#)

IRP_MJ_QUERY_EA 共用体の FLT_PARAMETERS

2020/08/04 • [Edit Online](#)

操作のFLT_IO_PARAMETER_BLOCK構造体のMajorFunctionフィールドがIRP_MJ_QUERY_EA場合に使用される共用体コンポーネント。

構文

```
typedef union _FLT_PARAMETERS {
    ... ;
    struct {
        ULONG                Length;
        PVOID                EaList;
        ULONG                EaListLength;
        ULONG_PTR_ALIGNMENT EaIndex;
        PVOID                EaBuffer;
        PMDL                 MdlAddress;
    } QueryEa;
    ... ;
} FLT_PARAMETERS, *PFLT_PARAMETERS;
```

メンバー

Quer

次のメンバーを含む FLT_PARAMETERS union 内の構造体。

[データ型]

EaBufferが指すバッファの長さ (バイト単位)。

EaList

クエリ対象の拡張属性を指定する、呼び出し元が指定した FILE_GET_EA_INFORMATION 構造化された入力バッファへのポインター。

EaListLength

EaListが指すバッファの長さ (バイト単位)。

Eライド Dex

拡張属性リストのスキャンを開始する位置のエントリのインデックス。操作の FLT_IO_PARAMETER_BLOCK 構造で SL_INDEX_SPECIFIED フラグが設定されていない場合、またはEaListが空でないリストを指している場合、このパラメーターは無視されます。

EaBuffer

拡張属性値が返される、呼び出し元が指定したFILE_FULL_EA_INFORMATION構造の出力バッファへのポインター。このメンバーは省略可能であり、MDL がMdladdressに指定されている場合はNULLにすることができます。「解説」を参照してください。

MdlAddress

EaBufferが指すバッファを記述するメモリ記述子リスト (MDL) のアドレス。このメンバーは省略可能であり、EaBufferでバッファが指定されている場合はNULLにすることができます。「解説」を参照してください。

注釈

IRP_MJ_QUERY_EA操作のFLT_PARAMETERS構造体には、コールバックデータ (FLT_CALLBACK_DATA) 構造体によっ

て表される、IRP ベースのクエリの拡張属性情報操作のパラメーターが含まれています。これは FLT_IO_PARAMETER_BLOCK 構造体に含まれています。

EaBufferとmdladdressの両方のバッファを指定する場合は、ミニフィルターを使用することをお勧めします。EaBufferが指すメモリは、呼び出しプロセスのコンテキスト内でアクセスされているユーザーモードアドレスである場合、またはカーネルモードアドレスの場合に有効です。

ミニフィルターによってmdladdressの値が変更された場合、事後操作コールバックの後、フィルターマネージャーは現在mdladdressに格納されている MDL を解放し、mdladdressの前の値を復元します。

IRP_MJ_QUERY_EA は、IRP ベースの操作です。

要件

ヘッダー: fltkernel .H (fltkernel .h を含む)

関連項目

[FILE_FULL_EA_INFORMATION](#)

[FLT_CALLBACK_DATA](#)

[FLT_IO_PARAMETER_BLOCK](#)

[FLT_IS_FASTIO_OPERATION](#)

[FLT_IS_FS_FILTER_OPERATION](#)

[FLT_IS_IRP_OPERATION](#)

[FLT_PARAMETERS](#)

[IoCheckEaBufferValidity](#)

[IRP_MJ_QUERY_EA](#)

IRP_MJ_クエリ_情報共用体の FLT_パラメーター

2020/04/24 • [Edit Online](#)

FLT_IO_パラメーター 操作のブロック構造のMajorFunctionフィールドが**IRP_MJ_クエリ_情報**である場合に使用される共用体コンポーネント。

構文

```
typedef union _FLT_PARAMETERS {
    ... ;
    struct {
        ULONG Length;
        FILE_INFORMATION_CLASS POINTER_ALIGNMENT FileInformationClass;
        PVOID InfoBuffer;
    } QueryFileInformation;
    ... ;
} FLT_PARAMETERS, *PFLT_PARAMETERS;
```

Members

QueryFileInformation

次のメンバーを含む構造体。

長さ

インフォバッファーに格納されているバッファーの長さ (バイト単位)。

FileInformationClass

返されるファイル情報の種類。次のいずれかです。

VALUE	
■■	ファイルの FILE_ALL_INFORMATION 構造体を返します。
FileAttributeTagInformation	ファイルの FILE_ATTRIBUTE_TAG_INFORMATION 構造体を返します。
FileBasicInformation	ファイルの FILE_BASIC_INFORMATION 構造体を返します。
Filecompression■■	ファイルの FILE_COMPRESSION_INFORMATION 構造体を返します。
FileEaInformation	ファイルの FILE_EA_INFORMATION 構造体を返します。
FileInternalInformation	ファイルの FILE_INTERNAL_INFORMATION 構造体を返します。

VALUE	戻り値
FileMoveClusterInformation	ファイルの FILE_MOVE_CLUSTER_INFORMATION 構造体を返します。
FileNameInformation	ファイルの FILE_NAME_INFORMATION 構造体を返します。
FileNetworkOpenInformation	ファイルの1つの FILE_NETWORK_OPEN_INFORMATION 構造体を返します。
FilePositionInformation	ファイルの1つの FILE_POSITION_INFORMATION 構造体を返します。
FileStandardInformation	ファイルの1つの FILE_STANDARD_INFORMATION 構造体を返します。
FileStreamInformation	ファイルの1つの FILE_STREAM_INFORMATION 構造体を返します。

インフォバッファー
ファイル情報が返される出力バッファーへのポインター。

注釈

IRP_MJ_クエリ_情報操作の [FLT_parameters](#) 構造体には、コールバックデータ ([FLT_callback_data](#)) 構造によって表されるクエリ情報操作のパラメーターが含まれています。これは、FLT_IO_パラメーター_ブロック構造体に含まれています。

IRP_MJ_クエリ_情報は、IRP ベースの操作でも、高速な i/o 操作でもかまいません。

要件

Header	Fltkernel.h (Fltkernel.h を含む)

関連項目

[ファイル_属性_タグ_情報](#)

[ファイル_基本_情報](#)

[ファイル_内部_情報](#)

[ファイル_名_情報](#)

[ファイル_ネットワーク__情報を開く](#)

[ファイル_位置_情報](#)

[ファイル_位置_情報](#) [ファイル_標準_情報](#)

[ファイル_ストリームの_情報](#)

[FLT_コールバック_データ](#)

[FLT_IO_パラメーター_ブロック](#)

[FLT_は_高速な操作_](#)

[FLT_は_FS_フィルターの_操作です。](#)

[FLT_は_IRP_操作です](#)

[FLT_パラメーター](#)

[IRP_MJ_クエリ_情報](#)

IRP_MJ_QUERY_OPEN union の FLT_パラメーター

2020/04/24 • [Edit Online](#)

操作の[FLT_IO_パラメーター_BLOCK](#)構造体のMAJORFUNCTIONフィールドが IRP_MJ_QUERY_OPEN の場合には、次の共用体コンポーネントが使用されます。

構文

```
typedef union _FLT_PARAMETERS {
    ... ;
    struct {
        PIRP                Irp;
        PVOID                FileInformation;
        PULONG               Length;
        FILE_INFORMATION_CLASS FileInformationClass;
    } QueryOpen;
    ... ;
} FLT_PARAMETERS, *PFLT_PARAMETERS;
```

Members

Irp

- この操作に関連付けられている IRP へのポインター。

FileInformation

- ルーチンがファイルオブジェクトに関する要求された情報を書き込む、呼び出し元が割り当てたバッファへのポインター。
*Fileinformationclass*メンバーは、呼び出し元が要求する情報の種類を指定します。

長さ

- Fileinformation*が指すバッファのサイズ (バイト単位)。

FileInformationClass

- FileInformation* が指すバッファ内のファイルについて返される情報の種類を指定します。デバイスと中間ドライバーでは、次のいずれかの[FILE_INFORMATION_CLASS](#) 値を指定できます。その他の値を指定すると、呼び出しは失敗し、*PreQueryOpen/PostQueryOpen* 呼び出しに渡すことはできません。

FILE_INFORMATION_CLASS 値	説明
FileStatInformation	FILE_STAT_INFORMATION 構造体。この構造体には、アクセスマスクが含まれています。アクセスマスクの詳細については、「 ACCESS_MASK 」を参照してください。
FileStatLxInformation	FILE_STAT_LX_INFORMATION 構造体。この構造体には、アクセスマスクが含まれています。アクセスマスクの詳細については、「 ACCESS_MASK 」を参照してください。
FileCaseSensitiveInformation	FILE_CASE_SENSITIVE_INFORMATION 構造体。

注釈

IRP_MJ_QUERY_OPEN 操作のFLT_parameters構造体には、コールバックデータ (FLT_callback_data) 構造体によって表されるqueryopen操作のパラメーターが含まれています。これは、FLT_IO_パラメーター_ブロック構造体に含まれています。

IRP_MJ_QUERY_OPEN は、ファイルシステム (FSFilter) コールバック操作です。

FSFilter のコールバック操作の詳細については、Fsrtlregisterfilesystemfiltercallbacksのリファレンスエントリを参照してください。

要件

バージョン	Windows 10 バージョン1703以降のバージョンの Windows オペレーティングシステムで使用できます。
Header	Fltkernel .h (Fltkernel. h を含む)

関連項目

[FLT_コールバック_データ](#)

[FLT_IO_パラメーター_ブロック](#)

[FLT_は_高速な操作_](#)

[FLT_は_FS_フィルターの_操作です。](#)

[FLT_は_IRP_操作です](#)

[FLT_パラメーター](#)

[FsRtlRegisterFileSystemFilterCallbacks](#) バック

IRP_MJ_QUERY_QUOTA 共用体の FLT_PARAMETERS

2020/08/04 • [Edit Online](#)

操作の [FLT_IO_PARAMETER_BLOCK](#) 構造体の MajorFunction フィールドが [IRP_MJ_QUERY_QUOTA](#) 場合に使用される共用体コンポーネント。

構文

```
typedef union _FLT_PARAMETERS {
    ... ;
    struct {
        ULONG                Length;
        PSID                 StartSid;
        PFILE_GET_QUOTA_INFORMATION SidList;
        ULONG                SidListLength;
        PVOID                QuotaBuffer;
        PMDL                 MdlAddress;
    } QueryQuota;
    ... ;
} FLT_PARAMETERS, *PFLT_PARAMETERS;
```

メンバー

QueryQuota

次のメンバーを含む構造体。

[データ型]

QuotaBufferが指すバッファの長さ (バイト単位)。

StartSid

クォータ一覧のスクランを開始するエントリのセキュリティ識別子 (SID) へのポインター (省略可能)。操作の [FLT_IO_PARAMETER_BLOCK](#) 構造体に [SL_INDEX_SPECIFIED](#) フラグが設定されていない場合、または **Sidlist** が空でないリストを指している場合、このパラメーターは無視されます。

SidList

クォータ情報を照会する Sid を指定する、呼び出し元が指定した [FILE_GET_QUOTA_INFORMATION](#) 構造化入力バッファへのポインター。

SidListLength

Sidlistが指すバッファの長さ (バイト単位)。

QuotaBuffer

クォータ情報が返される、呼び出し元が指定した [FILE_QUOTA_INFORMATION](#) 構造化された出力バッファへのポインター。このメンバーは省略可能であり、MDL が **Mdladdress** に指定されている場合は NULL にすることができます。「解説」を参照してください。

MdlAddress

QuotaBufferが指すバッファを記述するメモリ記述子リスト (MDL) のアドレス。このメンバーは省略可能であり、**QuotaBuffer**でバッファが指定されている場合はNULLにすることができます。「解説」を参照してください。

注釈

[IRP_MJ_QUERY_QUOTA](#) 操作の [FLT_PARAMETERS](#) 構造体には、コールバックデータ ([FLT_CALLBACK_DATA](#)) 構造体

によって表される、IRP ベースのクエリのクォータ情報操作のパラメーターが含まれています。これは FLT_IO_PARAMETER_BLOCK 構造体に含まれています。

QuotaBufferとmdladdressの両方のバッファーを指定する場合は、ミニフィルターを使用することをお勧めします。QuotaBufferが指すメモリは、呼び出しプロセスのコンテキスト内でアクセスされているユーザーモードアドレスである場合、またはカーネルモードアドレスの場合に有効です。

ミニフィルターによってmdladdressの値が変更された場合、事後操作コールバックの後、フィルターマネージャーは現在mdladdressに格納されている MDL を解放し、mdladdressの前の値を復元します。

IRP_MJ_QUERY_QUOTA は、IRP ベースの操作です。

要件

ヘッダー: fltkernel .H (fltkernel .h を含む)

関連項目

[FILE_QUOTA_INFORMATION](#)

[FLT_CALLBACK_DATA](#)

[FLT_IO_PARAMETER_BLOCK](#)

[FLT_IS_FASTIO_OPERATION](#)

[FLT_IS_FS_FILTER_OPERATION](#)

[FLT_IS_IRP_OPERATION](#)

[FLT_PARAMETERS](#)

[IoCheckQuotaBufferValidity](#)

[IRP_MJ_QUERY_QUOTA](#)

[SID](#)

IRP_MJ_QUERY_SECURITY 共用体の FLT_PARAMETERS

2020/08/04 • [Edit Online](#)

操作の **FLT_IO_PARAMETER_BLOCK** 構造体の MajorFunction フィールドが **IRP_MJ_QUERY_SECURITY** 場合に使用される共用体コンポーネント。

構文

```
typedef union _FLT_PARAMETERS {
    ... ;
    struct {
        SECURITY_INFORMATION    SecurityInformation;
        ULONG POINTER_ALIGNMENT Length;
        PVOID                   SecurityBuffer;
        PDML                    MdlAddress;
    } QuerySecurity;
    ... ;
} FLT_PARAMETERS, *PFLT_PARAMETERS;
```

メンバー

QuerySecurity

次のメンバーを含む構造体。

SecurityInformation

照会するセキュリティ情報を指定する、呼び出し元から提供された **SECURITY_INFORMATION** 値へのポインター。次のいずれか:

SECURITY_INFORMATION ㉒	㉒
OWNER_SECURITY_INFORMATION	オブジェクトの所有者識別子が照会されています。READ_CONTROL アクセスが必要です。
GROUP_SECURITY_INFORMATION	オブジェクトのプライマリグループ識別子が照会されています。READ_CONTROL アクセスが必要です。
DACL_SECURITY_INFORMATION	オブジェクトの随意アクセス制御リスト (DACL) を照会しています。READ_CONTROL アクセスが必要です。
SACL_SECURITY_INFORMATION	オブジェクトのシステム ACL (SACL) を照会しています。ACCESS_SYSTEM_SECURITY アクセスが必要です。

[データ型]

Securitybuffer が指すバッファの長さ (バイト単位)。

SecurityBuffer

指定したオブジェクトのセキュリティ記述子のコピーを受け取る、呼び出し元から提供される出力バッファへのポインター。呼び出しプロセスには、オブジェクトのセキュリティステータスの指定した側面を表示する権限が必要です。 **SECURITY_DESCRIPTOR** 構造体は、自己相対形式で返されます。このメンバーは省略可能であり、MDL が Mdladdress に指定されている場合は NULL にすることができます。「解説」を参照してください。

MdlAddress

Securitybufferが指すバッファを記述するメモリ記述子リスト (MDL) のアドレス。このメンバーは省略可能であり、securitybufferにバッファが指定されている場合はNULLにすることができます。「[解説](#)」を参照してください。

注釈

[IRP_MJ_QUERY_SECURITY](#) 操作の [FLT_PARAMETERS](#) 構造体には、コールバックデータ ([FLT_CALLBACK_DATA](#)) 構造体によって表される、IRP ベースのクエリセキュリティ情報操作のパラメーターが含まれています。これは [FLT_IO_PARAMETER_BLOCK](#) 構造体に含まれています。

Securitybufferとmdladdressバッファの両方が指定されている場合は、ミニフィルターで MDL を使用することをお勧めします。Securitybufferが指すメモリは、呼び出しプロセスのコンテキスト内でアクセスされているユーザーモードアドレスである場合、またはカーネルモードアドレスの場合に有効です。

ミニフィルターによってmdladdressの値が変更された場合、事後操作コールバックの後、フィルターマネージャーは現在mdladdressに格納されている MDL を解放し、mdladdressの前の値を復元します。

Windows XP 以降では、FLT_IO_PARAMETER_BLOCK 構造体のTargetfileobjectメンバーが指すオブジェクトは、名前付きデータストリームを表すことができます。名前付きデータストリームの詳細については、「[FILE_STREAM_INFORMATION](#)」を参照してください。

IRP_MJ_QUERY_SECURITY は、IRP ベースの操作です。

要件

ヘッダー: fltkernel .H (fltkernel .h を含む)

関連項目

[FILE_STREAM_INFORMATION](#)

[FLT_CALLBACK_DATA](#)

[FLT_IO_PARAMETER_BLOCK](#)

[FLT_PARAMETERS](#)

[IRP_MJ_QUERY_SECURITY](#)

[SECURITY_DESCRIPTOR](#)

[SECURITY_INFORMATION](#)

IRP_MJ の FLT_パラメーター_クエリ_ボリューム_情報共用体

2020/04/24 • [Edit Online](#)

FLT_IO_パラメーターのMajorFunctionフィールドが操作の_ブロック構造体である場合に使用される共用体コンポーネントは、**IRP_MJ_クエリ_ボリューム_情報**です。

構文

```
typedef union _FLT_PARAMETERS {
    ...    ;
    struct {
        ULONG                                Length;
        FS_INFORMATION_CLASS POINTER_ALIGNMENT FsInformationClass;
    } QueryVolumeInformation;
    PVOID VolumeBuffer;
    ...    ;
} FLT_PARAMETERS, *PFLT_PARAMETERS;
```

Members

QueryVolumeInformation

次のメンバーを含む構造体。

長さ

Volumebufferに格納されているバッファの長さ (バイト単位)。

FsInformationClass

ファイルシステムが返すボリューム情報の種類。次のいずれかです。

VALUE	「
FileFsAttributeInformation	ボリュームラベル、シリアル番号、作成時刻など、ボリュームに関する情報を含む FILE_FS_VOLUME_INFORMATION を返します。
FileFsControlInformation	ボリュームに関するファイルシステムコントロール情報を含む FILE_FS_CONTROL_INFORMATION 構造体を返します。
FileFsDeviceInformation	ボリュームのデバイス情報を含む FILE_FS_DEVICE_INFORMATION 構造体を返します。

VALUE	II
FileFsDriverPathInformation	指定されたドライバーがボリュームの i/o パスにあるかどうかに関する情報を含む FILE_FS_DRIVER_PATH_INFORMATION 構造体を返します。IRP_MJ_QUERY_VOLUME_INFORMATION 要求の発信者は、IRP をファイルシステムボリュームのデバイススタックに送信する前に、そのドライバーの名前を FILE_FS_DRIVER_PATH_INFORMATION 構造体に格納する必要があります。
FileFsFullSizeInformation	ボリューム上で使用可能な領域の合計量に関する情報を含む FILE_FS_FULL_SIZE_INFORMATION 構造体を返します。
FileFsObjectIdInformation	ボリュームのファイルシステム固有のオブジェクト ID 情報を含む FILE_FS_OBJECTID_INFORMATION 構造体を返します。これは、オペレーティングシステムによって割り当てられる (グローバル一意識別子 (GUID) ベースの) 一意のボリューム名と同じではないことに注意してください。
FileFsSizeInformation	IRP_MJ_QUERY_VOLUME_INFORMATION 要求を発信したスレッドに関連付けられているユーザーが使用できるボリューム上の領域の量に関する情報を格納する FILE_FS_SIZE_INFORMATION 構造体を返します。
FileFsVolumeInformation	ボリュームラベル、シリアル番号、作成時刻など、ボリュームに関する情報を含む FILE_FS_VOLUME_INFORMATION を返します。
FileFsSectorSizeInformation	ボリュームの物理的および論理的なセクターサイズに関する情報を含む FILE_FS_SECTOR_SIZE_INFORMATION 構造体を返します。

VolumeBuffer
ボリューム情報が返される出力バッファーへのポインター。

注釈

IRP_MJ_クエリ_ボリューム_情報操作の[FLT_PARAMETERS](#) 構造体には、コールバックデータ ([FLT_callback_data](#)) 構造によって表される、irp ベースのクエリボリューム情報操作のパラメーターが含まれています。これは、FLT_IO_パラメーター_ブロック構造体に含まれています。

IRP_MJ_クエリ_ボリューム_情報は、IRP ベースの操作です。

要件

Header	Fltkernel.h (Fltkernel.h を含む)

関連項目

[ファイル_FS_属性_情報](#)

ファイル_FS_制御_情報

ファイル_FS_デバイス_情報

ファイル_FS_ドライバー_パス_情報

ファイル_FS_完全な_サイズ_情報

ファイル_FS_OBJECTID_情報

ファイル_fs_セクタ_サイズ_情報 ファイル_FS_サイズ_情報

ファイル_FS_ボリューム_情報

FLT_コールバック_データ

FLT_IO_パラメーター_ブロック

FLT_は_高速な操作_

FLT_は_FS_フィルターの_操作です。

FLT_は_IRP_操作です

FLT_パラメーター

IRP_MJ_クエリ_情報

ZwQueryVolumeInformationFile

IRP_MJ_READ 共用体の FLT_PARAMETERS

2020/08/04 • [Edit Online](#)

操作の **FLT_IO_PARAMETER_BLOCK** 構造体の MajorFunction フィールドが **IRP_MJ_READ** 場合は、次の共用体コンポーネントが使用されます。

構文

```
typedef union _FLT_PARAMETERS {
    ... ;
    struct {
        ULONG                Length;
        ULONG POINTER_ALIGNMENT Key;
        LARGE_INTEGER        ByteOffset;
        PVOID                ReadBuffer;
        PMDL                 MdlAddress;
    } Read;
    ... ;
} FLT_PARAMETERS, *PFLT_PARAMETERS;
```

メンバー

読み取り

次のメンバーを含む構造体。

[データ型]

読み取るデータの長さ (バイト単位)。

Key

ターゲットファイルのバイト範囲ロックに関連付けられたキー値。

ByteOffset

読み取るデータのファイル内のバイトオフセットを開始します。

ReadBuffer

ファイルから読み取られたデータを受け取るバッファーへのポインター。このメンバーは省略可能であり、MDL が Mdladdress に指定されている場合は NULL にすることができます。「解説」を参照してください。

MdlAddress

Readbuffer メンバーが指すバッファーを記述するメモリ記述子リスト (MDL) のアドレス。このメンバーは省略可能であり、Readbuffer にバッファーが指定されている場合は NULL にすることができます。「解説」を参照してください。

注釈

IRP_MJ_READ 操作の **FLT_PARAMETERS** 構造体には、コールバックデータ (**FLT_CALLBACK_DATA**) 構造体によって表される読み取り操作のパラメーターが含まれています。これは FLT_IO_PARAMETER_BLOCK 構造体に含まれています。

Readbuffer と mdladdress バッファーの両方が指定されている場合は、ミニフィルターで MDL を使用することをお勧めします。Readbuffer が指すメモリは、呼び出しプロセスのコンテキスト内でアクセスされているユーザーモードアドレスである場合、またはカーネルモードアドレスの場合に有効です。

ミニフィルターによって mdladdress の値が変更された場合、事後操作コールバックの後、フィルターマネージャーは現在 mdladdress に格納されている MDL を解放し、mdladdress の前の値を復元します。

IRP_MJ_READ は、IRP ベースの操作でも、高速な i/o 操作でもかまいません。

要件

ヘッダー: fltkernel .H (fltkernel .h を含む)

関連項目

[FLT_CALLBACK_DATA](#)

[FLT_IO_PARAMETER_BLOCK](#)

[FLT_IS_FASTIO_OPERATION](#)

[FLT_IS_FS_FILTER_OPERATION](#)

[FLT_IS_IRP_OPERATION](#)

[FLT_PARAMETERS](#)

[FtReadFile](#)

[IRP_MJ_READ](#)

[ZwReadFile](#)

IRP_MJ_RELEASE_の FLT_パラメーター_MOD_書き込み共用体

2020/04/24 • [Edit Online](#)

次の共用体コンポーネントは、[FLT_IO_パラメーター_のブロック](#)構造のMajorFunctionフィールドが、_MOD_WRITE_のIRP_MJ_RELEASE である場合に使用されます。

構文

```
typedef union _FLT_PARAMETERS {
    ... ;
    struct {
        PERESOURCE ResourceToRelease;
    } ReleaseForModifiedPageWriter;
    ... ;
} FLT_PARAMETERS, *PFLT_PARAMETERS;
```

Members

ReleaseForModifiedPageWriter

次のメンバーを含む構造体。

ResourceToRelease

解放されるリソースへのポインター。

注釈

IRP_MJ_RELEASE_の[FLT_PARAMETERS](#)構造体は_MOD_書き込み操作に対して、コールバックデータによって表されるReleaseForModifiedPageWriter操作のパラメーターを含みます ([FLT_CALLBACK_データ](#))データ. これは、FLT_IO_パラメーター_ブロック構造体に含まれています。

IRP_MJ_RELEASE_FOR_MOD_WRITE はファイルシステム (FSFilter) コールバック操作です。

FSFilter のコールバック操作の詳細については、[Fsrtlregisterfilesystemfiltercallbacks](#)のリファレンスエントリを参照してください。

要件

Header	Fltkernel.h (Fltkernel.h を含む)

関連項目

[FLT_コールバック_データ](#)

[FLT_IO_パラメーター_ブロック](#)

[FLT_は_高速な操作_](#)

[FLT_は_FS_フィルターの_操作です。](#)

[FLT_は_IRP_操作です](#)

[FLT_パラメーター](#)

[FsRtlRegisterFileSystemFilterCallbacks](#) バック

FLT_IRP のパラメーター_MJ_リリース_の_セクション_同期

2020/04/24 • [Edit Online](#)

IRP のパラメーターがない_MJ_リリース_の_セクション_同期します。

IRP_MJ_SET_EA 共用体の FLT_PARAMETERS

2020/08/04 • [Edit Online](#)

操作のFLT_IO_PARAMETER_BLOCK構造体のMajorFunctionフィールドがIRP_MJ_SET_EA場合に使用される共用体コンポーネント。

構文

```
typedef union _FLT_PARAMETERS {  
    ... ;  
    struct {  
        ULONG Length;  
        PVOID EaBuffer;  
        PMDL MdlAddress;  
    } SetEa;  
    ... ;  
} FLT_PARAMETERS, *PFLT_PARAMETERS;
```

メンバー

SetEa

次のメンバーを含む構造体。

[データ型]

EaBufferが指すバッファの長さ (バイト単位)。

EaBuffer

設定する拡張属性 (EA) 値を格納する、呼び出し元が提供するFILE_FULL_EA_INFORMATION構造化された入力バッファへのポインタ。このメンバーは省略可能であり、MDL がMdladdressに指定されている場合は NULL にすることができます。「解説」を参照してください。

MdlAddress

EaBufferが指すバッファを記述するメモリ記述子リスト (MDL) のアドレス。このメンバーは省略可能であり、EaBufferでバッファが指定されている場合はNULLにすることができます。「解説」を参照してください。

注釈

IRP_MJ_SET_EA操作のFLT_PARAMETERS構造体には、コールバックデータ (FLT_CALLBACK_DATA) 構造体によって表される拡張属性の設定のパラメーターが含まれています。これは FLT_IO_PARAMETER_BLOCK 構造体に含まれています。

EaBufferとmdladdressの両方のバッファを指定する場合は、ミニフィルターを使用することをお勧めします。EaBufferが指すメモリは、呼び出しプロセスのコンテキスト内でアクセスされているユーザーモードアドレスである場合、またはカーネルモードアドレスの場合に有効です。

ミニフィルターによってmdladdressの値が変更された場合、事後操作コールバックの後、フィルターマネージャーは現在mdladdressに格納されている MDL を解放し、mdladdressの前の値を復元します。

IRP_MJ_SET_EA は、IRP ベースの操作です。

要件

ヘッダー: fltkernel .H (fltkernel .h を含む)

関連項目

[FILE_FULL_EA_INFORMATION](#)

[FLT_CALLBACK_DATA](#)

[FLT_IO_PARAMETER_BLOCK](#)

[FLT_IS_FASTIO_OPERATION](#)

[FLT_IS_FS_FILTER_OPERATION](#)

[FLT_IS_IRP_OPERATION](#)

[FLT_PARAMETERS](#)

[IoCheckEaBufferValidity](#)

[IRP_MJ_SET_EA](#)

IRP_MJ の FLT_パラメーター_設定_情報共用体

2020/04/24 • [Edit Online](#)

操作のFLT_IO_パラメーター_ブロック構造のMajorFunctionフィールドがIRP_MJ__情報を設定している場合に使用される共用体コンポーネント。

構文

```
typedef union _FLT_PARAMETERS {
    ... ;
    struct {
        ULONG Length;
        FILE_INFORMATION_CLASS POINTER_ALIGNMENT FileInformationClass;
        PFILE_OBJECT ParentOfTarget;
        union {
            struct {
                BOOLEAN ReplaceIfExists;
                BOOLEAN AdvanceOnly;
            };
            ULONG ClusterCount;
            HANDLE DeleteHandle;
        };
        PVOID InfoBuffer;
    } SetFileInformation;
    ... ;
} FLT_PARAMETERS, *PFLT_PARAMETERS;
```

Members

SetFileInformation

次のメンバーを含む構造体。

長さ

インフォバッファに格納されているバッファの長さ (バイト単位)。

FileInformationClass

ファイルに設定する情報の種類。次のいずれかです。

VALUE	⌘
File■	ファイルのFILE_ALLOCATION_INFORMATIONを設定します。
FileBasicInformation	ファイルのFILE_BASIC_INFORMATIONを設定します。
FileDispositionInformation	ファイルのFILE_DISPOSITION_INFORMATIONを設定します。
FileEndOfFileInformation	ファイルのFILE_END_OF_FILE_INFORMATIONを設定します。

VALUE	II
FileLinkInformation	ファイルのFILE_LINK_INFORMATIONを設定します。
FilePositionInformation	ファイルのFILE_POSITION_INFORMATIONを設定します。
FileRenameInformation	ファイルのFILE_RENAME_INFORMATIONを設定します。
FileValidDataLengthInformation	ファイルのFILE_VALID_DATA_LENGTH_INFORMATIONを設定します。

ParentOfTarget

名前変更またはリンク操作に使用します。Infobuffer>FileNameに完全修飾ファイル名が含まれている場合、またはInfobuffer>RootdirectoryがNULL以外の場合、このメンバーは、のターゲットであるファイルの親ディレクトリのファイルオブジェクトポインターになります。運用. それ以外の場合はNULLになります。

(名前のない構造体)

次のメンバーを含む構造体。

置換 Eifexists

名前変更またはリンク操作に使用します。同じ名前のファイルが既に存在する場合に、指定したファイルに置き換えるように指定するには、TRUEに設定します。指定した名前のファイルが既に存在する場合に名前の変更またはリンク操作が失敗する場合は、FALSEに設定します。

AdvanceOnly

ファイルの終わり操作のフラグ。これにより、Fileinformationclass == Fileendoffileinformationの場合に、[_ファイル情報構造の__](#)、endoffileメンバーファイルの使用が決定されます。TRUEの場合、ファイルの新しい有効なデータ長は、現在の有効なデータ長が増加した場合にのみ、endoffileから設定されます。FALSEの場合は、新しいファイルサイズがendoffileから設定されます。

ClusterCount

システム用に予約されています。使わないでください。

DeleteHandle

システム用に予約されています。使わないでください。

インフォバッファー

設定するファイル情報を格納している入力バッファーへのポインター。

注釈

IRP_MJ_SET_INFORMATION 操作のFLT_parameters 構造体には、コールバックデータによって表される設定情報操作のパラメーターが含まれています (FLT_callback_data)。データ. これは、FLT_IO_パラメーター_ブロック構造体に含まれています。

IRP_MJ_SET_情報は、IRP ベースの操作です。

AdvanceOnlyメンバーは、ディスク上の現在の有効なデータ長をendoffileの新しい有効なデータ長に進めるようにファイルシステムに通知するために、キャッシュマネージャーによってTRUEに設定されます。AdvanceOnlyがFALSEの場合は、endoffileメンバー内の新しいファイルサイズが、現在のファイルサイズより大きいまたは小さい値に設定されています。

要件

Header	Fltkernel.h (Fltkernel.h を含む)

関連項目

- [ファイル_割り当て_情報](#)
- [ファイル_基本_情報](#)
- [ファイル_の後処理_情報](#)
- [ファイル__ファイル_情報の末尾_](#)
- [ファイル_リンク_情報](#)
- [ファイル_位置_情報](#)
- [ファイル__情報の名前変更](#)
- [ファイル_有効な_データ_の長さ_情報](#)
- [FLT_コールバック_データ](#)
- [FLT_IO_パラメーター_ブロック](#)
- [FLT_は_高速な操作_](#)
- [FLT_は_FS_フィルターの_操作です。](#)
- [FLT_は_IRP_操作です](#)
- [FLT_パラメーター](#)
- [IRP_MJ_設定_情報](#)

IRP_MJ_SET_QUOTA 共用体の FLT_PARAMETERS

2020/08/04 • [Edit Online](#)

操作の [FLT_IO_PARAMETER_BLOCK](#) 構造体の MajorFunction フィールドが [IRP_MJ_SET_QUOTA](#) 場合に使用される共用体コンポーネント。

構文

```
typedef union _FLT_PARAMETERS {
    ... ;
    struct {
        ULONG Length;
        PVOID QuotaBuffer;
        PMDL MdlAddress;
    } SetQuota;
    ... ;
} FLT_PARAMETERS, *PFLT_PARAMETERS;
```

メンバー

SetQuota

次のメンバーを含む構造体。

[データ型]

QuotaBuffer が指すバッファの長さ (バイト単位)。

QuotaBuffer

設定するクォータ情報を格納する、呼び出し元が指定した [FILE_QUOTA_INFORMATION](#) 構造化された入力バッファへのポインター。このメンバーは省略可能であり、MDL が Mdladdress に指定されている場合は NULL にすることができます。「[解説](#)」を参照してください。

MdlAddress

QuotaBuffer が指すバッファを記述するメモリ記述子リスト (MDL) のアドレス。このメンバーは省略可能であり、QuotaBuffer でバッファが指定されている場合は NULL にすることができます。「[解説](#)」を参照してください。

注釈

[IRP_MJ_SET_QUOTA](#) 操作の [FLT_PARAMETERS](#) 構造体には、コールバックデータ ([FLT_CALLBACK_DATA](#)) 構造体によって表される、設定-クォータ情報操作のパラメーターが含まれています。これは [FLT_IO_PARAMETER_BLOCK](#) 構造体に含まれています。

QuotaBuffer と mdladdress の両方のバッファを指定する場合は、ミニフィルターを使用することをお勧めします。QuotaBuffer が指すメモリは、呼び出しプロセスのコンテキスト内でアクセスされているユーザーモードアドレスである場合、またはカーネルモードアドレスの場合に有効です。

ミニフィルターによって mdladdress の値が変更された場合、事後操作コールバックの後、フィルターマネージャーは現在 mdladdress に格納されている MDL を解放し、mdladdress の前の値を復元します。

IRP_MJ_SET_QUOTA は、IRP ベースの操作です。

要件

ヘッダー: fltkernel .H (fltkernel .h を含む)

関連項目

[FILE_QUOTA_INFORMATION](#)

[FLT_CALLBACK_DATA](#)

[FLT_IO_PARAMETER_BLOCK](#)

[FLT_IS_FASTIO_OPERATION](#)

[FLT_IS_FS_FILTER_OPERATION](#)

[FLT_IS_IRP_OPERATION](#)

[FLT_PARAMETERS](#)

[IoCheckQuotaBufferValidity](#)

[IRP_MJ_SET_QUOTA](#)

IRP_MJ の FLT_パラメーター_設定_セキュリティ共用体

2020/04/24 • [Edit Online](#)

FLT_IO_パラメーターのMajorFunctionフィールドが操作の_ブロック構造体である場合に使用される共用体コンポーネントが、IRP_MJ__セキュリティを設定します。

構文

```
typedef union _FLT_PARAMETERS {
    ... ;
    struct {
        SECURITY_INFORMATION SecurityInformation;
        PSECURITY_DESCRIPTOR SecurityDescriptor;
    } SetSecurity;
    ... ;
} FLT_PARAMETERS, *PFLT_PARAMETERS;
```

Members

SetSecurity

次のメンバーを含む構造体。

SecurityInformation

セキュリティ記述子に設定するセキュリティ情報を指定する[セキュリティ_情報](#) 値へのポインター。この値には、次のいずれかを指定できます。

SECURITY_INFORMATION ㉒	㉒
DACL_SECURITY_INFORMATION	オブジェクトの随意アクセス制御リスト (DACL) が設定されています。WRITE_DAC アクセスが必要です。
GROUP_SECURITY_INFORMATION	オブジェクトのプライマリグループ識別子が設定されています。WRITE_OWNER アクセスが必要です。
OWNER_SECURITY_INFORMATION	オブジェクトの所有者識別子が設定されています。WRITE_OWNER アクセスが必要です。
SACL_SECURITY_INFORMATION	オブジェクトのシステム ACL (SACL) が設定されています。ACCESS_SYSTEM_SECURITY アクセスが必要です。

SecurityDescriptor

オブジェクトに割り当てるセキュリティ情報の値を格納する[セキュリティ_記述子](#)の構造体へのポインター。

注釈

IRP_MJ_設定_セキュリティ操作のFLT_parameters構造体には、コールバックデータ (FLT_callback によって表されるセキュリティ情報の設定操作のパラメーターが含まれてい_データ) 構造体。これは、FLT_IO_パラメーター_ブロック構造体に含ま

れています。

IRP_MJ_SET_セキュリティは、IRP ベースの操作です。

要件

Header	Fltkernel.h (Fltkernel.h を含む)

関連項目

[FLT_コールバック_データ](#)

[FLT_IO_パラメーター_ブロック](#)

[FLT_は_高速な操作_](#)

[FLT_は_FS_フィルターの_操作です。](#)

[FLT_は_IRP_操作です](#)

[FLT_パラメーター](#)

[IRP_MJ_設定_セキュリティ](#)

[セキュリティ_記述子](#)

[セキュリティ_情報](#)

IRP_MJ の FLT_パラメーター_設定_ボリューム_情報 共用体

2020/04/24 • [Edit Online](#)

FLT_IO_パラメーターのMajorFunctionフィールドが操作の_ブロック構造体である場合に使用される共用体コンポーネント_は、**_ボリューム_情報を設定**します。

構文

```
typedef union _FLT_PARAMETERS {
    ...    ;
    struct {
        ULONG                                Length;
        FS_INFORMATION_CLASS POINTER_ALIGNMENT FsInformationClass;
        PVOID                                VolumeBuffer;
    } SetVolumeInformation;
    ...    ;
} FLT_PARAMETERS, *PFLT_PARAMETERS;
```

Members

SetVolumeInformation

次のメンバーを含む構造体。

長さ

Volumebufferに格納されているバッファの長さ (バイト単位)。

FsInformationClass

ボリュームに対して設定する情報の種類。次のいずれかです。

VALUE	「
FileFsControlInformation	ボリュームの FILE_FS_CONTROL_INFORMATION を設定します。
FileFsLabelInformation	ボリュームの FILE_FS_LABEL_INFORMATION を設定します。
FileFsObjectIdInformation	ボリュームの FILE_FS_OBJECTID_INFORMATION を設定します。

VolumeBuffer

設定するボリューム情報の値を格納している入力バッファへのポインター。

注釈

IRP_MJ_設定_ボリューム_情報 操作の**FLT_PARAMETERS** 構造体には、コールバックデータによって表されるセットボリューム情報操作のパラメーターが含まれています (**FLT_CALLBACK_DATA**) 構造体。これは、FLT_IO_パラメーター_ブロック構造体に含まれています。

IRP_MJ_設定_ボリューム_情報は、IRP ベースの操作です。

要件

Header	Fltkernel .h (Fltkernel. h を含む)

関連項目

[ファイル_FS_制御_情報](#)

[ファイル_FS_ラベル_情報](#)

[ファイル_FS_OBJECTID_情報](#)

[FLT_コールバック_データ](#)

[FLT_IO_パラメーター_ブロック](#)

[FLT_は_高速な操作_](#)

[FLT_は_FS_フィルターの_操作です。](#)

[FLT_は_IRP_操作です](#)

[FLT_パラメーター](#)

[IRP_MJ__ボリューム_情報の設定](#)

[ZwSetVolumeInformationFile](#)

IRP_MJ_システム_コントロール共用体の FLT_パラメーター

2020/04/24 • [Edit Online](#)

操作のFLT_IO_パラメーター_ブロック構造のMAJORFUNCTIONフィールドが IRP_MJ_システム_コントロールである場合に使用される共用体コンポーネント。

構文

```
typedef union _FLT_PARAMETERS {
    ... ;
    struct {
        ULONG_PTR ProviderId;
        PVOID      DataPath;
        ULONG      BufferSize;
        PVOID      Buffer;
    } WMI;
    ... ;
} FLT_PARAMETERS, *PFLT_PARAMETERS;
```

Members

WMI

次のメンバーを含む構造体。

ProviderId

このパラメーターの意味は、操作のマイナー関数コードによって異なります。(次の「解説」を参照してください)。

データパス

このパラメーターの意味は、操作のマイナー関数コードによって異なります。(次の「解説」を参照してください)。

BufferSize

このパラメーターの意味は、操作のマイナー関数コードによって異なります。(次の「解説」を参照してください)。

格納

このパラメーターの意味は、操作のマイナー関数コードによって異なります。(次の「解説」を参照してください)。

注釈

IRP_MJ_システム_制御操作のFLT_PARAMETERS構造体には、コールバックデータ (FLT_callback_data) によって表されるシステム制御操作のためのパラメーターが含まれています。データ.これは、FLT_IO_パラメーター_ブロック構造体に含まれています。

IRP_MJ_システム_制御パラメーターの意味は、マイナー関数のコードによって異なります。(FLT_IO_パラメーター_ブロック構造体のminorfunctionメンバーを参照してください)。詳細については、次のマイナー関数コードの参照エントリを参照してください。

[IRP__1 つの_インスタンス_変更](#)

[IRP__1 つの_項目_変更](#)

[IRP___コレクションを無効にする](#)

[IRP___イベントを無効にする](#)

[IRP___コレクションを有効にする](#)

[IRP___イベントを有効にする](#)

[IRP__メソッドを実行_](#)

[IRP__クエリ_すべての_データ](#)

[IRP__クエリ_単一_インスタンス](#)

[IRP__REGINFO](#)

[IRP__REGINFO_EX](#)

IRP_MJ_システム_制御は、IRP ベースの操作です。

要件

Header	Fltkernel .h (Fltkernel. h を含む)

関連項目

[FLT_コールバック_データ](#)

[FLT_IO_パラメーター_ブロック](#)

[FLT_は_高速な操作_](#)

[FLT_は_FS_フィルターの_操作です。](#)

[FLT_は_IRP_操作です](#)

[FLT_パラメーター](#)

[IRP__1 つの_インスタンス_変更](#)

[IRP__1 つの_項目_変更](#)

[IRP___コレクションを無効にする](#)

[IRP___イベントを無効にする](#)

[IRP___コレクションを有効にする](#)

[IRP___イベントを有効にする](#)

[IRP__メソッドを実行_](#)

[IRP__クエリ_すべての_データ](#)

[IRP__クエリ_単一_インスタンス](#)

[IRP__REGINFO](#)

[IRP__REGINFO_EX](#)

IRP_MJ_ボリュームの FLT_パラメーター_マウント 共用体

2020/04/24 • [Edit Online](#)

次の共用体コンポーネントは、[FLT_IO_パラメーター_ブロック](#)構造のMajorFunctionフィールドが、操作に対する IRP_MJ_ボリューム_マウントされている場合に使用されます。

構文

```
typedef union _FLT_PARAMETERS {  
    ... ;  
    struct {  
        ULONG DeviceType;  
    } MountVolume;  
    ... ;  
} FLT_PARAMETERS, *PFLT_PARAMETERS;
```

Members

MountVolume

次のメンバーを含む構造体。

(Devicetype

新しくマウントされたボリュームのファイルシステムのボリュームデバイスオブジェクトのデバイスの種類。次のいずれかです。

ファイル_デバイス_CD_ROM_ファイル_システム

ファイル_デバイス_ディスク_ファイル_システム

ファイル_デバイス_ネットワーク_ファイル_システム

注釈

IRP_MJ_VOLUME_のマウント操作の[FLT_parameters](#)構造体には、コールバックデータ ([FLT_callback_data](#)) 構造によって表されるボリュームマウント操作のパラメーターが含まれています。これは、FLT_IO_パラメーター_ブロック構造体に含まれています。

IRP_MJ_ボリューム_マウントは高速 i/o 操作です。

要件

Header	Fltkernel.h (Fltkernel.h を含む)

関連項目

[FLT_コールバック_データ](#)

[FLT_IO_パラメーター_ブロック](#)

[FLT_は_高速な操作_](#)

[FLT_は_FS_フィルターの_操作です。](#)

FLT_は_IRP_操作です

FLT_パラメーター

IRP_MJ_ファイル_システム_コントロール

IRP_MJ_WRITE 共用体の FLT_PARAMETERS

2020/08/04 • [Edit Online](#)

操作の **FLT_IO_PARAMETER_BLOCK** 構造体の MajorFunction フィールドが **IRP_MJ_WRITE** 場合は、次の共用体コンポーネントが使用されます。

構文

```
typedef union _FLT_PARAMETERS {  
    ... ;  
    struct {  
        ULONG          Length;  
        ULONG          Key;  
        LARGE_INTEGER  ByteOffset;  
        PVOID          WriteBuffer;  
        PMDL           MdlAddress;  
    } Write;  
    ... ;  
} FLT_PARAMETERS, *PFLT_PARAMETERS;
```

メンバー

書き込み

次のメンバーを含む構造体。

[データ型]

書き込むデータの長さ (バイト単位)。

Key

ターゲットファイルのバイト範囲ロックに関連付けられたキー値。

ByteOffset

書き込むデータのファイル内のバイトオフセットを開始します。

WriteBuffer

ファイルに書き込まれるデータを格納しているバッファーへのポインター。このメンバーは省略可能であり、MDL が Mdladdress に指定されている場合は NULL にすることができます。「[解説](#)」を参照してください。

MdlAddress

Writebuffer メンバーが指すバッファーを記述するメモリ記述子リスト (MDL) のアドレス。このメンバーは省略可能であり、writebuffer にバッファーが指定されている場合は NULL にすることができます。「[解説](#)」を参照してください。

注釈

IRP_MJ_WRITE 操作の **FLT_PARAMETERS** 構造体には、コールバックデータ (**FLT_CALLBACK_DATA**) 構造体によって表される書き込み操作のパラメーターが含まれています。これは FLT_IO_PARAMETER_BLOCK 構造体に含まれています。

Writebuffer と mdladdress バッファーの両方が指定されている場合は、ミニフィルターで MDL を使用することをお勧めします。Writebuffer が指すメモリは、呼び出しプロセスのコンテキスト内でアクセスされているユーザーモードアドレスである場合、またはカーネルモードアドレスの場合に有効です。

ミニフィルターによって mdladdress の値が変更された場合、事後操作コールバックの後、フィルターマネージャーは現在 mdladdress に格納されている MDL を解放し、mdladdress の前の値を復元します。

IRP_MJ_WRITE は、IRP ベースの操作でも、高速な i/o 操作でもかまいません。

要件

ヘッダー: fltkernel .H (fltkernel .h を含む)

関連項目

[FLT_CALLBACK_DATA](#)

[FLT_IO_PARAMETER_BLOCK](#)

[FLT_IS_FASTIO_OPERATION](#)

[FLT_IS_FS_FILTER_OPERATION](#)

[FLT_IS_IRP_OPERATION](#)

[FLT_PARAMETERS](#)

[FtWriteFile](#)

[IRP_MJ_WRITE](#)

[ZwWriteFile](#)

IRP _ MJ _ CLEANUP (IFS)

2020/08/04 • [Edit Online](#)

送信時

IRP _ MJ _ CLEANUP 要求の受信は、ファイルオブジェクトのハンドル参照カウントが0に達したことを示します。(つまり、ファイルオブジェクトへのすべてのハンドルが閉じられています)。多くの場合、ユーザーモードアプリケーションが、ファイルオブジェクトへの最後の未処理ハンドルで Microsoft Win32 `CloseHandle`関数 (またはカーネルモードドライバが`zwclose`を呼び出したとき) を呼び出したときに送信されます。

ファイルオブジェクトへのすべてのハンドルが閉じられている場合は、必ずしもファイルオブジェクトが使用されなくなったことを意味するわけではないことに注意してください。キャッシュマネージャーやメモリマネージャーなどのシステムコンポーネントでは、ファイルオブジェクトへの未解決の参照が保持される場合があります。これらのコンポーネントは、IRP _ MJ _ CLEANUP 要求が受信された後でも、ファイルに対して読み取りまたは書き込みを行うことができます。

操作: ファイルシステムドライバー

ターゲットデバイスオブジェクトがファイルシステムのコントロールデバイスオブジェクトである場合は、ファイルシステムドライバーが IRP を完了する必要があります。

それ以外の場合、ファイルシステムドライバーはクリーンアップ要求を処理する必要があります。

操作: ファイルシステムフィルタードライバー

ターゲットデバイスオブジェクトがフィルタードライバーのコントロールデバイスオブジェクトである場合、フィルタードライバーは IRP を完了する必要があります。

それ以外の場合は、必要な処理を実行した後、フィルタードライバーが IRP をスタック上の次の下位のドライバーに渡す必要があります。

ファイルシステムフィルタードライバーの作成者は、`IoCreateStreamFileObject`によって IRP _ MJ _ CLEANUP 要求がボリュームのファイルシステムドライバースタックに送信されることに注意してください。ファイルシステムは、IRP MJ create 以外の操作の副作用としてストリームファイルオブジェクトを作成することが多いため、__、フィルタードライバーがストリームファイルオブジェクトの作成を確実に検出することは困難です。したがって、フィルタードライバーは、以前に表示さ__れて__いないファイルオブジェクトに対して、irp MJ CLEANUP 要求と irp MJ CLOSE 要求を受け取ることを想定して__います。

フィルタードライバーの作成者は、`IoCreateStreamFileObject`とは異なり、`IoCreateStreamFileMJ tLite`では、IRP __ のクリーンアップ要求がファイルシステムのドライバースタックに送信されないことにも注意してください。このため、ファイルシステムでは、IRP MJ create 以外の操作の副作用としてストリームファイルオブジェクトを作成することが多いため、__ フィルタードライバーがストリームファイルオブジェクトの作成を確実に検出することは困難です。そのため、フィルタードライバーは、以前に表示さ__れていないファイルオブジェクトに対して IRP MJ CLOSE 要求を受け取ることを想定しています。

パラメーター

ファイルシステムまたはフィルタードライバーは、指定された IRP で`IoGetLocation` `enti`を呼び出して、irp 内の独自の`スタックの場所`へのポインターを取得します。次の一覧には、`irpSp`として示されています。(IRP は、`irp`として表示されます)。ドライバーでは、クリーンアップ要求の処理中に、IRP の次のメンバーと IRP スタックの場所に設定されている情報を使用できます。

`DeviceObject`

ターゲットデバイスオブジェクトへのポインター。

`Irps -> フラグ`

この要求には、次のフラグが設定されています。

IRP _ CLOSE _ 操作

IRP _ 同期 _ API

Irp- > 最後の完了状態と要求された操作に関する情報を受け取る[IO _ 状態 _ ブロック](#)構造への *iostatus* ポインター。

Irpsp- > *DeviceObject*に関連付けられているファイルオブジェクトへの *FileObject* ポインター。

Irpsp- > *FileObject*パラメーターには、関連する*FileObject*フィールドへのポインターが含まれています。これは、ファイルオブジェクト構造でも _ あります。ファイルオブジェクト構造の "関連性のあるfileobject " フィールド _ は、IRP MJ CLEANUP の処理中は無効であり、_ _ 使用しないでください。

Irpsp- > *MajorFunction*は、IRP MJ クリーンアップを指定し _ _ ます。

関連項目

[IO _ スタックの _ 場所](#)

[IO _ 状態 _ ブロック](#)

[IoCreateStreamFileObject](#)

[IoCreateStreamFileLite Tlite](#)

[IoGetLocation Entiの場所](#)

[IRP](#)

[IRP _ MJ _ CLEANUP \(WDK カーネルリファレンス\)](#)

[IRP _ MJ _ CLOSE](#)

[IRP _ MJ の _ 作成](#)

[ZwClose](#)

IRP _ MJ _ CLOSE (IFS)

2020/08/04 • [Edit Online](#)

送信時

IRP MJ CLOSE 要求の受信は、ファイル _ オブジェクトの参照カウントが0に達したことを示します。通常、ファイルシステムドライバーまたはその他のカーネルモードコンポーネントがファイルオブジェクトで **ObDereferenceObject** を呼び出したことが原因です。通常、この要求はクリーンアップ要求の後に続きます。ただし、これは必ずしも、終了要求がクリーンアップ要求の直後に受信されることを意味するわけではありません。

操作: ファイルシステムドライバー

ターゲットデバイスオブジェクトがファイルシステムのコントロールデバイスオブジェクトである場合は、必要な処理を実行した後で、ファイルシステムドライバーが IRP を完了する必要があります。

それ以外の場合は、ファイルシステムドライバーが close 要求を処理する必要があります。

操作: ファイルシステムフィルタードライバー

ターゲットデバイスオブジェクトがフィルタードライバーのコントロールデバイスオブジェクトである場合、フィルタードライバーは、コントロールデバイスオブジェクトとの通信を終了し、IRP を完了するために必要な処理を実行する必要があります。

それ以外の場合、フィルタードライバーは、フィルタードライバーによって保持されているファイルごとのオブジェクトおよびファイルごとのオブジェクトのコンテキスト情報を削除するなど、必要な処理を実行した後、スタック上の次の下位のドライバーに IRP を渡す必要があります。

ファイルシステムフィルタードライバーの作成者は、**IoCreateStreamFileObject** によって **IRP _ MJ _ CLEANUP** 要求がボリュームのファイルシステムドライバースタックに送信されることに注意してください。ファイルシステムは、**IRP _ MJ _ create** 以外の操作の副作用としてストリームファイルオブジェクトを作成することが多いため、フィルタードライバーがストリームファイルオブジェクトの作成を確実に検出することは困難です。したがって、フィルタードライバーは、以前に表示されていないファイルオブジェクトに対して、**irp _ MJ _ CLEANUP** 要求と **irp _ MJ _ CLOSE** 要求を受け取ることを想定しています。

フィルタードライバーの作成者は、**IoCreateStreamFileObject** とは異なり、**IoCreateStreamFileMjTlite** では、**IRP _ の _ クリーンアップ** 要求がファイルシステムのドライバースタックに送信されないことに注意してください。このため、ファイルシステムでは、**IRP _ MJ _ create** 以外の操作の副作用としてストリームファイルオブジェクトを作成することが多いため、フィルタードライバーがストリームファイルオブジェクトの作成を確実に検出することは困難です。そのため、フィルタードライバーは、以前に表示されていないファイルオブジェクトに対して **IRP _ MJ _ CLOSE** 要求を受け取ることを想定しています。

パラメーター

ファイルシステムまたはフィルタードライバーは、指定された IRP で **IoGetLocationEntire** を呼び出して、irp 内の独自の **スタックの場所** へのポインターを取得します。次の一覧には、*irp* として示されています。(IRP は、*irp* として表示されます)。ドライバーは、終了要求を処理するときに、IRP の次のメンバーと IRP スタックの場所に設定されている情報を使用できます。

DeviceObject

ターゲットデバイスオブジェクトへのポインター。

Irp -> フラグ

この要求には、次のフラグが設定されています。

IRP _ CLOSE _ 操作

IRP _ 同期 _ API

Irpsp- > 最後の完了状態と要求された操作に関する情報を受け取る[IO _ 状態 _ ブロック](#)構造への `iostatus` ポインター。

Irpsp- > *DeviceObject*に関連付けられているファイルオブジェクトへの `FileObject` ポインター。

Irpsp- > *FileObject*パラメーターには、関連する[FileObject](#)フィールドへのポインターが含まれています。これは、ファイルオブジェクト構造でも _ あります。ファイルオブジェクト構造の関連性のある[fileobject](#)フィールド _ は、IRP MJ CLOSE の処理中は無効であり、_ 使用でき _ ません。

Irpsp- > *MajorFunction*は、IRP MJ CLOSE を指定し _ _ ます。

関連項目

[IO _ スタックの _ 場所](#)

[IO _ 状態 _ ブロック](#)

[IoCreateStreamFileObject](#)

[IoCreateStreamFileLite](#)

[IoGetLocation](#) [Enti](#)の場所

[IRP](#)

[IRP _ MJ _ CLOSE \(WDK カーネルリファレンス\)](#)

[IRP _ MJ の _ クリーンアップ](#)

[IRP _ MJ の _ 作成](#)

[ObDereferenceObject](#)

IRP _ MJ _ CREATE (IFS)

2020/08/04 • [Edit Online](#)

送信時

I/o マネージャーは、_ _ 新しいファイルまたはディレクトリが作成されているとき、または既存のファイル、デバイス、ディレクトリ、またはボリュームが開かれているときに、IRP MJ CREATE 要求を送信します。通常、この IRP は、[CreateFile](#)などの Microsoft Win32 関数を呼び出したユーザーモードアプリケーションの代わりに送信されるか、[iocreatefile](#)、[iocreatefilecreatedevice](#)、[Zwcreatefile](#)、または[zwcreatefile](#)と呼ばれるカーネルモードコンポーネントの代わりに送信されます。作成要求が正常に完了した場合、アプリケーションまたはカーネルモードコンポーネントは、ファイルオブジェクトへのハンドルを受け取ります。

操作: ファイルシステムドライバー

ターゲットデバイスオブジェクトがファイルシステムのコントロールデバイスオブジェクトである場合、ファイルシステムドライバーのディスパッチルーチンは、irp を完了し、適切な値に設定した後、適切な NTSTATUS*値を返す必要 > *があります。* > *

それ以外の場合、ファイルシステムドライバーは作成要求を処理する必要があります。

操作: ファイルシステムフィルタードライバー

ターゲットデバイスオブジェクトがフィルタードライバーのコントロールデバイスオブジェクトである場合、フィルタードライバーのディスパッチルーチンは、irp を完了し、適切な値に設定した後、適切な NTSTATUS*値を返す必要 > *があります。* > *

それ以外の場合、フィルタードライバーは必要な処理を実行し、フィルターの性質に応じて、IRP を完了するか、スタック上の次の下位のドライバーに渡します。

一般に、フィルタードライバーは、IRP _ MJ _ CREATEに応答して** _ 保留状態**を返すことはできません。ただし、下位レベルのドライバーから** _ 保留状態**が返された場合、フィルタードライバーはこの状態値をドライバーチェーンに渡す必要があります。

ファイルシステムフィルタードライバーの作成者は、[IoCreateStreamFileObject](#)によってIRP _ MJ _ CLEANUP要求がボリュームのファイルシステムドライバースタックに送信されることに注意してください。ファイルシステムは、IRP _ MJ _ create以外の操作の副作用としてストリームファイルオブジェクトを作成することが多いため、フィルタードライバーがストリームファイルオブジェクトの作成を確実に検出することは困難です。したがって、フィルタードライバーは、以前に表示されていないファイルオブジェクトに対して、irp _ MJ _ CLEANUP要求と[irp _ MJ _ CLOSE](#)要求を受け取ることを想定しています。

[IoCreateStreamFileMJ Tlite](#)の場合、IRP _ _ CLEANUP要求は送信されません。

NOTE

レガシフィルタードライバーが作成後のコールバックで作成を再発行する場合は、その再解析ポイント (補助バッファ) に関連付けられているバッファをNULLに設定する必要があります。レガシフィルタードライバーがこのバッファを解放せず、NULLに設定した場合、ドライバーはメモリをリークします。フィルタマネージャはこれを実行するため、ミニフィルタドライバはこれを行う必要はありません。

パラメーター

ファイルシステムまたはフィルタードライバーは、指定された IRP で[IoGetLocation](#) *enti*を呼び出して、irp 内の独自の[スタックの場所](#)へのポインターを取得します。次の一覧には、*irp*spとして示されています。(IRP は、*irp*として表示されます)。ドライバーは、create 要求を処理するときに、IRP の次のメンバーと IRP スタックの場所に設定されている情報を使用できます。

DeviceObject

ターゲットデバイスオブジェクトへのポインター。

Irp -> *AssociatedIrp.SystemBuffer*

ファイルオブジェクトが拡張属性を持つファイルを表す場合は、ファイルの** [_ 完全な _ EA _ 情報](#) **を指すポインターです。それ以外の場合、このメンバーはNULLに設定されます。

Irp -> [フラグ](#)

この要求には、次のフラグが設定されています。

IRP の [_ 作成 _ 操作](#)

IRP [_ 遅延 _ IO _ 完了](#)

IRP [_ 同期 _ API](#)

Irp -> *Irp->requestormode*は、操作を要求したプロセスの実行モード (**kernelmode** または **モード**) を示します。SL [_ FORCE _ アクセス _ チェック](#)フラグが設定されている場合は、* > *irp->requestormode**が **kernelmode** での場合でも、アクセスチェックを実行する必要があることに注意してください。

Irp -> 最後の完了状態と要求された操作に関する情報を受け取る[IO \[_ 状態 _ ブロック\]\(#\)](#)構造への *iostatus* ポインター。この構造体の情報メンバーは、ファイルシステムによって次のいずれかの値に設定されます。

[_作成されたファイル](#)

ファイル [_](#) が [_](#) 存在しません

ファイル [_](#) が存在します

開かれたファイル [_](#)

上書きされたファイル [_](#)

ファイルの [_](#) 置き換え

Irp -> ファイルのサイズの初期割り当てサイズ (バイト単位)。0以外の値は、ファイルが作成、上書き、または置き換えられない限り、無効です。

Irpsp -> 作成または開くファイルを表すために I/O マネージャーによって作成されるファイルオブジェクトへの *FileObject* ポインター。ファイルシステムは、IRP MJ [CREATE](#) 要求を処理するときに、[_](#) [_](#) このファイルオブジェクトの *fscontext* および場合によっては *FsContext2* フィールドを、ファイルシステム固有の値に設定します。このため、ファイルシステムによって [create](#) 要求が処理されるまで、*Fscontext* フィールドと *FsContext2* フィールドの値を有効と見なすことはできません。詳細については、「[ファイルストリーム、ストリームコンテキスト、およびストリームごとのコンテキスト](#)」を参照してください。

[Fltcancelfileopen](#) および [iocancelfileopen](#) は、[_](#) [_](#) [_](#) ファイルオブジェクトの *Flags* フィールドに、FO [ファイルオープンキャンセル](#) フラグを設定します。このフラグを設定すると、IRP [_](#) MJ [_](#) [CREATE](#) 要求が取り消され、このファイルオブジェクトに対して [irp \[_\]\(#\) MJ \[_\]\(#\) CLOSE](#) 要求が発行されます。作成要求がキャンセルされると、再発行することはできません。

Irpsp -> *FileObject* パラメーターには、関連する *FileObject* フィールドへのポインターが含まれています。これは、ファイルオブジェクト構造でも [_](#) あります。ファイル *RelatedFileObject* [_](#) オブジェクト構造の関連する *fileobject* フィールドは、既に開いているファイルオブジェクトを基準として、特定のファイルが開かれていることを示すために使用されます。これは通常、相対ファイルがディレクトリであることを示していますが、ストリームベースのファイルは、既にファイルのストリームを使用して開いている可能性があります。ファイルオブジェクト構造の関連性のある *fileobject* フィールド [_](#) は、IRP MJ [CREATE](#) の処理中にのみ有効です [_](#) [_](#) 。

Irpsp -> 次の1つ以上にフラグを追加します。

_	_

III	II
SL_CASE_SENSITIVE	このフラグが設定されている場合、ファイル名の比較では大文字と小文字が区別されます。
SL_FORCE_ACCESS_CHECK	このフラグが設定されている場合、> <i>irp->requestormode</i> の値がkernelmode 1の場合でもアクセスチェックを実行する必要があります。
SL_OPEN_PAGING_FILE	このフラグが設定されている場合、ファイルはページングファイルになります。
SL_OPEN_TARGET_DIRECTORY	このフラグが設定されている場合は、ファイルの親ディレクトリを開く必要があります。

*Irpsp- >MajorFunction*は、IRP MJ CREATE を指定し __ ます。

*Irpsp- >EaLength * > tembufferAssociatedIrp.Sys**で、バッファのサイズ (バイト単位) を作成します。 *Irpsp- >AssociatedIrp.Systembuffer*の値がNULLの場合、このメンバーは0である必要があります。

Irpsp- >Parameters. ファイルを作成または開くときに適用される属性フラグの fileattributes ビットマスク。明示的に指定された属性は、ファイルが作成、置き換えられた場合、または上書きされた場合にのみ適用されます。既定では、この値はファイル属性 NORMAL です。これは、__ 他のフラグまたは互換性のあるフラグの論理和の組み合わせによってオーバーライドできます。このメンバーは、**loccreatefil**を指定する *fileattributes* パラメーターに対応します。

Irpsp- >Parameters. 作成またはファイルを開くときに適用されるオプションを指定するフラグのオプションビットマスク。ファイルが既に存在する場合に実行されるアクションも指定します。

このパラメーターの上位8ビットは、**loccreatefil**指定されたデバイスの処理/パラメーターに対応します。

このメンバーの下位24ビットは、*Createoptions*パラメーターに対応して**loccreatefil**を指定します。ファイルのスキャンを実行するファイルシステムフィルターおよびミニフィルタードライバ (ウイルス対策プログラムなど) では、ファイルが OPLOCKED フラグの場合に特に注意する必要があります __ ます。このフラグが設定されている場合、フィルターでは、IRP _ MJ の作成操作をブロックしたり、遅延させたりすることはできません __ 。

__ __ プリ作成 (ディスパッチ) パスに OPLOCKED フラグが設定されている場合、ファイルが完了すると、次のいずれかの種類の操作を開始できません。これは、oplock が解除される可能性があるためです。

IRP _ MJ _ CLEANUP irp _ MJ _ CREATE irp _ MJ _ FILE _ SYSTEM _ CONTROL IRP _ MJ _ FLUSH buffer _ IRP _ MJ _ LOCK _ CONTROL irp _ MJ _ READ irp _ MJ _ SET _ INFORMATION irp _ MJ _ WRITE フィルターまたはミニフィルターでファイルの完了を受け入れることができない場合 __ __ 、OPLOCKED フラグの場合、irp MJ CREATE 要求を完了する必要があります __ __ ます状態共有 __ 違反。

[__ __ 完了 (作成後)] パスに OPLOCKED フラグが設定されている場合、ファイルが完了すると、ファイルシステムが *Irpsp- > iostatus.status* をステータス OPLOCK [__ __ __ 進行中の状態] の値に設定しているかどうかを確認する必要があります。この状態値が設定されていない場合は、フィルターがファイルに対して上記の操作のいずれかを開始することが安全です。この状態値が設定されている場合、oplock はまだ解除されていないため、oplock 解除を引き起こす可能性がある操作をフィルターで開始することはできません。したがって、次のいずれかの条件が満たされるまで、ファイルに対する上記のすべての操作をフィルターで延期する必要があります。

- Oplock の所有者は、_ ファイルシステムに FSCTL oplock ブレーク確認要求を送信し __ ます。
- フィルターまたはミニフィルター以外のシステムコンポーネントは、oplock の解除が完了するまで待機する必要がある i/o 要求 (IRP _ MJ _ READ や irp MJ WRITE など) をファイルシステムに送信し __ ます。フィルターまたはミニフィルターは、この新しい操作のディスパッチ (または事前操作コールバック) ルーチンから上記の操作のいずれかを開始できます。これは、ディ

スパッチまたは preoperation コールバックルーチンが、oplock の解除が完了するまで待機状態になるためです。

*Irps- >SecurityContext- > *オブジェクトのサブジェクトコンテキスト、アクセスの種類、およびその他の必要なアクセスの種類を含む[アクセス _ 状態](#) 構造へのポインター。

*Irps- >SecurityContext- > DesiredAccess*アクセス _ マスク構造を指定して、ファイルに要求されたアクセス権を指定します。詳細については、[IoCreateFileDesiredAccess](#) パラメーターの説明を参照してください。 *DesiredAccess*

*Irps- >*パラメーター。ファイルに要求された共有アクセス権のアクセスビットマスクを作成します。このメンバーがゼロの場合は、排他アクセスが要求されます。詳細については、[IoCreateFile](#)指定している場合は、このパラメーターの説明を参照してください。

関連項目

[アクセス _ マスク](#)

[アクセスの _ 状態](#)

[_EA の完全な _ _ 情報](#)

[FltCancelFileOpen](#)

[FltReissueSynchronousIo](#)

[IO _ スタックの _ 場所](#)

[IO _ 状態 _ ブロック](#)

[IoCancelFileOpen](#)

[IoCheckEaBufferValidity](#)

[IoCreateFile](#)

[IoCreateFile](#)の場合は、デバイスを IoInt にします。

[IoCreateStreamFileObject](#)

[IoCreateStreamFileObject TLite](#)

[IoGetLocation](#) Entiの場所

[IRP](#)

[IRP _ MJ の _ クリーンアップ](#)

[IRP _ MJ _ CLOSE](#)

[IRP _ MJ _ CREATE \(WDK カーネルリファレンス\)](#)

[ZwCreateFile](#)

[ZwOpenFile](#)

IRP_MJ_CREATE_MAILSLLOT

2020/04/24 • [Edit Online](#)

送信時

I/O マネージャーは、新しいメールスロットを作成または開いたときに、IRP_MJ_CREATE_MAILSLLOT 要求を送信します。通常、この IRP は次のように送信されます。

- [Createmailslot](#)などの Microsoft Win32 関数を呼び出したユーザーモードアプリケーションに代わって。
- または、[locreatefile](#)または[iocreatefile](#)の場合は、[iodevicethint](#)というカーネルモードコンポーネントの代わりに。

メールスロット作成要求が正常に完了した場合、アプリケーションまたはカーネルモードコンポーネントは、メールスロットファイルインスタンスへのハンドルを受け取ります。

IRP_MJ_CREATE_MAILSLLOT の処理は[IRP_MJ_CREATE](#)とほぼ同じです。

操作: ファイルシステムドライバ

ターゲットデバイスオブジェクトがファイルシステムのコントロールデバイスオブジェクトである場合、*irp > IoStatus.Status*および*irp->*を設定した後に、ファイルシステムドライバのディスパッチルーチンが *irp* を完了し、適切な NTSTATUS 値を返す必要があります。*IoStatus* 情報に適切な値を指定します。

それ以外の場合、ファイルシステムドライバは作成要求を処理する必要があります。

操作: ファイルシステムレガシフィルタードライバ

ターゲットデバイスオブジェクトがレガシフィルタードライバのコントロールデバイスオブジェクトである場合、*irp > iostatus.Status*と*irp->*を設定した後で、そのドライバのディスパッチルーチンが *irp* を完了し、適切な NTSTATUS 値を返す必要があります。*IoStatus* 情報に適切な値を指定します。

それ以外の場合、レガシフィルタードライバは必要な処理を実行し、フィルタの性質に応じて、IRP を完了するか、スタック上の次の下位のドライバに渡します。

一般に、レガシフィルタードライバは、IRP_MJ_CREATE_MAILSLLOTへの応答としてSTATUS_PENDINGを返さないようにする必要があります。ただし、下位レベルのドライバからSTATUS_PENDINGが返された場合、レガシフィルタードライバはこの状態値をドライバチェーンに渡す必要があります。

パラメーター

ファイルシステムまたはレガシフィルタードライバは、指定された IRP で[iogetlocation entitu](#)を呼び出して、*irp* 内の独自の[スタック位置](#)へのポインターを取得します。次のパラメーターでは、*irp*が *Irps*と*irpsp*をポイントして、IO_STACK_LOCATION を指します。ドライバは、create 要求を処理するときに、IRP の次のメンバーと IRP スタックの場所に設定されている情報を使用できます。

- *DeviceObject*は、ターゲットデバイスオブジェクトへのポインターです。
- *Irps > Flags*は、この要求に対して次のフラグに設定されます。
 - IRP_CREATE_OPERATION
 - IRP_DEFER_IO_COMPLETION
 - IRP_SYNCHRONOUS_API
- *Irps > IoStatus*は、最終的な完了状態と要求された操作に関する情報を受け取るIO_STATUS_BLOCK構造を指します。この構造体の情報メンバーは、ファイルシステムによって次のいずれかの値に設定されます。

- FILE_CREATED
 - FILE_OPENED
 - *Irps-> Irp->requestormode*操作を要求したプロセスの実行モード (*kernelmode* でまたはモード) を示します。SL_FORCE_ACCESS_CHECK フラグが設定されている場合、> *Irps->requestormode*が*kernelmode* の場合でも、アクセスチェックを実行する必要があることに注意してください。
 - *Irps-> MajorFunction*は IRP_MJ_CREATE_MAILSLOT に設定されます。
 - *Irps-> フラグ*を SL_FORCE_ACCESS_CHECK に設定できます。このフラグが設定されている場合は、*Irps-> Irp->requestormode*の値が*kernelmode* の場合でもアクセスチェックを実行する必要があります。
 - *Irps-> SecurityContext-> AccessState*は、オブジェクトのサブジェクトコンテキスト、許可されたアクセスの種類、およびその他の必要なアクセスの種類を含むACCESS_STATE構造体へのポインターです。
 - *Irps-> Parameters SecurityContext-> DesiredAccess*は、メールスロットに要求されたアクセス権を指定するACCESS_MASK 構造です。詳細については、[IoCreateFileDesiredAccess](#) パラメーターの説明を参照してください。
 - *Irps-> Parameters. CreateMailslot. オプション*は、メールスロットを作成または開くときに適用されるオプションを指定するフラグのビットマスクです。また、メールスロットが既に存在する場合に実行される操作も指定します。
- このパラメーターの上位8ビットは、[IoCreateFile](#)指定されたデバイスの処理パラメーターに対応します。
- このメンバーの下位24ビットは、*CreateOptions*パラメーターに対応して[IoCreateFile](#)を指定します。
- *Irps-> パラメーター. CreateMailslot. 共有アクセス*は、メールスロットに要求された共有アクセス権のビットマスクです。このメンバーがゼロの場合は、排他アクセスが要求されます。詳細については、[IoCreateFile](#)指定している場合は、このパラメーターの説明を参照してください。
 - *Irps-> パラメーター. CreateMailslot. parameters*は、メールスロットを作成するときの CREATE パラメーターを含むMAILSLOT_CREATE_PARAMETERS構造体へのポインターです。
 - *Irps-> FileObject*は、作成または開くメールスロットを表すために i/o マネージャーによって作成されるファイルオブジェクトへのポインターです。ファイルシステムが IRP_MJ_CREATE_MAILSLOT 要求を処理すると、このファイルオブジェクトのFscontextとFsContext2フィールドが、ファイルシステム固有の値に設定されます。このため、ファイルシステムによって create 要求が処理されるまで、FscontextフィールドとFsContext2フィールドの値を有効と見なすことはできません。詳細については、「[ファイルストリーム、ストリームコンテキスト、およびストリームごとのコンテキスト](#)」を参照してください。

[IoCancelFileOpen](#) (および[fltCancelFileOpen](#)) は、ファイルオブジェクトのFlagsフィールドにFO_FILE_OPEN_CANCELLED フラグを設定します。このフラグを設定すると、IRP_MJ_CREATE_MAILSLOT 要求が取り消され、このファイルオブジェクトに対してIRP_MJ_CLOSE要求が発行されることを示します。作成要求がキャンセルされると、再発行することはできません。

*Irps-> FileObject*パラメーターには、FILE_OBJECT 構造である、関連するfileobjectフィールドへのポインターが含まれています。FILE_OBJECT 構造体の関連するfileobjectフィールドは、既に開いているファイルオブジェクトに対して、特定のメールスロットが開かれていることを示すために使用されます。これは通常、相対ファイルがディレクトリであることを示していますが、ストリームベースのファイルは、既にファイルのストリームを使用して開いている可能性があります。FILE_OBJECT 構造体の関連性のあるフィールドは、IRP_MJ_CREATE_MAILSLOTの処理中にのみ有効です。

関連項目

[ACCESS_MASK](#)

[ACCESS_STATE](#)

[FLT_PARAMETERS](#)

[IRP_MJ_CREATE_MAILSLOT の FLT_PARAMETERS](#)

FltCancelFileOpen

IO_STACK_LOCATION

IO_STATUS_BLOCK

IoCancelFileOpen

IoCreateFile

IoCreateFileの場合、デバイスを Ioint にします。

IoCreateStreamFileObject

IoCreateStreamFileObject Tlite

IoGetLocation Entiの場所

IRP

IRP_MJ_CLEANUP

IRP_MJ_CLOSE

IRP_MJ_CREATE (WDK カーネルリファレンス)

MAILSLOT_CREATE_PARAMETERS

IRP_MJ_CREATE_NAMED_PIPE

2020/04/24 • [Edit Online](#)

送信時

I/O マネージャーは、新しい名前付きパイプを作成または開いているときに、IRP_MJ_CREATE_NAMED_PIPE 要求を送信します。通常、この IRP は次のように送信されます。

- [CreateNamedPipe](#)などの Microsoft Win32 関数を呼び出したユーザーモードアプリケーションに代わって。
- または、[IoCreateFile](#)または[IoCreateFileEx](#)の場合は、[IoDeviceHint](#)というカーネルモードコンポーネントの代わりに。

名前付きパイプの作成要求が正常に完了した場合、アプリケーションまたはカーネルモードコンポーネントは、名前付きパイプファイルインスタンスのサーバー終端へのハンドルを受け取ります。

IRP_MJ_CREATE_NAMED_PIPE の処理は[IRP_MJ_CREATE](#)とほぼ同じです。

操作: ファイルシステムドライバー

ターゲットデバイスオブジェクトがファイルシステムのコントロールデバイスオブジェクトである場合、*irp > IoStatus.Status*および*irp->IoStatus.Information*を設定した後、ファイルシステムドライバーのディスパッチルーチンが *irp* を完了し、適切な NTSTATUS 値を返す必要があります。*IoStatus.Information* に適切な値を指定します。

それ以外の場合、ファイルシステムドライバーは作成要求を処理する必要があります。

操作: ファイルシステムレガシフィルタードライバー

ターゲットデバイスオブジェクトがレガシフィルタードライバーのコントロールデバイスオブジェクトである場合、*irp > IoStatus.Status*と*irp->IoStatus.Information*を設定した後で、そのドライバーのディスパッチルーチンが *irp* を完了し、適切な NTSTATUS 値を返す必要があります。*IoStatus.Information* に適切な値を指定します。

それ以外の場合、レガシフィルタードライバーは必要な処理を実行し、フィルターの性質に応じて、IRP を完了するか、スタック上の次の下位のドライバーに渡します。

一般に、レガシフィルタードライバーは、IRP_MJ_CREATE_NAMED_PIPE への応答としてSTATUS_PENDINGを返さないようにする必要があります。ただし、下位レベルのドライバーからSTATUS_PENDINGが返された場合、レガシフィルタードライバーはこの状態値をドライバチェーンに渡す必要があります。

パラメーター

ファイルシステムまたはレガシフィルタードライバーは、指定された IRP で[IoGetLocation](#)を呼び出して、*irp* 内の独自の[IoStackLocation](#)へのポインターを取得します。次のパラメーターでは、*irp*が *Irp* と *Irpsp* をポイントして、IO_STACK_LOCATION を指します。ドライバーは、create 要求の処理中に、IRP の次のメンバーと IRP スタックの場所に設定されている情報を使用できます。

- *DeviceObject*は、ターゲットデバイスオブジェクトへのポインターです。
- *Irpsp* フラグは、この要求に対して次のフラグに設定されます。
 - IRP_CREATE_OPERATION
 - IRP_DEFER_IO_COMPLETION
 - IRP_SYNCHRONOUS_API
- *Irpsp* の *IoStatus*は、最終的な完了状態と要求された操作に関する情報を受け取るIO_STATUS_BLOCK構造を指します。この構造体の情報メンバーは、ファイルシステムによって次のいずれかの値に設定されます。

- FILE_CREATED
- FILE_OPENED
- *Irps-> Irp->requestormode*操作を要求したプロセスの実行モード (**kernelmode** でまたはモード) を示します。SL_FORCE_ACCESS_CHECK フラグが設定されている場合、> *Irps->requestormode*が**kernelmode** の場合でも、アクセスチェックを実行する必要があることに注意してください。
- *Irps-> MajorFunction*は IRP_MJ_CREATE_NAMED_PIPE に設定されます。
 - *Irps-> フラグ*を SL_FORCE_ACCESS_CHECK に設定できます。このフラグが設定されている場合は、*Irps-> Irp->requestormode*の値が**kernelmode** の場合でもアクセスチェックを実行する必要があります。
- *Irps-> SecurityContext-> AccessState*は、オブジェクトのサブジェクトコンテキスト、アクセスの種類、およびその他の必要なアクセスの種類を含む[ACCESS_STATE](#)構造体へのポインターです。
- *Irps-> SecurityContext-> DesiredAccess*は、名前付きパイプに要求されたアクセス権を指定する ACCESS_MASK 構造です。詳細については、[IoCreateFileDesiredAccess](#)パラメーターの説明を参照してください。
- *Irps-> Parameters. CreatePipe. オプション*は、名前付きパイプを作成または開くときに適用されるオプションを指定するフラグのビットマスクです。また、名前付きパイプが既に存在する場合に実行されるアクションも指定します。

このパラメーターの上位8ビットは、[IoCreateFile](#)指定されたデバイスの**処理**パラメーターに対応します。

このメンバーの下位24ビットは、*CreateOptions*パラメーターに対応して[IoCreateFile](#)を指定します。

- *Irps-> Parameters. CreatePipe. 名前付きパイプ*に要求された共有アクセス権のビットマスクを指定します。このメンバーがゼロの場合は、排他アクセスが要求されます。詳細については、[IoCreateFile](#)指定している場合は、このパラメーターの説明を参照してください。
- *Irps-> パラメーター. CreatePipe. パラメーター*は、名前付きパイプが作成されているときの CREATE パラメーターを含む[NAMED_PIPE_CREATE_PARAMETERS](#)構造体へのポインターです。
- *Irps-> FileObject*作成または開く名前付きパイプを表すために i/o マネージャーによって作成されるファイルオブジェクトへのポインター。ファイルシステムが IRP_MJ_CREATE_NAMED_PIPE 要求を処理すると、このファイルオブジェクトのFscontextとFsContext2フィールドが、ファイルシステム固有の値に設定されます。このため、ファイルシステムによって create 要求が処理されるまで、FscontextフィールドとFsContext2フィールドの値を有効と見なすことはできません。詳細については、「[ファイルストリーム、ストリームコンテキスト、およびストリームごとのコンテキスト](#)」を参照してください。

[IoCancelFileOpen](#) (および[fltCancelFileOpen](#)) は、ファイルオブジェクトのFlagsフィールドに FO_FILE_OPEN_CANCELLED フラグを設定します。このフラグを設定すると、IRP_MJ_CREATE_NAMED_PIPE 要求が取り消され、このファイルオブジェクトに対して[IRP_MJ_CLOSE](#)要求が発行されることを示します。作成要求がキャンセルされると、再発行することはできません。

*Irps-> FileObject*パラメーターには、FILE_OBJECT 構造である、関連するfileobjectフィールドへのポインターが含まれています。FILE_OBJECT 構造体の関連するfileobjectフィールドは、既に開いているファイルオブジェクトに対して特定の**名前付きパイプが開かれていることを示すために使用されます**。これは通常、**相対ファイルがディレクトリであることを示しています**が、ストリームベースのファイルは、既にファイルのストリームを使用して開いている可能性があります。FILE_OBJECT 構造体の**関連性のあるフィールドは、IRP_MJ_CREATE_NAMED_PIPEの処理中にのみ有効です**。

関連項目

[ACCESS_MASK](#)

[ACCESS_STATE](#)

[FLT_PARAMETERS](#)

[IRP_MJ_CREATE_NAMED_PIPE の FLT_PARAMETERS](#)

[FtCancelFileOpen](#)

IO_STACK_LOCATION

IO_STATUS_BLOCK

IoCancelFileOpen

IoCreateFile

IoCreateFileの場合、デバイスを IoDevice にします。

IoCreateStreamFileObject

IoCreateStreamFileObject TlFile

IoGetLocation Entiの場所

IRP

IRP_MJ_CLEANUP

IRP_MJ_CLOSE

IRP_MJ_CREATE (WDK カーネルリファレンス)

NAMED_PIPE_CREATE_PARAMETERS

IRP _ MJ _ DEVICE _ CONTROL (IFS)

2020/08/04 • [Edit Online](#)

送信時

IRP _ MJ _ デバイス _ 制御要求は、i/o マネージャーおよびその他のカーネルモードドライバによって送信されます。通常、この IRP は、Microsoft Win32 [DeviceIoControl](#) 関数を呼び出したユーザーモードアプリケーションの代わりに、または [ZwDeviceIoControlFile](#) を呼び出したカーネルモードコンポーネントの代わりに送信されます。

操作: ファイルシステムドライバ

ファイルシステムドライバは、ボリュームを開いているハンドルで要求が発行されているかどうかを判断するために、ファイルオブジェクトを抽出してデコードする必要があります。この場合は、ファイルシステムドライバが IRP を、ボリュームがマウントされている記憶装置のデバイスドライバに渡す必要があります。それ以外の場合、ドライバは IRP を失敗させる必要があります。

操作: ファイルシステムフィルタードライバ

フィルタードライバは必要な処理を実行する必要があります。フィルタの性質によっては、IRP を完了するか、スタック上の次の下位のドライバに渡します。

パラメーター

ファイルシステムまたはフィルタードライバは、指定された IRP に対して [IoGetLocation](#) を呼び出して、irp 内の独自の [StackLocation](#) へのポインターを取得します。次の一覧に *irpSp* として示されています。(IRP は、*irp* として表示されます)。ドライバは、デバイス制御要求を処理するときに、IRP の次のメンバーと IRP スタックの場所に設定されている情報を使用できます。

DeviceObject

ターゲットデバイスオブジェクトへのポインター。

Irpb->AssociatedIrp.SystemBuffer

ターゲットデバイスのデバイスドライバに渡されるシステム指定の入力バッファへのポインター。メソッドのバッファまたはメソッドの直接 i/o に使用されます。このパラメーターが必須かどうかは、特定の i/o 制御コードによって異なります。

Irpb->IoStatus

最後の完了状態と要求された操作に関する情報を受け取る [IO _ STATUS _ BLOCK](#) 構造体へのポインター。詳細については、[ZwDeviceIoControlFile](#) の *IoStatusBlock* パラメーターの説明を参照してください。

Irpb->MdlAddress

ターゲットデバイスのデバイスドライバに渡される出力バッファを記述するメモリ記述子リスト (MDL) のアドレス。メソッドダイレクト i/o に使用されます。このパラメーターが必須かどうかは、特定の i/o 制御コードによって異なります。

Irpb->Irpb->RequestorMode

操作を要求したプロセスの実行モード (*kernelmode* でまたはモード) を示します。

Irpb->UserBuffer

ターゲットデバイスのデバイスドライバに渡される、呼び出し元から提供される出力バッファへのポインター。メソッドのバッファに使用されるか、または i/o ではないメソッドに使用されます。このパラメーターが省略可能か必須かは、特定の i/o 制御コードによって決まります。

Irpb->FileObject

DeviceObject に関連付けられているファイルオブジェクトへのポインター。

Irpb->FileObject パラメーターには、関連する *FileObject* フィールドへのポインターが含まれています。これは、ファイルオブジェ

クト構造でも _ あります。ファイルオブジェクト構造の "関連性のあるfileobject " フィールド _ は、IRP MJ デバイスコントロールの処理中は無効であり、_ _ _ 使用しないでください。

IrpSp- > MajorFunction

IRP _ MJ デバイスコントロールを指定し _ _ ます。

IrpSp- > DeviceIoControl. InputBufferLength

*Irp > AssociatedIrp.Systembuffer*が指すバッファのサイズ (バイト単位)。

IrpSp- > DeviceIoControl. IoControlCode

ターゲットデバイスのデバイスドライバに渡される IOCTL 関数コード。

IOCTL 要求の詳細については、「カーネルモードアーキテクチャガイド」の「[i/o 制御コードの使用](#)」および Microsoft Windows SDK のドキュメントの「デバイスの入力と出力の制御コード」を参照してください。

IrpSp- > DeviceIoControl の長さ

* > UserBuffer*によってポイントされるバッファのサイズ (バイト単位)。

IrpSp- > DeviceIoControl. Type3InputBuffer

メソッドを使用するカーネルモード要求の入力バッファ (_ どちらも使用しません)。

関連項目

[IO _ スタックの _ 場所](#)

[IO _ 状態 _ ブロック](#)

[IoBuildDeviceIoControlRequest](#)

[IoGetLocation](#) Entiの場所

[IoGetFunctionCodeFromCtlCode](#)

[IRP](#)

[IRP _ MJ _ デバイス _ コントロール \(WDK カーネルリファレンス\)](#)

[I/O 制御コードの使用](#)

[ZwDeviceIoControlFile](#)

IRP_MJ_DIRECTORY_CONTROL (IFS)

2020/08/04 • [Edit Online](#)

送信時

IRP_MJ_DIRECTORY_CONTROL 要求は、i/o マネージャーおよびその他のカーネルモードドライバによって送信されます。たとえば、ユーザーモードアプリケーションが `ReaddirectoryFindNextVolumeMountPoint` や `FindNextVolumeMountPoint` などの Microsoft Win32 関数を呼び出したときや、カーネルモードコンポーネントが `zwquerydirectoryfile` を呼び出したときなどに送信できます。

操作: ファイルシステムドライバ

ファイルシステムドライバは、マイナー関数のコードを確認して、どのディレクトリ制御操作が要求されているかを判断する必要があります。有効なマイナー関数コードを次に示します。

IRP_MN	説明
IRP_MN_NOTIFY_CHANGE_DIRECTORY	ディレクトリへの変更の通知要求を示します。通常、この要求をすぐに満たすのではなく、ファイルシステムドライバが IRP をプライベートキューに保持します。ディレクトリに変更が行われると、ファイルシステムドライバは通知を実行し、デキューを実行して IRP を完了します。
IRP_MN_QUERY_DIRECTORY	ディレクトリクエリ要求を示します。照会できる情報の種類はファイルシステムによって異なりますが、一般的には次のものが含まれます。 <code>FileFullDirectoryInformation</code> 、 <code>DirectoryInformation</code> 、 <code>FileDirectoryInformation</code> 、 <code>Filebothdirectoryinformation</code> 、 <code>FileBothDirectoryInformation</code> 、 <code>FileNamesInformation</code> 、 <code>FileObjectIdInformation</code> 、 <code>FileReparsePointInformation</code>

NOTE

`FileQuotaInformation` クラスは互換性のために残されています。代わりに `IRP_MJ_QUERY_QUOTA` を使用する必要があります。

要求された操作を実行すると、ファイルシステムドライバが IRP を完了する必要があります。

操作: ファイルシステムフィルタードライバ

フィルタードライバは、この IRP をスタック上の次の下位のドライバに渡す必要があります。

パラメーター

ファイルシステムまたはフィルタードライバは、指定された IRP で `IoGetLocation` を呼び出して、irp 内の独自の **スタックの場所** へのポインターを取得します。次の一覧には、`irp->IoStatus` として示されています。(IRP は、`irp` として表示されます)。ドライバは、「ディレクトリ制御要求の処理」で、IRP の次のメンバーと IRP スタックの場所に設定されている情報を使用できます。

DeviceObject

ターゲットデバイスオブジェクトへのポインター。

Irql > IoStatus

最終的な完了ステータスと要求された操作に関する情報を受け取る `IO_STATUS_BLOCK` 構造体へのポインター。

Irp->UserBuffer

ディレクトリの内容に関する要求された情報を受け取る、呼び出し元から提供される出力バッファへのポインター。

IrpSp->FileObject

*DeviceObject*に関連付けられているファイルオブジェクトへのポインター。

*IrpSp->FileObject*パラメーターには、FILE_OBJECT 構造である、関連するfileobjectフィールドへのポインターが含まれています。FILE_OBJECT 構造体の、IRP_MJ_DIRECTORY_CONTROLの処理中には無効なので、使用することはできません。

IrpSp->フラグ

IRP_MN_QUERY_DIRECTORY には、次のフラグを設定できます。

III	II
SL_INDEX_SPECIFIED	<i>IrpSp->Parameters. QueryDirectory. FileIndex</i> によってインデックスが指定されているディレクトリのエントリでスキャンを開始します。
SL_RESTART_SCAN	ディレクトリの最初のエントリでスキャンを開始します。このフラグが設定されていない場合は、以前の IRP_MN_QUERY_DIRECTORY 要求からスキャンを再開します。
SL_RETURN_SINGLE_ENTRY	最初に見つかったエントリだけを返します。
SL_RETURN_ON_DISK_ENTRIES_ONLY	ディレクトリの仮想化またはジャストインタイム拡張を実行するすべてのフィルターに対して、要求をファイルシステムに渡し、現在ディスク上にあるエントリを返すように指示します。

IRP_MN_NOTIFY_CHANGE_DIRECTORY には、次のフラグを設定できます。

III	II
SL_WATCH_TREE	このディレクトリのすべてのサブディレクトリも監視する必要がある場合は、TRUEに設定します。ディレクトリ自体だけを監視する場合は、FALSEに設定します。

IrpSp->MajorFunction

IRP_MJ_DIRECTORY_CONTROL を指定します。

IrpSp->MinorFunction

次のいずれか:

- IRP_MN_NOTIFY_CHANGE_DIRECTORY
- IRP_MN_QUERY_DIRECTORY

IrpSp->Parameters. NotifyDirectory. このフィルター

詳細については、[Fsrtlnotifyfullchangedirectory](#)への参照 *フィルターパラメーターの説明*を参照してください。

IrpSp->Parameters. NotifyDirectory. Length

*Irp >UserBuffer*が指すバッファの長さ (バイト単位)。

IrpSp->Parameters. QueryDirectory. FileIndex

ディレクトリスキャンを開始するファイルのインデックス。SL_INDEX_SPECIFIED フラグが設定されていない場合は無視されます。このパラメーターは、どの Win32 関数またはカーネルモードサポータールーチンでも指定できません。現在、このファイルは、32ビットの NT ベースのプラットフォームのみに存在する NT 仮想 DOS コンピューター (NTVDM) によってのみ使用されます。ファイルインデックスは NTFS などのファイルシステムに対して定義されていないことに注意してください。これは、親ディレクトリ内のファイルの位置が固定されておらず、並べ替え順序を維持するためにいつでも変更できることに注意してください。

IrpSp->Parameters. QueryDirectory. FileInformationClass

次に示す値のいずれかを指定します。

I	II
■■■■ (& i)	各ファイルのFILE_BOTH_DIR_INFORMATION構造体を返します。
FileDirectoryInformation	各ファイルのFILE_DIRECTORY_INFORMATION構造体を返します。
FileFullDirectoryInformation	各ファイルのFILE_FULL_DIR_INFORMATION"構造体を返します。
FileIdBothDirectoryInformation	各ファイルのFILE_ID_BOTH_DIR_INFORMATION構造体を返します。
FileIdFullDirectoryInformation	各ファイルのFILE_ID_FULL_DIR_INFORMATION構造体を返します。
FileNamesInformation	各ファイルのFILE_NAMES_INFORMATION構造体を返します。
FileObjectIdInformation	各ファイルのFILE_OBJECTID_INFORMATION構造体を返します。
FileQuotaInformation	互換性のために残されています。代わりにIRP_MJ_QUERY_QUOTAを使用してください。
FileReparsePointInformation	ディレクトリの1つのFILE_REPARSE_POINT_INFORMATION構造体を返します。

IrpSp->Parameters. QueryDirectory. FileName
指定したディレクトリ内のファイルの名前 (省略可能)。

IrpSp->Parameters. QueryDirectory. Length
Irp > UserBufferが指すバッファの長さ (バイト単位)。

関連項目

[FILE_BOTH_DIR_INFORMATION](#)

[FILE_DIRECTORY_INFORMATION](#)

[FILE_FULL_DIR_INFORMATION](#)

[FILE_ID_BOTH_DIR_INFORMATION](#)

[FILE_ID_FULL_DIR_INFORMATION](#)

[FILE_NAMES_INFORMATION](#)

[FILE_OBJECTID_INFORMATION](#)

[FILE_REPARSE_POINT_INFORMATION](#)

[FsRtlNotifyFullChangeDirectory](#)

[IO_STACK_LOCATION](#)

[IO_STATUS_BLOCK](#)

IoGetLocationEntiの場所

IRP

IRP_MJ_QUERY_QUOTA

ZwQueryDirectoryFile

IRP_MJ_FILE_SYSTEM_CONTROL (IFS)

2020/08/04 • [Edit Online](#)

送信時

IRP_MJ_FILE_system_CONTROL 要求は、i/o マネージャーおよびその他のカーネルモードドライバによって送信されます。たとえば、ユーザーモードアプリケーションが Microsoft Win32 [DeviceIoControl](#) 関数を呼び出してファイルシステム i/o 制御 (FSCTL) 要求を送信した場合などに送信できます。

操作: ファイルシステムドライバー

ファイルシステムドライバーまたはレグナイザーは、マイナー関数のコードを確認して、どのファイルシステム制御操作が要求されているかを判断する必要があります。

ファイルシステムドライバーは、次のマイナー関数コードを処理する必要があります。

'''	''
IRP_MN_KERNEL_CALL	この要求は IRP_MN_USER_FS_REQUEST (次に説明) と同じですが、要求のソースは信頼されたカーネルコンポーネントである点が異なります。
IRP_MN_MOUNT_VOLUME	ボリュームマウント要求を示します。ファイルシステムドライバーがファイルシステムの形式と一致しないボリュームに対してこの IRP を受信した場合、ファイルシステムドライバーは STATUS_UNRECOGNIZED_VOLUME を返します。
IRP_MN_USER_FS_REQUEST	FSCTL 要求を示します。これは、Microsoft Win32 DeviceIoControl 関数を呼び出したユーザーモードアプリケーションに代わって、または ZwDeviceIoControlFile または IoBuildDeviceIoControlRequest を呼び出したカーネルモードコンポーネントの代理として発生する可能性があります。 FSCTL 要求の詳細については、Microsoft Windows SDK のドキュメントの「デバイスの入力と出力の制御コード」を参照してください。
IRP_MN_VERIFY_VOLUME	ボリューム検証要求を示します。リムーバブルメディアの場合、ファイルシステムは、メディアが削除され、ファイルシステムが以前に使用していたボリュームと同じボリュームであることを検出したときに、そのボリュームを確認する必要があります。ボリュームが変更されている場合、ファイルシステムは未処理のハンドルをすべて無効にする必要があります。また、この新しいメディアのファイルシステムが変更された場合にもエラーが返されます。この要求は、ほとんどの場合、フロッピードライブに使用されます。

ファイルシステムのレグナイザーは、次のマイナー関数コードを処理する必要があります。

III	II
IRP_MN_LOAD_FILE_SYSTEM	ファイルシステム要求の読み込みを示します。

要求された操作を実行した後、ファイルシステムドライバーまたはレグナイザーが IRP を完了する必要があります。

操作: ファイルシステムフィルタードライバー

フィルタードライバーは、この IRP をスタック上の次の下位のドライバーに渡す必要があります。

パラメーター

ファイルシステムまたはフィルタードライバーは、指定された IRP で **IoGetLocation** を呼び出して、irp 内の独自の **スタックの場所** へのポインターを取得します。次の一覧には、*irpSp* として示されています。(IRP は、*irp* として表示されます)。ドライバーは、ファイルシステムコントロール要求の処理中に、IRP の次のメンバーと IRP スタックの場所に設定されている情報を使用できます。

DeviceObject

ターゲットデバイスオブジェクトへのポインター。

Irp -> AssociatedIrp.SystemBuffer

ターゲットボリュームのファイルシステムまたはファイルシステムフィルタードライバーに渡されるシステム指定の入力バッファーへのポインター。メソッドのバッファーまたはメソッドの直接 i/o に使用されます。このパラメーターが必須かどうかは、特定のファイルシステム制御コードによって異なります。

Irp -> IoStatus

最後の完了状態と要求された操作に関する情報を受け取る **IO _ 状態 _ ブロック** 構造体へのポインター。

Irp -> MdlAddress

ターゲットボリュームのファイルシステムまたはファイルシステムフィルタードライバーに渡される出力バッファーを記述するメモリ記述子リスト (MDL) のアドレス。メソッドダイレクト i/o に使用されます。このパラメーターが必須かどうかは、特定の i/o 制御コードによって異なります。

Irp -> UserBuffer

ターゲットボリュームのファイルシステムまたはファイルシステムフィルタードライバーに渡される、呼び出し元から提供される出力バッファーへのポインター。メソッドのバッファーに使用されるか、または i/o ではないメソッドに使用されます。このパラメーターが省略可能か必須かは、特定の i/o 制御コードによって決まります。

IrpSp -> FileObject

*DeviceObject*に関連付けられているファイルオブジェクトへのポインター。

IrpSp -> FileObject パラメーターには、関連する **FileObject** フィールドへのポインターが含まれています。これは、ファイルオブジェクト構造でもあります。ファイルオブジェクト構造の関連性のある **fileobject** フィールドは、IRP_MJ_FILE_SYSTEM_CONTROL の処理中は無効であるため、___ 使用しないでください。

IrpSp -> Flags

次のフラグは、IRP を確認するボリュームに設定できます。

SL に _ による _ RAW _ マウント

IrpSp -> MajorFunction

IRP _ MJ _ ファイルシステムコントロールを指定します。

IrpSp -> MinorFunction

次のいずれか:

- IRP の全 _ _ カーネル _ 呼び出し

- IRP の全 __ 負荷の _ ファイル _ システム
- IRP の全 __ マウント _ ボリューム
- IRP の全 __ ユーザーのユーザー _ FS _ 要求
- IRP の全 __ 検査の _ ボリューム

IrpSp- > Parameters. FileSystemControl. FsControlCode

ターゲットボリュームのファイルシステムまたはファイルシステムフィルタードライバーに渡される FSCTL 関数コード。IRP の __ ユーザー _ FS _ 要求のみで使います。

IOCTL 要求と FSCTL 要求の詳細については、「カーネルモードアーキテクチャガイド」の「[i/o 制御コードの使用](#)」および Microsoft Windows SDK のドキュメントの「[デバイスの入力と出力の制御コード](#)」を参照してください。

IrpSp- > Parameters. FileSystemControl. InputBufferLength

*Irp > AssociatedIrp.SystemBuffer*が指すバッファのサイズ (バイト単位)。

IrpSp- > Parameters. Filesystembufferlength

* > UserBuffer*によってポイントされるバッファのサイズ (バイト単位)。

IrpSp- > Parameters Type3InputBuffer

メソッドを使用したカーネルモード要求の入力バッファが _ ありません。

IrpSp- > MountVolume. DeviceObject

ボリュームがマウントされる実際のデバイスのデバイスオブジェクトへのポインター。ファイルシステムフィルタードライバーでは、このパラメーターを使用しないでください。

IrpSp- > MountVolume. Vpb

マウントするボリュームのボリュームパラメーターブロック (VPB) へのポインター。リムーバブルメディアをサポートするファイルシステムでは、このパラメーターで渡された VPB を以前に使用した VPB に置き換えることができます。このようなファイルシステムでは、ボリュームがマウントされた後、このポインターが有効であるとは見なされなくなります。これらのファイルシステムをフィルタ処理するファイルシステムフィルタードライバーでは、次のように、このパラメーターを使用する必要があります。IRP を下位レベルのドライバーに送信する前に、フィルタによって MountVolume の値が保存されます。* > > * ボリュームが正常にマウントされると、フィルタはこのポインターを使用して、正しい VPB ポインターを取得できます。

IrpSp- > Parameters. VerifyVolume. DeviceObject

検証するボリュームのデバイスオブジェクトへのポインター。

IrpSp- > Parameters. VerifyVolume. Vpb

検証するボリュームの VPB へのポインター。

関連項目

[IO _ スタックの _ 場所](#)

[IO _ 状態 _ ブロック](#)

[IoBuildAsynchronousFsdRequest](#)

[IoBuildDeviceIoControlRequest](#)

[IoBuildSynchronousFsdRequest](#)

[Iogetlocation Entiの場所](#)

[IRP](#)

[ZwDeviceIoControlFile](#)

IRP _ MJ の _ フラッシュ _ バッファー (IFS)

2020/08/04 • [Edit Online](#)

送信時

IRP _ MJ _ FLUSH _ BUFFERS 要求は、バッファーデータをディスクにフラッシュする必要がある場合に、i/o マネージャーと他のオペレーティングシステムコンポーネント、およびその他のカーネルモードドライバによって送信されます。たとえば、ユーザーモードアプリケーションが `Flushfilebuffers` などの Microsoft Win32 関数を呼び出した場合に、これを送信できます。(ファイルシステムドライバおよびファイルシステムフィルタードライバの場合、通常は、(IRP の送信よりも) `Ccflushcache` を呼び出すことをお勧めします)。

データの内部バッファーを保持するすべてのファイルシステムおよびフィルタードライバは、この IRP を処理する必要があります。これにより、ファイルデータまたはメタデータへの変更をシステムのシャットダウン間で保持できるようになります。

操作: ファイルシステムドライバ

ファイルシステムドライバは、ファイルオブジェクトに関連付けられている重要なデータまたはメタデータをディスクにフラッシュし、IRP を完了する必要があります。この IRP を処理する方法の詳細については、FASTFAT サンプルを調べてください。

操作: ファイルシステムフィルタードライバ

フィルタードライバは、ファイルオブジェクトに関連付けられている重要なデータまたはメタデータをディスクにフラッシュし、この IRP をスタック上の次の下位のドライバに渡します。

パラメーター

ファイルシステムまたはフィルタードライバは、指定された IRP で `Iogetlocation enti` を呼び出して、irp 内の独自の **スタックの場所** へのポインターを取得します。次の一覧には、`irpsp` として示されています。(IRP は、`irp` として表示されます)。ドライバは、次の IRP のメンバーで設定されている情報を使用できます。また、バッファーのフラッシュの要求を処理するときに、IRP スタックの場所です。

DeviceObject

ターゲットデバイスオブジェクトへのポインター。

Irp- > iostatus

最後の完了状態と要求された操作に関する情報を受け取る **IO _ 状態 _ ブロック** 構造体へのポインター。

IrpSp- > FileObject

DeviceObject に関連付けられているファイルオブジェクトへのポインター。

IrpSp- > FileObject パラメーターには、関連する `FileObject` フィールドへのポインターが含まれています。これは、ファイルオブジェクト構造でも _ あります。ファイルオブジェクト構造の "関連性のある `fileobject` " フィールド _ は、IRP MJ FLUSH バッファーの処理中は無効であり、_ _ _ 使用しないでください。

IrpSp- > MajorFunction

IRP _ MJ フラッシュバッファーを指定し _ _ ます。

関連項目

[CcFlushCache](#)

[IO _ スタックの _ 場所](#)

IO _ 状態 _ ブロック

logetlocation Entiの場所

IRP

IRP _ MJ の _ フラッシュ _ バッファー (WDK カーネルリファレンス)

2020/08/04 • [_ Edit Online](#)

ターゲットデバイスのデバイスドライバに渡される、呼び出し元から提供される出力バッファへのポインター。このパラメータは、メソッドの `_` バッファまたはメソッドが `i/o` ではない場合に使用され `_` ます。このパラメータが省略可能必須かは、特定の `i/o` 制御コードによって決まります。

IrpSp- > FileObject

*DeviceObject*に関連付けられているファイルオブジェクトへのポインター。

*IrpSp- > FileObject*パラメーターには、関連するFileObjectフィールドへのポインターが含まれています。これは、ファイルオブジェクト構造でも_あります。ファイルオブジェクト構造のMJ fileobjectフィールド_は、IRP 内部デバイスコントロールの処理中は無効であり、_ _ _ _ 使用しないでください。

IrpSp- > MajorFunction

IRP _ MJ _ 内部デバイスコントロールを指定し_ _ ます。

IrpSp- > DeviceIoControl. InputBufferLength

*Irp > AssociatedIrp.Systembuffer*が指すバッファのサイズ (バイト単位)。

IrpSp- > DeviceIoControl. IoControlCode

ターゲットデバイスのデバイスドライバに渡される IOCTL 関数コード。

IOCTL 要求の詳細については、「カーネルモードアーキテクチャガイド」の「[i/o 制御コードの使用](#)」および Microsoft Windows SDK のドキュメントの「デバイスの入力と出力の制御コード」を参照してください。

IrpSp- > DeviceIoControl の長さ

* > UserBuffer*によってポイントされるバッファのサイズ (バイト単位)。

IrpSp- > DeviceIoControl. Type3InputBuffer

メソッドを使用したカーネルモード要求の入力バッファが_ありません。

関連項目

[IO _ スタックの _ 場所](#)

[IO _ 状態 _ ブロック](#)

[IoBuildDeviceIoControlRequest](#)

[IoGetLocation Entiの場所](#)

[IoGetFunctionCodeFromCtlCode](#)

[IRP](#)

[IRP _ MJ _ デバイス _ コントロール](#)

[IRP _ MJ _ 内部 _ デバイス _ コントロール \(WDK カーネルリファレンス\)](#)

[I/O 制御コードの使用](#)

[ZwDeviceIoControlFile](#)

IRP_MJ_LOCK_CONTROL

2020/04/24 • [Edit Online](#)

送信時

IRP_MJ_LOCK_CONTROL 要求は、i/o マネージャーおよびその他のカーネルモードドライバによって送信されます。

操作: ファイルシステムドライバー

ファイルシステムドライバーは、ターゲットデバイスオブジェクトがファイルシステムのコントロールデバイスオブジェクトであるかどうかを判断するために、ファイルオブジェクトを抽出してデコードする必要があります。この場合、ファイルシステムドライバーは、ロック要求を処理せずに、必要に応じて IRP を完了する必要があります。

そうしないと、ユーザーファイルを開くハンドルに対して要求が発行された場合、ファイルシステムドライバーは、マイナー関数コードによって示された操作を実行し、IRP を完了する必要があります。それ以外の場合、ドライバーは IRP を失敗させる必要があります。

有効なマイナー関数コードを次に示します。

III	II
IRP_MN_LOCK	Microsoft Win32 LockFile 関数を呼び出したユーザーモードアプリケーションに代わって、バイト範囲ロック要求を示します。
IRP_MN_UNLOCK_ALL	ファイルに対するすべてのバイト範囲ロックを解放する要求を示します。通常は、ファイルオブジェクトへの最後の未処理ハンドルが閉じられているためです。
IRP_MN_UNLOCK_ALL_BY_KEY	指定されたキー値を持つすべてのバイト範囲ロックを解放する要求を示します。
IRP_MN_UNLOCK_SINGLE	Microsoft Win32 UnlockFile 関数を呼び出したユーザーモードアプリケーションに代わって、1つのバイト範囲ロックを解放する要求を示します。

操作: ファイルシステムフィルタードライバー

ファイルシステムフィルタードライバーは、必要な処理を実行した後、スタック上の次の下位のドライバーに IRP を渡す必要があります。

パラメーター

ファイルシステムまたはフィルタードライバーは、指定された IRP で **IoGetLocation** **enti**を呼び出して、irp 内の独自の**スタックの場所**へのポインターを取得します。次の一覧には、*irpsp*として示されています。(IRP は、*irp*として表示されます)。ドライバーは、ロック制御要求を処理するときに、IRP の次のメンバーと IRP スタックの場所に設定されている情報を使用できます。

DeviceObject

ターゲットデバイスオブジェクトへのポインター。

Irp>IoStatus

最終的な完了状態と要求された操作に関する情報を受け取る、[IO_ステータス_ブロック](#)構造へのポインター。

IrpSp->FileObject

*DeviceObject*に関連付けられているファイルオブジェクトへのポインター。

*IrpSp->FileObject*パラメーターには、関連するfileobjectフィールドへのポインターが含まれています。これは、ファイル_オブジェクト構造体でもあります。IRP_MJ_LOCK_CONTROL の処理中は、ファイル_オブジェクト構造の関連性のあるフィールドは無効であるため、使用できません。

IrpSp->フラグ

次の1つまたは複数を実行します。

FLAG	「
SL_EXCLUSIVE_LOCK	このフラグが設定されている場合は、排他的なバイト範囲ロックが要求されます。それ以外の場合は、共有ロックが要求されません。
SL_FAIL_IMMEDIATELY	このフラグが設定されている場合、ロック要求はすぐに許可されない場合は失敗します。

IrpSp->MajorFunction

IRP_MJ_ロック_コントロールを指定します。

IrpSp->MinorFunction

次のいずれかを指定します。

- IRP__ロック
- IRP__ロック解除_すべて
- IRP_、_キーによってすべての__ロック解除_
- IRP__のロック解除_SINGLE

IrpSp->パラメーター。LockControl。ByteOffset

ロックまたはロック解除するバイト範囲のファイル内の開始バイトオフセット。

IrpSp->パラメーター。LockControl。キー

バイト範囲ロックのキー。

IrpSp->Parameters。LockControl。Length

ロックまたはロック解除するバイト範囲の長さ (バイト単位)。

「

[FltProcessFileLock](#)

[FsRtlProcessFileLock](#)

[IO_スタック_の場所](#)

[IO_状態_ブロック](#)

[IoGetCurrentIrpStackLocation](#)

[IRP](#)

IRP _ MJ _ PNP (IFS)

2020/08/04 • [Edit Online](#)

送信時

プラグアンドプレイ Manager は、_ _ システムでプラグアンドプレイアクティビティが発生するたびに、IRP MJ PNP 要求を送信します。その他のオペレーティングシステムコンポーネントおよびその他のカーネルモードドライバは、_ _ マイナー関数コードに応じて、特定の IRP MJ PNP 要求を送信することもできます。

ドライバのプラグアンドプレイの IRP 処理要件の詳細については、「[プラグアンドプレイ](#)」を参照してください。

IRP MJ PNP のマイナー関数コードに関する参照情報について _ _ は、「[プラグアンドプレイの小さな irp](#)」を参照してください。

操作: ファイルシステムドライバ

ファイルシステムは、マイナー関数のコードを確認して、どの操作が要求されているかを判断する必要があります。ファイルシステムでは、次のマイナー関数コードを処理する必要があります。

IRP MJ PNP	説明
IRP_MN_CANCEL_REMOVE_DEVICE	以前のクエリ削除要求が取り消されたことを示します。この要求は、キャンセルに関連するクリーンアップを実行する必要がある場合に、ファイルシステムに警告するために送信されます。
IRP_MN_QUERY_REMOVE_DEVICE	デバイスが削除されようとしていることを示します。ファイルシステムがデバイスにマウントされている場合、PnP マネージャーはこの要求をファイルシステムとファイルシステムフィルタに送信します。デバイスに開いているハンドルがある場合、通常、ファイルシステムはクエリ削除要求に失敗します。それ以外の場合、通常、ファイルシステムはボリュームをロックして、今後の作成要求が成功しないようにします。マウントされたファイルシステムがクエリ削除要求をサポートしていない場合、PnP マネージャーはデバイスのクエリ削除要求に失敗します。
IRP_MN_REMOVE_DEVICE	デバイスが削除されようとしていることを示します。ファイルシステムがデバイスにマウントされている場合、PnP マネージャーは、この IRP をファイルシステムとファイルシステムフィルタに送信します。ファイルシステムは、この IRP をデバイスのストレージドライバにすぐに渡す必要があります。その後、ファイルシステムによってボリュームがマウント解除される完了ルーチンを設定します。
IRP_MN_START_DEVICE	デバイスが起動中であることを示します。ファイルシステムは、この IRP をデバイスのストレージドライバに渡す必要があります。
IRP_MN_SURPRISE_REMOVAL	デバイスが削除されたことを示します。ファイルシステムがデバイスにマウントされている場合、PnP マネージャーは、この IRP をファイルシステムとファイルシステムフィルタに送信します。ファイルシステムは、この IRP をデバイスのストレージドライバにすぐに渡す必要があります。その後、ファイルシステムによってボリュームがマウント解除される完了ルーチンを設定します。

操作: ファイルシステムフィルタードライバー

ファイルシステムフィルタードライバーは、次のガイドラインに従って、PnP Irp を処理する必要があります。

- ユーザーがボリュームを正常に削除しようとする、PnP マネージャーは、IRP を使用した `_ _` クエリの `_` 削除デバイス要求を送信し `_` ます。この IRP を受信すると、フィルターはボリューム上の開いているハンドルをすべて閉じ、スタック上の次の下位のドライバに IRP を渡します。これは非常に重要です。ドライバが開いているハンドルをすべて閉じるのに失敗した場合、ボリュームがマウント解除されないようにします。これにより、物理デバイスの取り出しを防止できます。

NOTE

IRP は、`_` デバイスの `_` 削除要求を受信すると `_ _`、安全に削除できるすべてのボリュームを直ちにマウント解除します。そのため、FAT ボリュームにアタッチされたフィルターは、フィルターの完了ルーチンが呼び出される前に、そのフィルターデバイスオブジェクトが解放されることを期待する必要があります。NTFS ファイルシステムでは、この操作は行われません。そのため、NTFS ボリュームにアタッチされたフィルターは、フィルターの完了ルーチンが呼び出されたときに、そのデバイスオブジェクトがボリュームにアタッチされることを期待できます。

- Irp が完了した後に受信される irp はデバイス要求を削除しますが、IRP によって削除されたデバイスの削除要求または IRP の削除後のデバイス要求を受信 `_ _ _ _` する前に `_ _ _ _ _ _`、スタックを (ストレージデバイススタックによって失敗するように) 安全に渡すことができます。
- IRP による `_ _` クエリの `_ _` 削除要求に応じて、ボリュームの開いているハンドルをすべて既に閉じた後に、フィルターが irp を完了したデバイスの削除要求を受け取る `_ _ _ _` と、ハンドルを再び開くことができます。ただし、フィルターは、IRP がスタック内のドライバによって正常に完了した後で、完了ルーチンでのみ実行できます。
- フィルターは、irp が完了したデバイスの削除要求を受信したときに、irp で irp を保持していない `_ _ _` 限り、irp で処理を実行する必要はありません `_ _ _ _`。キューで Irp を保持している場合、フィルターはそのボリュームのすべての Irp をデキューし、エラーが発生した `< >` 後、`< >` スタック上の次の下位のドライバに irp を渡す前に失敗します。
- IRP の突然の `_ _ _` 削除要求を受信すると、フィルターは次の操作を実行します。
 - 未処理の参照がないまでファイルシステムがスタックをクリーンアップできないため、開いているすべてのハンドルをボリュームに対して閉じます。
 - フィルターが Irp をキューに保持している場合は、エラーを発生させるか、スタックに渡すことができます (ストレージデバイススタックによって失敗する場合)。

パラメーター

ファイルシステムまたはフィルタードライバーは、指定された IRP で `IoGetLocation` を呼び出して、irp 内の独自の `スタックの場所` へのポインターを取得します。次の一覧には、`irpSp` として示されています。(IRP は、`irp` として表示されます)。ドライバは、プラグアンドプレイ要求の処理中に、IRP の次のメンバーと IRP スタックの場所に設定されている情報を使用できます。

DeviceObject

ターゲットデバイスオブジェクトへのポインター。

Irp->IoStatus

最後の完了状態と要求された操作に関する情報を受け取る `IO _ 状態 _ ブロック` 構造体へのポインター。

IrpSp->FileObject

PnP Irp の場合、このポインターは NULL である必要があります。

IrpSp->MajorFunction

IRP MJ PNP を指定し `_ _` ます。

IrpSp->MinorFunction

次のいずれか:

- IRP を終了する `_ _ _` デバイスの削除 `_`

- [IRP を実行する __ クエリ_ デバイスの削除 _](#)
- [IRP のすべての __ デバイスの削除 _](#)
- [IRP の全 __ 開始 _ デバイス](#)
- [IRP _ の _ 突然の _ 削除](#)

関連項目

[IO _ スタックの _ 場所](#)

[IO _ 状態 _ ブロック](#)

[IoGetLocationEntiの場所](#)

[IRP](#)

[IRP _ MJ _ PNP \(WDK カーネルリファレンス\)](#)

[IRP を終了する __ _ デバイスの削除 _](#)

[IRP を実行する __ クエリ_ デバイスの削除 _](#)

[IRP のすべての __ デバイスの削除 _](#)

[IRP の全 __ 開始 _ デバイス](#)

[IRP _ の _ 突然の _ 削除](#)

IRP _ MJ _ クエリ _ EA

2020/08/04 • [Edit Online](#)

送信時

IRP _ MJ _ QUERY _ EA 要求は、ユーザーモードアプリケーションがファイルの拡張属性に関する情報を要求したときに、i/o マネージャーと他のオペレーティングシステムコンポーネント、およびその他のカーネルモードドライバによって送信されます。

操作: ファイルシステムドライバ

ファイルシステムで拡張属性がサポートされている場合、ファイルシステムドライバはクエリを処理し、IRP を完了する必要があります。それ以外の場合は、ステータス EAS がサポートされていない状態を返します。

操作: ファイルシステムフィルタードライバ

フィルタードライバは、この IRP をスタック上の次の下位のドライバに渡す必要があります。

パラメーター

ファイルシステムまたはフィルタードライバは、指定された IRP で **IoGetLocation** 呼び出しを使用して、irp 内の独自の **スタックの場所** へのポインターを取得します。次の一覧には、*Irpsp* として示されています。(IRP は、*Irpsp* として表示されます)。ドライバは、irp _ MJ QUERY EA 要求の処理中に、irp の次のメンバーと irp スタックの場所に設定されている情報を使用できます。

DeviceObject

ターゲットデバイスオブジェクトへのポインター。

Irpsp -> AssociatedIrps

中間システムバッファーとして使用されるシステム指定の出力バッファーへのポインター。メソッドバッファー i/o に使用されます。

Irpsp -> IoStatus

最終的な完了ステータスと要求された操作に関する情報を受け取る **IO _ 状態 _ ブロック** 構造体へのポインター。

Irpsp -> MdlAddress

拡張属性情報を受け取る出力バッファーを記述するメモリ記述子リスト (MDL) のアドレス。メソッドダイレクト i/o に使用されます。

Irpsp -> UserBuffer

拡張属性情報を受け取る、呼び出し元によって提供されるファイルへのポインター。**** _ 完全 _ EA _ 情報**** が構造化された出力バッファーです。I/O ではないメソッドに対して使用されます。

Irpsp -> FileObject

DeviceObject に関連付けられているファイルオブジェクトへのポインター。

Irpsp -> FileObject パラメーターには、関連する *FileObject* フィールドへのポインターが含まれています。これは、ファイルオブジェクト構造でもあります。ファイルオブジェクト構造の "関連性のある fileobject" フィールドは、IRP MJ QUERY EA の処理中は無効であり、使用しないでください。

Irpsp -> Flags

次の値のうち1つ以上を指定します。

III	II
SL_INDEX_SPECIFIED	インデックスが指定されている拡張属性の一覧のエントリから、" <i>IrpSp</i> - > パラメーター" でスキャンを開始します。
SL_RESTART_SCAN	一覧の最初のエントリからスキャンを開始します。このフラグが設定されていない場合は、以前の IRP_MJ_QUERY_EA 要求からスキャンを再開します。
SL_RETURN_SINGLE_ENTRY	最初に見つかったエントリだけを返します。

IrpSp- > *MajorFunction*

IRP_MJ_QUERY_EA を指定し __ ます。

IrpSp- > パラメーター. *Quer. e*:

拡張属性リストのスキャンを開始する位置のエントリのインデックス。このパラメーターは、__ 指定された SL インデックスフラグが設定されていない場合や、*irpSp* > パラメーターが空でないリストを指している場合は無視されます。

IrpSp- > パラメーター. *EaList*

呼び出し元から提供されるファイルへのポインターは、クエリ対象の拡張属性を指定する** __ EA _ 情報**で構成された入力バッファーを取得します。

IrpSp- > パラメーター. この値は、

IrpSp によってポイントされたバッファーの長さ (バイト単位)。* > Quer. *ealist**。

IrpSp- > パラメーター. *quer*

出力バッファーの長さ (バイト単位)。

コメント

短いバッファーが指定され、ステータス _ バッファーの _ オーバーフローが返されると、NTFS は最後にファイル全体の _ 完全な _ EA 情報のエントリを返し _ ます。短いバッファーが指定されていて、ステータス _ バッファー _ が小さすぎる場合 _ 、NTFS は _ 完全な _ EA _ 情報エントリをすべてファイルに収めることができませんでした。

NOTE

Windows Vista 以降では、FAT16 は拡張属性をサポートしなくなりました。

関連項目

[_EA の完全な __ 情報](#)

[ファイル _ の _ EA _ 情報の取得](#)

[IO _ スタックの _ 場所](#)

[IO _ 状態 _ ブロック](#)

[IoCheckEaBufferValidity](#)

[IoGetLocationEntire場所](#)

[IRP](#)

[IRP _ MJ _ SET _ EA](#)

IRP _ MJ _ クエリ _ 情報 (IFS)

2020/08/04 • [Edit Online](#)

送信時

IRP _ MJ _ QUERY _ INFORMATION 要求は、i/o マネージャーおよびその他のカーネルモードドライバによって、送信されます。この要求は、たとえば、ユーザーモードアプリケーションが `GetFileInformationByHandle` などの Microsoft Win32 関数を呼び出したとき、またはカーネルモードコンポーネントが `Zwqueryinformationfile` を呼び出したときに送信できます。

操作: ファイルシステムドライバ

ファイルシステムドライバは、ファイルまたはディレクトリが開いているユーザーを表すかどうかを判断するために、ファイルオブジェクトを抽出してデコードする必要があります。その場合、ドライバはクエリを処理し、IRP を完了する必要があります。それ以外の場合、ドライバはクエリを処理せずに、必要に応じて IRP を完了する必要があります。

クエリ可能なファイルとディレクトリの情報の種類は、ファイルシステムによって異なりますが、一般的には次のものが含まれます。

FileAllInformation FileAlternateNameInformation Fileallinformation FileBasicInformation
FilecompressedFileEaInformation information FileInternalInformation FileNameInformation
FileNetworkOpenInformation FilePositionInformation FileStandardInformation FileStreamInformation File の情報をパラメーターとして `Zwqueryinformationfile` に渡すこともできますが、この情報はファイルシステムに依存していません。このため、`Zwqueryinformationfile` は、IRP _ MJ _ QUERY _ 情報要求をファイルシステムに送信することなく、この情報を直接提供します。

これらの情報の種類の詳細については、以下の「関連項目」も参照してください。使用可能なすべての情報の種類の一覧については、`_ ntifs` のファイル情報クラス列挙体を参照してください。

操作: Network リダイレクタードライバ

`RDBSS` に基づいていないネットワークリダイレクタードライバは、`___ Fileallinformation` または `FileNameInformation` に対する IRP MJ QUERY 情報要求を受信すると、`\\ \` サーバー名の前に1つの先頭に円記号が付いたファイル名の完全な "サーバー共有ファイル" パスで応答する必要があります。この名前情報の形式は、汎用名前付け規則 (UNC) 名 (たとえば、`* \ \ サーバー \ 共有 \ フォルダー \ filename.txt*`) としてアクセスされるファイル、またはマップされたドライブ (`x: \ folder \ filename.txt*` など) にあるファイルに対して返される必要があります。

ネットワークミニリダイレクタードライバ (`rdbs.sys` と動的にリンクするドライバ、または `rdbslib` を使用して静的にリンクするドライバ) の場合、`___ FILENAMEINFORMATION` に対する IRP MJ QUERY INFORMATION 要求は `RDBSS` によって内部的に処理され、正しい名前情報が返されます。ネットワークミニリダイレクタードライバの場合、`___ FileAllInformation` に対する IRP MJ QUERY INFORMATION 要求は、要求の名前情報部分に対して `RDBSS` によって内部的に処理されます。`FileAllInformation` 要求のその他の部分は、ネットワークミニリダイレクタードライバに対して、解決するために個別の要求として送信されます。

`FileAlternateNameInformation` に対する IRP MJ QUERY 情報要求を受信するネットワークリダイレクターは、ファイルの `___` 短い名前が存在する場合、パス情報を含まない短い名前 (8.3 文字) で応答する必要があります。

操作: ファイルシステムフィルタードライバ

フィルタードライバは、この IRP をスタック上の次の下位のドライバに渡す必要があります。

パラメーター

ファイルシステムまたはフィルタードライバは、指定された IRP で `Iogetlocation enti` を呼び出して、irp 内の独自の **スタックの**

場所へのポインターを取得します。次の一覧には、*irpsp*として示されています。(IRP は、*irp*として表示されます)。ドライバーは、次の IRP のメンバーと、クエリファイル情報要求の処理での IRP スタックの場所に設定されている情報を使用できます。

DeviceObject

ターゲットデバイスオブジェクトへのポインター。

Irp > AssociatedIrp.SystemBuffer

ファイルまたはディレクトリの情報が返される出力バッファへのポインター。この情報は、次のいずれかの構造に格納されます。

すべての情報をファイルに

ファイル 属性 タグ 情報

ファイルの 基本 情報

ファイルの 圧縮 情報

ファイル EA の 情報

ファイルの 内部 情報

ファイル 名 情報

ファイル ネットワークの オープン 情報

ファイルの 位置 情報

ファイルの 標準 情報

ファイル ストリームの 情報

ファイル リンクの 情報

Irp -> 最後の完了状態と要求された操作に関する情報を受け取る**IO 状態 ブロック**構造への *iostatus* ポインター。詳細については、**Zwqueryinformationfile**の*iostatusblock*パラメーターの説明を参照してください。ルーチン。

Irp -> *UserBuffer* (省略可能) 呼び出し元から提供される出力バッファへのポインター。i/o マネージャーによる i/o の完了時に、*Irp > AssociatedIrp.Systembuffer*の内容がコピーされます。ドライバーは、このバッファを使用して要求のデータを返しません。

Irpsp -> *DeviceObject*に関連付けられているファイルオブジェクトへの *FileObject* ポインター。

Irpsp -> *FileObject*パラメーターには、関連する*FileObject*フィールドへのポインターが含まれています。これは、ファイルオブジェクト構造でもあります。ファイルオブジェクト構造の *MJ fileobject*フィールドは、IRP のクエリ情報の処理中は無効であり、使用できません。

Irpsp -> *MajorFunction*は、IRP *MJ* クエリ情報を指定します。

Irpsp -> *Parameters. QueryFile. FileInformationClass*種類のファイル情報を照会します。このメンバーには、次のいずれかの値を指定できます。

I	II
■	ファイルの FILE_ALL_INFORMATION 構造体を返します。
FileAttributeTagInformation	ファイルの FILE_ATTRIBUTE_TAG_INFORMATION 構造体を返します。
FileBasicInformation	ファイルの FILE_BASIC_INFORMATION 構造体を返します。

I	II
Filecompress	ファイルの FILE_COMPRESSION_INFORMATION 構造体を返します。
FileEaInformation	ファイルの FILE_EA_INFORMATION 構造体を返します。
FileInternalInformation	ファイルのFILE_INTERNAL_INFORMATION構造体を返します。
FileNameInformation	ファイルのFILE_NAME_INFORMATION構造体を返します。
FileNetworkOpenInformation	ファイルの1つのFILE_NETWORK_OPEN_INFORMATION構造体を返します。
FilePositionInformation	ファイルの1つのFILE_POSITION_INFORMATION構造体を返します。
FileStandardInformation	ファイルの1つのFILE_STANDARD_INFORMATION構造体を返します。
FileStreamInformation	ファイルの1つのFILE_STREAM_INFORMATION構造体を返します。
File	ファイルのFILE_LINKS_INFORMATION構造体を返します。

Irpsp- >パラメーター。 *Irps* >*AssociatedIrps.Systembuffer*が指すバッファの長さをバイト単位で格納します。

解説

IRP _ MJ _ QUERY _ INFORMATION 操作は、常に i/o マネージャーによってバッファリングされます。要求されたファイルまたはディレクトリの情報を返すために使用される *Irps* >*AssociatedIrps.Systembuffer*は、i/o マネージャーによって非ページプールメモリから割り当てられます。その結果、オペレーティングシステムから返された *Irps* >*AssociatedIrps.Systembuffer*は、常に *Irpsp-* > *Parameters.queryfile* の長さの有効なアドレスになります。

* > *AssociatedIrps**は、i/o マネージャーによって内部的に使用され、ファイルシステムまたはファイルシステムフィルタードライバーでは使用できません。

関連項目

[ファイルの _ 配置 _ 情報](#)

[ファイル _ 属性 _ タグ _ 情報](#)

[ファイルの _ 基本 _ 情報](#)

[ファイルの _ 内部 _ 情報](#)

[ファイル _ 名 _ 情報](#)

[ファイル _ ネットワークの _ オープン _ 情報](#)

[ファイルの _ 位置 _ 情報](#)

[ファイルの _ 標準 _ 情報](#)

[ファイル _ ストリームの _ 情報](#)

[ファイル _ リンクの _ 情報](#)

[IO _ スタックの _ 場所](#)

[IO _ 状態 _ ブロック](#)

[IoCheckEaBufferValidity](#)

[Iogetlocation Entiの場所](#)

[IRP](#)

[IRP _ MJ の _ 設定 _ 情報](#)

[ZwQueryInformationFile](#)

IRP_MJ_クエリ_クォータ

2020/04/24 • [Edit Online](#)

送信時

IRP_MJ_クエリ_クォータ要求は、i/o マネージャーによって送信されます。この要求は、たとえば、ユーザーモードアプリケーションがdiskquotacontrol:: などの Microsoft Win32 メソッドを呼び出したときに送信できます。

操作: ファイルシステムドライバー

ファイルシステムがディスククォータをサポートしている場合、ファイルシステムドライバーはファイルオブジェクトを抽出してデコードし、ファイルまたはディレクトリが開いているユーザーを表すかどうかを判断します。その場合、ドライバーはクエリを処理し、IRP を完了する必要があります。それ以外の場合、ドライバーはクエリを処理せずに、必要に応じて IRP を完了する必要があります。

操作: ファイルシステムフィルタードライバー

フィルタードライバーは、クォータの動作を明示的にオーバーライドする必要がない限り、この IRP をスタック上の次の下位のドライバーに渡す必要があります。

パラメーター

ファイルシステムまたはフィルタードライバーは、指定された IRP で **IoGetLocation** 関数を呼び出して、irp 内の独自の **スタックの場所** へのポインターを取得します。次の一覧には、*irp* として示されています。(IRP は、*irp* として表示されます)。ドライバーは、次の IRP のメンバーと、クエリクォータ情報要求の処理での IRP スタックの場所に設定されている情報を使用できます。

DeviceObject

ターゲットデバイスオブジェクトへのポインター。

DeviceObject-> フラグ

次のようにして、データをドライバーに渡す方法を指定するために、IO_IO フラグと_DIRECT_IO フラグを使用して_します。

IO_METHOD	I/O METHOD
~ DO_BUFFERED_IO	~ DO_DIRECT_IO
METHOD_NEITHER	~ DO_BUFFERED_IO
DO_DIRECT_IO	METHOD_DIRECT
DO_BUFFERED_IO	~ DO_DIRECT_IO
METHOD_BUFFERED	DO_BUFFERED_IO
DO_DIRECT_IO	METHOD_BUFFERED

Irp-> AssociatedIrp

_バッファ_IO フラグが DeviceObject-> フラグで設定されている場合に、中間システムバッファとして使用されるシステム指定のバッファへのポインター。それ以外の場合、このメンバーはNULLに設定されます。

Irp > IoStatus

最終的な完了状態と要求された操作に関する情報を受け取る、IO_ステータス_ブロック構造へのポインター。

Irp-> UserBuffer

呼び出し元から提供されたファイル_クォータ_情報を指すポインター。ボリュームのクォータ情報を受け取ります。

IrpSp-> FileObject

DeviceObjectに関連付けられているファイルオブジェクトへのポインター。

IrpSp-> FileObjectパラメーターには、関連するfileobjectフィールドへのポインターが含まれています。これは、ファイル_オブジェクト構造体でもあります。IRP_MJ_QUERY_のクォータの処理中に、ファイル_オブジェクト構造の関連性のあるフィールドは無効です。使用しないでください。

IrpSp-> フラグ

このメンバーは、次の1つまたは複数にすることができます。

FLAG	「
SL_INDEX_SPECIFIED	IrpSp-> パラメーターでインデックスが指定されているクォータリストのエントリで、スキャンを開始します。QueryQuota. StartSid
SL_RESTART_SCAN	一覧の最初のエントリからスキャンを開始します。このフラグが設定されていない場合は、以前の IRP_MJ_QUERY_QUOTA 要求からスキャンを再開します。
SL_RETURN_SINGLE_ENTRY	最初に見つかったエントリだけを返します。

IrpSp-> MajorFunction

IRP_MJ_クエリ_クォータを指定します。

IrpSp-> Parameters. QueryQuota. Length

Irp > UserBufferが指すバッファーの長さ (バイト単位)。

IrpSp-> Parameters. QueryQuota. SidList

クォータ情報が返される Sid のリストへのポインター (省略可能)。リスト内の各エントリは、_クォータ_情報の構造を取得_ファイルです。この構造体は次のように定義されています。

<pre>typedef struct _FILE_GET_QUOTA_INFORMATION { ULONG NextEntryOffset; ULONG SidLength; SID Sid; } FILE_GET_QUOTA_INFORMATION, *PFILE_GET_QUOTA_INFORMATION;</pre>	
「「「	「
NextEntryOffset	バッファー内に複数のエントリが存在する場合は、次の FILE_GET_QUOTA_INFORMATION エントリのバイトオフセット。このメンバーは、このメンバーの後に他のエントリがない場合は0になります。
SidLength	Sidメンバーの長さ (バイト単位)。
Sid	セキュリティ識別子 (SID)

IrpSp-> パラメーター。QueryQuota. SidListLength

Sid のリスト (指定されている場合) の長さ (バイト単位)。

IrpSp-> Parameters. QueryQuota. StartSid

返された情報が最初のエントリ以外のエントリで開始されることを示す SID へのポインター (省略可能)。SID リストが指定されている場合、このパラメーターは無視されます。

「

[ファイル__クォータ_情報の取得](#)

[ファイル_クォータの_情報](#)

[IO_スタック_の場所](#)

[IO_状態_ブロック](#)

[IoCheckQuotaBufferValidity](#)

[IoGetCurrentIrpStackLocation](#)

[IRP](#)

[IRP_MJ_設定_クォータ](#)

IRP_MJ_クエリ_セキュリティ

2020/04/24 • [Edit Online](#)

送信時

IRP_MJ_QUERY_セキュリティ要求は、i/o マネージャーによって送信されます。たとえば、ユーザーモードアプリケーションがGetsecurityinfoなどの Microsoft Win32 関数を呼び出した場合に、これを送信できます。

操作: ファイルシステムドライバー

ファイルシステムドライバーは、ファイルオブジェクトを抽出してデコードし、ユーザーファイルまたはディレクトリが開いているかどうかを判断する必要があります。その場合、ドライバーはクエリを処理し、IRP を完了する必要があります。それ以外の場合、ドライバーはクエリを処理せずに、必要に応じて IRP を完了する必要があります。

操作: ファイルシステムフィルタードライバー

フィルタードライバーは、この IRP をスタック上の次の下位のドライバーに渡す必要があります。

パラメーター

ファイルシステムまたはフィルタードライバーは、指定された IRP で **IoGetLocation** を呼び出して、irp 内の独自の **スタックの場所** へのポインターを取得します。次の一覧には、*Irps* として示されています。(IRP は、*irp* として表示されます)。ドライバーは、クエリセキュリティ要求の処理で、IRP の次のメンバーと IRP スタックの場所に設定されている情報を使用できます。

DeviceObject

ターゲットデバイスオブジェクトへのポインター。

Irps->IoStatus

最後の完了状態と要求された操作に関する情報を受け取る、**IO_STATUS_BLOCK** 構造体へのポインター。

Irps->UserBuffer

指定したオブジェクトのセキュリティ記述子のコピーを受け取る、呼び出し元から提供される出力バッファーへのポインター。呼び出しプロセスには、オブジェクトのセキュリティステータスの指定した側面を表示する権限が必要です。**セキュリティ記述子**の構造体は、自己相対形式で返されます。

Irps->FileObject

*DeviceObject*に関連付けられているファイルオブジェクトへのポインター。

Windows XP 以降では、ファイルオブジェクトは名前付きデータストリームを表すことができます。名前付きデータストリームの詳細については、「**ファイルストリームの情報**」を参照してください。

Irps->FileObject パラメーターには、関連する **fileobject** フィールドへのポインターが含まれています。これは、**ファイルオブジェクト** 構造体でもあります。IRP_MJ_クエリ_セキュリティの処理中は、**ファイルオブジェクト** 構造体の "関連性のある" フィールドは無効です。このフィールドは使用できません。

Irps->MajorFunction

IRP_MJ_クエリ_セキュリティを指定します。

Irps->Parameters.QuerySecurity.Length

Irps->UserBuffer パラメーターが指すバッファーのサイズ (バイト単位)。

Irps->パラメーター.QuerySecurity.SecurityInformation

操作の **セキュリティ情報** 構造体へのポインター。

SECURITYINFORMATION Ⅱ	Ⅱ
OWNER_SECURITY_INFORMATION	オブジェクトの所有者識別子が照会されていることを示します。 READ_CONTROL アクセスが必要です。
GROUP_SECURITY_INFORMATION	オブジェクトのプライマリグループ識別子に対してクエリが行われていることを示します。READ_CONTROL アクセスが必要です。
DACL_SECURITY_INFORMATION	オブジェクトの随意アクセス制御リスト (DACL) に対してクエリが行われていることを示します。READ_CONTROL アクセスが必要です。
SACL_SECURITY_INFORMATION	オブジェクトのシステム ACL (SACL) に対してクエリが行われていることを示します。ACCESS_SYSTEM_SECURITY アクセスが必要です。

関連項目

[ファイル_ストリームの_情報](#)

[IO_スタック_の場所](#)

[IO_状態_ブロック](#)

[Iogetlocation Entiの場所](#)

[IRP](#)

[IRP_MJ_設定_セキュリティ](#)

[セキュリティ_記述子](#)

[セキュリティ_情報](#)

IRP_MJ_クエリ_ボリューム_情報

2020/04/24 • • [Edit Online](#)

送信時

IRP_MJ_QUERY_VOLUME_INFORMATION 要求は、i/o マネージャーによって送信されます。たとえば、ユーザーモードアプリケーションがGetdiskfreespace領域やGetfiletypeなどの Microsoft Win32 関数を呼び出した場合に、これを送信できます。

操作: ファイルシステムドライバー

ファイルシステムドライバーは、ターゲットデバイスオブジェクトがファイルシステムのコントロールデバイスオブジェクトであるかどうかを判断するために、ファイルオブジェクトを抽出してデコードする必要があります。その場合、またはボリュームが開いているハンドル (またはボリューム上のオブジェクトが開いているハンドル) で要求が発行されている場合は、ファイルシステムドライバーが要求を処理し、IRP を完了する必要があります。

それ以外の場合、ファイルシステムドライバーはクエリを失敗させ、IRP を完了する必要があります。

クエリ可能なボリューム情報の種類は、ファイルシステムによって異なりますが、一般的には次のようなものがあります。

FileFsAttributeInformation

FileFsDeviceInformation

FileFsSizeInformation

FileFsVolumeInformation

使用可能なすべての情報の種類の一覧については、以下の「*Irp->Parameters*」を参照してください。

操作: ネットワークリダイレクトドライバー

FileFsDeviceInformation の要求を受信するネットワークリダイレクターは、ファイル **_FS_デバイス_情報** 構造のDeviceCharacteristicsメンバーのオプションの1つとして、ファイル_リモート_デバイスを含める必要があります。

操作: ファイルシステムフィルタードライバー

フィルタードライバーは、この IRP をスタック上の次の下位のドライバーに渡す必要があります。

パラメーター

ファイルシステムまたはフィルタードライバーは、指定された IRP で **loglocation enti** を呼び出して、irp 内の独自の **スタックの場所** へのポインターを取得します。次の一覧には、*irp* として示されています。(IRP は、*irp* として表示されます)。ドライバーは、IRP の次のメンバーと IRP スタックの場所に設定されている情報を、クエリボリューム情報要求の処理に使用できます。

DeviceObject

ターゲットデバイスオブジェクトへのポインター。

Irp->AssociatedIrp

ボリューム情報が返されるシステム提供の出力バッファーへのポインター。この情報は、次のいずれかの構造に格納されます。

ファイル_FS_属性_情報

ファイル_FS_制御_情報

ファイル_FS_デバイス_情報

ファイル_FS_ドライバー_パス_情報

ファイル_FS_完全な_サイズ_情報

ファイル_FS_OBJECTID_情報

ファイル_FS_サイズ_情報

ファイル_FS_ボリューム_フラグ_情報

ファイル_FS_ボリューム_情報

ファイル_FS_セクタ_サイズ_情報

FileFsVolumeFlagsInformation クラスと関連付けられている**ファイル_FS_ボリューム_情報**構造は、Windows Vista 以降のバージョンで使用できます。

*Irp->IoStatus*最終的な完了状態と要求された操作に関する情報を受け取る、**IO_ステータス_ブロック**構造へのポインター。

Irp->UserBuffer/o マネージャーによる i/o の完了時に、*Irp->AssociatedIrp*の内容をコピーする、呼び出し元から提供される出力バッファーへのポインター (省略可能)。ドライバーは、このバッファーを使用して要求のデータを返しません。

*Irps->FileObject DeviceObject*に関連付けられているファイルオブジェクトへのポインター。

*Irps->FileObject*パラメーターには、関連するfileobjectフィールドへのポインターが含まれています。これは、ファイル_オブジェクト構造体でもあります。IRP_MJ_QUERY_ボリューム_情報の処理中は、ファイル_オブジェクト構造の関連性のあるフィールドは無効であり、使用できません。

*Irps->MajorFunction*IRP_MJ_クエリ_ボリューム_情報を指定します。

*Irps->Parameters. QueryVolume. FsInformationClass*ファイルシステムによって返されるボリューム情報の種類を指定します。このメンバーは、次のいずれかになります。

VALUE	II
FileFsAttributeInformation	ボリュームを担当するファイルシステムに関する属性情報を含む FILE_FS_ATTRIBUTE_INFORMATION 構造体を返します。
FileFsControllInformation	ボリュームに関するファイルシステムコントロール情報を含む FILE_FS_CONTROL_INFORMATION 構造体を返します。
FileFsDeviceInformation	ボリュームのデバイス情報を含む FILE_FS_DEVICE_INFORMATION 構造体を返します。
FileFsDriverPathInformation	指定されたドライバーがボリュームの i/o パスにあるかどうかに関する情報を含む FILE_FS_DRIVER_PATH_INFORMATION 構造体を返します。 IRP_MJ_QUERY_VOLUME_INFORMATION要求の発信者は、IRP をファイルシステムボリュームのデバイススタックに送信する前に、そのドライバーの名前をFILE_FS_DRIVER_PATH_INFORMATION構造体に格納する必要があります。

VALUE	II
FileFsFullSizeInformation	ボリューム上で使用可能な領域の合計量に関する情報を含む FILE_FS_FULL_SIZE_INFORMATION 構造体を返します。
FileFsObjectIdInformation	ボリュームのファイルシステム固有のオブジェクト ID 情報を含む FILE_FS_OBJECTID_INFORMATION 構造体を返します。これは、オペレーティングシステムによって割り当てられる (GUID ベースの) 一意のボリューム名と同じではないことに注意してください。
FileFsSizeInformation	IRP_MJ_QUERY_VOLUME_INFORMATION要求を発信したスレッドに関連付けられているユーザーが使用できるボリューム上の領域の量に関する情報を含む FILE_FS_SIZE_INFORMATION 構造体を返します。
FileFsVolumeInformation	ボリュームラベル、シリアル番号、作成時刻などのボリュームに関する情報を含む FILE_FS_VOLUME_INFORMATION を返します。
FileFsSectorSizeInformation	ボリュームの物理的および論理的なセクターサイズに関する情報を含む FILE_FS_SECTOR_SIZE_INFORMATION 構造体を返します。

*Irpsp->*パラメーター。 *QueryVolume.Irp>UserBuffer*が指すバッファの長さ (バイト単位)。返されると、この変数はバッファに書き込まれたバイト数を受け取ります。

「

[ファイル_FS_属性_情報](#)

[ファイル_FS_制御_情報](#)

[ファイル_FS_デバイス_情報](#)

[ファイル_FS_ドライバ_パス_情報](#)

[ファイル_FS_完全な_サイズ_情報](#)

[ファイル_FS_OBJECTID_情報](#)

[ファイル_fs_セクタ_サイズ_情報](#) [ファイル_FS_サイズ_情報](#)

[ファイル_FS_ボリューム_情報](#)

[IO_スタック_の場所](#)

[IO_状態_ブロック](#)

[IoGetCurrentIrpStackLocation](#)

[IRP](#)

[IRP_MJ__ボリューム_情報の設定](#)

[ZwQueryVolumeInformationFile](#)

IRP _ MJ _ READ (IFS)

2020/08/04 • [Edit Online](#)

送信時

IRP _ MJ _ READ 要求は、i/o マネージャーまたはファイルシステムドライバーによって送信されます。この要求は、たとえば、ユーザーモードアプリケーションがReadFileなどの Microsoft Win32 関数を呼び出したとき、またはカーネルモードコンポーネントがzwreadfileを呼び出したときに送信できます。

操作: ファイルシステムドライバー

ファイルシステムドライバーは、パラメーターとマイナー関数コードを決定するために、ファイルオブジェクトを抽出してデコードする必要があります。

メモリ記述子リスト (MDL) の読み取り要求では、ファイルシステムがマイナー関数コードを確認して、どの操作が要求されているかを判断する必要があります。有効なマイナー関数コードを次に示します。これらは、キャッシュされたファイル i/o に対してのみ使用できます。

- IRP _ _ 完了完了
- IRP _ _ 完全完了 _ MDL
- IRP _ の _ 包括的な _ MDL _ DPC
- IRP _ _ 圧縮済み
- IRP の全 _ _ DPC
- IRP の全 _ _ MDL
- IRP の全 _ _ MDL _ DPC
- IRP _ _ 通常

この IRP の処理の詳細については、Windows Driver Kit (WDK) に含まれている CDFS と FASTFAT のサンプルを参照してください。

操作: ファイルシステムフィルタードライバー

フィルタードライバーは必要な処理を実行し、フィルターの性質に応じて、IRP を完了するか失敗させるか、スタック上の次の下位のドライバーに渡します。

パラメーター

ファイルシステムまたはフィルタードライバーは、指定された IRP でlogetlocation entiを呼び出して、irp 内の独自のスタックの場所へのポインターを取得します。次の一覧には、irpspとして示されています。(IRP は、irpとして表示されます)。ドライバーは、次の IRP のメンバーと IRP スタックの場所に含まれている情報を読み取り要求の処理中に使用できます。

DeviceObject

ターゲットデバイスオブジェクトへのポインター。

Irp > AssociatedIrp.SystemBuffer

実行バッファー _ _ IO フラグが*DeviceObject に > *設定されている場合、中間システムバッファーとして使用されるシステム指定

のバッファーへのポインター。それ以外の場合、このメンバーはNULLに設定されます。

Irp- > iostatus

最後の完了状態と要求された操作に関する情報を受け取るIO _ 状態 _ ブロック構造体へのポインター。詳細については、「[Zwreadfile](#)への*iostatusblock*パラメーターの説明」を参照してください。

Irp- > mdladdress

読み取るデータを含むページを記述するメモリ記述子リスト (MDL) のアドレス。

Irp- > UserBuffer

ファイルから読み取られたデータを受け取る、呼び出し元から提供される出力バッファーへのポインター。

IrpSp- > FileObject

*DeviceObject*に関連付けられているファイルオブジェクトへのポインター。FO _ 同期 _ IO フラグが*irpsp- > > Flags*で設定されている場合、ファイルオブジェクトは同期 i/o 用に開かれています。

*Irpsp- > FileObject*パラメーターには、関連するFileObjectフィールドへのポインターが含まれています。これは、ファイルオブジェクト構造でも _ あります。RelatedFileObject _ IRP MJ READ の処理中は、ファイルオブジェクト構造の "関連性のあるfileobject" フィールドは無効です _ 。使用しないで _ ください。

IrpSp- > MajorFunction

IRP MJ READ を指定し _ ます。

IrpSp- > minorfunction

要求される操作を指定し、次のいずれかを含みます。

- IRP _ _ 完了完了
- IRP _ _ 完全完了 _ MDL
- IRP _ の _ 包括的な _ MDL _ DPC
- IRP _ _ 圧縮済み
- IRP の全 _ _ DPC
- IRP の全 _ _ MDL
- IRP の全 _ _ MDL _ DPC
- IRP _ _ 通常

IrpSp- > Parameters. 読み取りバイトオフセット

_読み取るデータのファイル内での開始バイトオフセットを指定する大きな整数変数。

IrpSp- > Parameters. Read. Key

ターゲットファイルのバイト範囲ロックに関連付けられたキー値。

IrpSp- > Parameters. Read. Length

読み取るデータの長さ (バイト単位)。読み取り操作が成功した場合は、*Irp- > iostatus*が指すIO _ 状態 _ ブロック構造体の情報メンバーに読み取られたバイト数が返されます。

解説

ファイルシステムは、ファイルの最後に、基になるファイルストレージデバイスのセクターサイズの倍数まで、書き込み操作と読み取り

操作を実行します。読み取り前または書き込み前の操作を処理する場合、バッファの割り当てとスワップを行うフィルターでは、割り当てられたバッファのサイズを、関連付けられているデバイスのセクターサイズの倍数まで丸める必要があります。そうでない場合は、基になるファイルシステムから転送されるデータの長さが、バッファに割り当てられた長さを超えます。バッファのスワップの詳細については、「[Swapbuffers のミニサンプル](#)」を参照してください。

関連項目

[CcMdlRead](#)

[CcMdlReadComplete](#)

[IO _ スタックの _ 場所](#)

[IO _ 状態 _ ブロック](#)

[IoGetLocation Entiの場所](#)

[IRP](#)

[IRP _ MJ _ READ \(WDK カーネルリファレンス\)](#)

[IRP _ MJ _ 書き込み](#)

[ZwReadFile](#)

IRP_MJ_設定_EA

2020/04/24 • [Edit Online](#)

送信時

I/O マネージャーは、ファイルの拡張属性を設定するために、IRP_MJ_SET_EA 要求を送信します。

操作: ファイルシステムドライバー

ファイルシステムが拡張属性をサポートしている場合は、ファイルシステムドライバーが要求を処理し、IRP を完了する必要があります。それ以外の場合、ファイルシステムドライバーは `_サポートされていない_EAS_ステータス` を返す必要があります。

操作: ファイルシステムフィルタードライバー

フィルタードライバーは、この IRP をスタック上の次の下位のドライバーに渡す必要があります。

パラメーター

ファイルシステムまたはフィルタードライバーは、指定された IRP で `IoGetLocation` 関数を呼び出して、irp 内の独自の `スタックの場所` へのポインターを取得します。次の一覧には、`Irpsp` として示されています。(IRP は、`irp` として表示されます)。ドライバーは、set extended attributes 要求の処理で、IRP の次のメンバーと IRP スタックの場所に設定されている情報を使用できます。

DeviceObject

ターゲットデバイスオブジェクトへのポインター。

Irps->AssociatedIrp

設定する拡張属性情報が格納されているシステム指定の入力バッファーへのポインター。メソッド_バッファー i/o に使用されます。

Irps->IoStatus

最終的な完了状態と要求された操作に関する情報を受け取る、`IO_ステータス_ブロック` 構造へのポインター。

Irps->MdlAddress

拡張属性情報を受け取る入力バッファーを記述するメモリ記述子リスト (MDL) のアドレス。メソッド_直接 i/o に使用されます。

Irps->UserBuffer

呼び出し元が指定したファイルへのポインター `_完全_EA_` 拡張属性情報を受け取る情報構造化入力バッファーです。I/O ではない_メソッドに使用されます。

Irps->FileObject

DeviceObject に関連付けられているファイルオブジェクトへのポインター。

Irps->FileObject パラメーターには、関連する `fileobject` フィールドへのポインターが含まれています。これは、`ファイル_オブジェクト` 構造体でもあります。ファイル_オブジェクト構造の_の対象となるフィールドは、IRP_MJ の処理中は無効であり、EA_設定されているため、使用できません。

Irps->MajorFunction

EA_設定_IRP_MJ を指定します。

Irps->Parameters. SetEa. 長さ

入力バッファーの長さ (バイト単位)。

「

ファイル_EA_の完全_情報

IO_スタック_の場所

IO_状態_ブロック

IoCheckEaBufferValidity

IoGetCurrentIrpStackLocation

IRP

IRP_MJ_クエリ_EA

IRP _ MJ _ SET の _ 情報 (IFS)

2020/08/04 • [Edit Online](#)

送信時

IRP _ MJ _ SET 情報の _ 要求は、i/o マネージャーおよびその他のカーネルモードドライバーによって送信されます。たとえば、ユーザーモードのアプリケーションが `Setendoffile` などの Microsoft Win32 関数を呼び出したときや、カーネルモードのコンポーネントが `Zwsetinformationfile` を呼び出したときなどに送信できます。

操作: ファイルシステムドライバー

ファイルシステムドライバーは、ファイルオブジェクトを抽出してデコードし、ユーザーファイルまたはディレクトリが開いているかどうかを判断する必要があります。存在する場合は、ファイルシステムドライバーが要求を適切に処理し、IRP を完了する必要があります。

ファイルとディレクトリには、次の種類の情報を設定できます。

FileBasicInformation

FileDispositionInformation

FileLinkInformation (ディレクトリ階層でサイクルを作成できるようにするファイルシステムの場合)

FilePositionInformation

FileRenameInformation

次の種類の情報は、ファイルに対してのみ設定できます。

Fileの情報

FileEndOfFileInformation

FileLinkInformation (NTFS などのファイルシステムの場合は、ディレクトリ階層内でサイクルを作成することはできません)

FileValidDataLengthInformation

操作: ファイルシステムフィルタードライバー

フィルタードライバーは、この IRP をスタック上の次の下位のドライバーに渡す必要があります。

パラメーター

ファイルシステムまたはフィルタードライバーは、指定された IRP で `IoGetLocation` を呼び出して、irp 内の独自の **スタックの場所** へのポインターを取得します。次の一覧には、`irp` として示されています。(IRP は、`irp` として表示されます)。ドライバーは、set file information 要求の処理で、IRP の次のメンバーと IRP スタックの場所に設定されている情報を使用できます。

DeviceObject

ターゲットデバイスオブジェクトへのポインター。

Irp > AssociatedIrp.SystemBuffer

設定するファイルまたはディレクトリの情報が格納されている入力バッファーへのポインター。この情報は、次のいずれかの構造に格納されます。

ファイル _ 割り当て _ 情報

ファイルの _ 基本 _ 情報

ファイルの処理 _ _ 情報

ファイル _ _ の終わりに _ 関するファイル _ 情報

ファイル _ リンク _ 情報

ファイルの _ 位置 _ 情報

ファイル _ 名の変更 _ 情報

ファイルの _ 有効な _ データ _ 長 _ 情報

Irp- > 最後の完了状態と要求された操作に関する情報を受け取るIO _ 状態 _ ブロック構造への *iostatus* ポインター。詳細については、[Zwsetinformationfile](#)への*iostatusblock*パラメーターの説明を参照してください。

Irpsp- > *DeviceObject*に関連付けられているファイルオブジェクトへの *FileObject* ポインター。

Irpsp- > *FileObject*パラメーターには、関連する*FileObject*フィールドへのポインターが含まれています。これは、ファイルオブジェクト構造でも _ あります。ファイルオブジェクト構造の関連性のあるフィールドは、_ IRP MJ SET 情報の処理中は無効であり、_ 使用でき _ ません。

Irpsp- > *MajorFunction* IRP _ MJ SET 情報を指定し _ ます。

Irpsp- > *AdvanceOnly*は、ファイルの終端操作のフラグです。これにより、*fileinformationclassfileendoffileinformation* の場合に、** _ _ _ ファイル _ 情報構造の** *endoffile* member *file END* が使用され == *FileEndOfFileInformation* ます。TRUEの場合、ファイルの新しい有効なデータ長は、現在の有効なデータ長が増加した場合にのみ、*endoffile*から設定されます。FALSEの場合は、新しいファイルサイズが*endoffile*から設定されます。

Irpsp- > *Parameters. SetFile. ClusterCount*はシステム用に予約されています。

Irpsp- > パラメーター。システムで使用するために予約されている *DeleteHandle*。

Irpsp- > パラメーター。 *FileInformationClass*は、ファイルに設定する情報の種類を指定します。次のいずれか:

I	II
File■	ファイルのFILE_ALLOCATION_INFORMATIONを設定します。
FileBasicInformation	ファイルのFILE_BASIC_INFORMATIONを設定します。
FileDispositionInformation	ファイルのFILE_DISPOSITION_INFORMATIONを設定します。
FileEndOfFileInformation	ファイルのFILE_END_OF_FILE_INFORMATIONを設定します。
FileLinkInformation	ファイルのFILE_LINK_INFORMATIONを設定します。
FilePositionInformation	ファイルのFILE_POSITION_INFORMATIONを設定します。
FileRenameInformation	ファイルのFILE_RENAME_INFORMATIONを設定します。

I	II
FileValidDataLengthInformation	ファイル のFILE_VALID_DATA_LENGTH_INFORMATIONを設定 します。

Irpsp->パラメーター。名前の変更またはリンク操作のために使用します。*>AssociatedIrp.Systembuffer->FileNameに完全修飾ファイル名が含まれている場合、またはirp>AssociatedIrp.SysTembuffer->rootdirectory*がNULLでない場合、このメンバーは、操作の対象となるファイルの親ディレクトリのファイルオブジェクトポインターになります。それ以外の場合はNULLになります。

Irpsp->パラメーター。*Irp*>*AssociatedIrp.Systembuffer*が指すバッファの長さをバイト単位で格納します。

Irpsp->パラメーター。置換 *Eifexists* がTRUEに設定されている場合、同じ名前のファイルが既に存在する場合は、指定されたファイルで置き換える必要があることを指定します。指定した名前のファイルが既に存在する場合に名前の変更操作が失敗する場合は、FALSEに設定します。

解説

AdvanceOnlyメンバーは、ディスク上の現在の有効なデータ長をendoffileの新しい有効なデータ長に進めるようにファイルシステムに通知するために、キャッシュマネージャーによってTRUEに設定されます。AdvanceOnlyがFALSEの場合は、endoffileメンバー内の新しいファイルサイズが、現在のファイルサイズより大きいまたは小さい値に設定されています。

関連項目

[ファイル _ 割り当て _ 情報](#)

[ファイルの _ 基本 _ 情報](#)

[ファイルの処理 _ _ 情報](#)

[ファイル _ _ の終わりに _ 関するファイル _ 情報](#)

[ファイル _ リンク _ 情報](#)

[ファイルの _ 位置 _ 情報](#)

[ファイル _ 名の変更 _ 情報](#)

[ファイルの _ 有効な _ データ _ 長 _ 情報](#)

[IO _ スタックの _ 場所](#)

[IO _ 状態 _ ブロック](#)

[Iogetlocation Entiの場所](#)

[IRP](#)

[IRP _ MJ の _ クエリ _ 情報](#)

[ZwSetInformationFile](#)

IRP_MJ_設定_クォータ

2020/04/24 • [Edit Online](#)

送信時

IRP_MJ_SET_QUOTA 要求は、i/o マネージャーによって送信されます。たとえば、ユーザーモードアプリケーションが `diskquotacontrol::` などの Microsoft Win32 メソッドを呼び出した場合に、これを送信できます。

操作: ファイルシステムドライバ

IRP_MJ_SET_QUOTA および IRP_MJ_クエリ_クォータは Windows NT 4.0 に存在しましたが、ファイルシステムでは使用されませんでした。Windows 2000 以降のシステムでは、NTFS でディスククォータをサポートするために使用されます。新しいファイルシステムによるこれらの Irp のサポートはオプションです。

操作: ファイルシステムフィルタードライバ

フィルタードライバは、クォータの動作を明示的にオーバーライドする必要がある限り、この IRP をスタック上の次の下位のドライバに渡す必要があります。

パラメーター

ファイルシステムまたはフィルタードライバは、指定された IRP で `IoGetLocation` を呼び出して、irp 内の独自の **スタックの場所** へのポインターを取得します。次の一覧には、`irpSp` として示されています。(IRP は、`irp` として表示されます)。ドライバは、set quota information 要求の処理で、IRP の次のメンバーと IRP スタックの場所に設定されている情報を使用できます。

DeviceObject

ターゲットデバイスオブジェクトへのポインター。

DeviceObject->フラグ

DO_バッファリング_IO フラグが設定されている場合、呼び出し元はバッファされた i/o メソッドを要求しました。それ以外の場合、呼び出し元は、i/o ではなく、メソッドを要求しました。

Irp->AssociatedIrp

_バッファリング_IO フラグが *DeviceObject->フラグ* で設定されている場合に、中間システムバッファとして使用されるシステム指定のバッファへのポインター。それ以外の場合、このメンバーは NULL に設定されます。

Irp->IoStatus

最終的な完了状態と要求された操作に関する情報を受け取る、**IO_ステータス_ブロック** 構造へのポインター。

Irp->UserBuffer

ボリュームに対して追加または変更するクォータエントリを格納する、呼び出し元が指定したバッファへのポインター。

IrpSp->FileObject

DeviceObject に関連付けられているファイルオブジェクトへのポインター。

IrpSp->FileObject パラメーターには、関連する fileobject フィールドへのポインターが含まれています。これは、ファイル_オブジェクト構造体でもあります。IRP_MJ の処理中に、ファイル_オブジェクト構造の "関連性のある" フィールドが無効になります。このフィールドは、クォータ_クォータを設定して使用する必要があります。

IrpSp->MajorFunction

_クォータを設定_IRP_MJ を指定します。

IrpSp->Parameters. SetQuota. Length

Irp > *UserBuffer*が指すバッファの長さをバイト単位で指定します。

「

[ファイル_クォータの_情報](#)

[IO_スタック_の場所](#)

[IO_状態_ブロック](#)

[IoCheckQuotaBufferValidity](#)

[IoGetCurrentIrpStackLocation](#)

[IRP](#)

[IRP_MJ_クエリ_クォータ](#)

IRP_MJ_設定_セキュリティ

2020/04/24 • [Edit Online](#)

送信時

IRP_MJ_SET_セキュリティ要求は、i/o マネージャーによって送信されます。この要求は、たとえば、ユーザーモードアプリケーションがSetsecurityinfoなどの Win32 関数を呼び出したときに送信できます。

操作: ファイルシステムドライバー

ファイルシステムドライバーは、ファイルオブジェクトを抽出してデコードし、ユーザーファイルまたはディレクトリが開いているかどうかを判断する必要があります。その場合、ドライバーは要求を処理し、IRP を完了する必要があります。それ以外の場合、ドライバーは要求を処理せずに、必要に応じて IRP を完了する必要があります。

操作: ファイルシステムフィルタードライバー

フィルタードライバーは、この IRP をスタック上の次の下位のドライバーに渡す必要があります。

パラメーター

ファイルシステムまたはフィルタードライバーは、指定された IRP で **iogetlocation enti** を呼び出して、irp 内の独自の **スタックの場所** へのポインターを取得します。次の一覧には、*irpsp* として示されています。(IRP は、*irp* として表示されます)。ドライバーは、set security information 要求の処理で、IRP の次のメンバーと IRP スタックの場所に設定されている情報を使用できます。

DeviceObject

ターゲットデバイスオブジェクトへのポインター。

Irp->IoStatus

最後の完了状態と要求された操作に関する情報を受け取る、**IO_ステータス_ブロック** 構造体へのポインター。

IrpSp->FileObject

*DeviceObject*に関連付けられているファイルオブジェクトへのポインター。

IrpSp->FileObject パラメーターには、関連するfileobjectフィールドへのポインターが含まれています。これは、ファイル_オブジェクト構造体でもあります。IRP_MJ の処理中に、ファイル_オブジェクト構造の "関連性のある" フィールドは無効になります。セキュリティで保護されているため、使用しないでください。

IrpSp->MajorFunction

セキュリティ_設定_IRP_MJ を指定します。

IrpSp->パラメーター。SetSecurity. SecurityDescriptor

オブジェクトに割り当てるセキュリティ情報の値を格納している**セキュリティ_記述子**の構造体へのポインター。

IrpSp->パラメーター。SetSecurity. SecurityInformation

セキュリティ記述子に設定するセキュリティ情報を指定する**セキュリティ_情報**型の値。この値には、次のいずれかを指定できます。

SECURITYINFORMATION ㉒

㉒

SECURITYINFORMATION ㊦	㊦
DACL_SECURITY_INFORMATION	オブジェクトの随意アクセス制御リスト (DACL) が設定されていることを示します。WRITE_DAC アクセスが必要です。
GROUP_SECURITY_INFORMATION	オブジェクトのプライマリグループ識別子が設定されていることを示します。WRITE_OWNER アクセスが必要です。
OWNER_SECURITY_INFORMATION	オブジェクトの所有者識別子が設定されていることを示します。WRITE_OWNER アクセスが必要です。
SACL_SECURITY_INFORMATION	オブジェクトのシステム ACL (SACL) が設定されていることを示します。ACCESS_SYSTEM_SECURITY アクセスが必要です。

「

[IO_スタック_の場所](#)

[IO_状態_ブロック](#)

[IoGetCurrentIrpStackLocation](#)

[IRP](#)

[IRP_MJ_クエリ_セキュリティ](#)

[セキュリティ_記述子](#)

[セキュリティ_情報](#)

IRP_MJ_ボリューム情報の設定

2020/04/24 • [Edit Online](#)

送信時

IRP_MJ_SET_VOLUME_INFORMATION 要求は、i/o マネージャーによって送信されます。たとえば、ユーザーモードアプリケーションがSetVolumeLabelなどの Microsoft Win32 関数を呼び出したときに送信できます。

操作: ファイルシステムドライバー

ファイルシステムドライバーは、開いているユーザーボリュームを表すかどうかを判断するために、ファイルオブジェクトを抽出してデコードする必要があります。存在する場合は、ファイルシステムドライバーが適切なボリューム情報を設定し、IRP を完了する必要があります。それ以外の場合、ファイルシステムは、ボリューム情報を設定せずに、必要に応じて IRP を完了する必要があります。

設定できるボリューム情報の種類は、ファイルシステムによって異なりますが、一般的には次の1つ以上を含みます。

FileFsControlInformation

FileFsLabelInformation

FileFsObjectIdInformation

使用可能なすべての情報の種類の一覧については、ntifs の「FS_INFORMATION_クラス列挙体」を参照してください。

操作: ファイルシステムフィルタードライバー

フィルタードライバーは、この IRP をスタック上の次の下位のドライバーに渡す必要があります。

パラメーター

ファイルシステムまたはフィルタードライバーは、指定された IRP で **IoGetLocation** 関数を呼び出して、irp 内の独自の **スタックの場所** へのポインターを取得します。次の一覧には、*irp* として示されています。(IRP は、*irp* として表示されます)。ドライバーは、set volume information 要求の処理で、IRP の次のメンバーと IRP スタックの場所に設定されている情報を使用できます。

DeviceObject

ターゲットデバイスオブジェクトへのポインター。

Irp->AssociatedIrp

設定するボリューム情報の値を格納している入力バッファーへのポインター。この情報は、次のいずれかの構造に格納されます。

ファイル_FS_制御_情報

ファイル_FS_ラベル_情報

ファイル_FS_OBJECTID_情報

Irp->IoStatus 最終的な完了状態と要求された操作に関する情報を受け取る、**IO_ステータス_ブロック** 構造へのポインター。

Irp->FileObject *DeviceObject* に関連付けられているファイルオブジェクトへのポインター。

Irp->FileObject パラメーターには、関連する *fileobject* フィールドへのポインターが含まれています。これは、ファイル_オブジェクト構造体でもあります。IRP_MJ の処理中に、ファイル_オブジェクト構造体の "関連性のある" フィールドは無効になります。_ボリューム_情報を_設定して使用することはできません。

*Irpsp->MajorFunction_*ボリューム_情報を設定_IRP_MJ を指定します。

*Irpsp->Parameters. SetVolume. FsInformationClass*ボリュームに対して設定する情報の種類を指定します。この値には、次のいずれかを指定できます。

VALUE	⌘
FileFsControllInformation	ボリュームのFILE_FS_CONTROL_INFORMATIONを設定します。
FileFsLabelInformation	ボリュームのFILE_FS_LABEL_INFORMATIONを設定します。
FileFsObjectIdInformation	ボリュームのFILE_FS_OBJECTID_INFORMATIONを設定します。

*Irpsp->Parameters. SetVolume. Length Irp>AssociatedIrp*によってポイントされるバッファの長さ (バイト単位)。

「

[ファイル_FS_制御_情報](#)

[ファイル_FS_ラベル_情報](#)

[ファイル_FS_OBJECTID_情報](#)

[IO_スタック_の場所](#)

[IO_状態_ブロック](#)

[IoGetCurrentIrpStackLocation](#)

[IRP](#)

[IRP_MJ_クエリ_ボリューム_情報](#)

[ZwQueryVolumeInformationFile](#)

[ZwSetVolumeInformationFile](#)

IRP _ MJ _ SHUTDOWN (IFS)

2020/08/04 • [Edit Online](#)

送信時

IRP _ MJ _ SHUTDOWN 要求は、システムがシャットダウンされるときに、i/o マネージャーまたはファイルシステムドライバーによって送信されます。

操作: ファイルシステムドライバー

ファイルシステムは、必要なクリーンアップを実行し、状態が SUCCESS の IRP を完了する必要があり _ ます。

操作: ファイルシステムフィルタードライバー

フィルタードライバーは、この IRP をスタック上の次の下位のドライバーに渡す必要があります。

パラメーター

ファイルシステムまたはフィルタードライバーは、指定された IRP で **IoGetLocation** **Enti** を呼び出して、irp 内の独自の **スタックの場所** へのポインターを取得します。次の一覧には、*irpSp* として示されています。(IRP は、*irp* として表示されます)。ドライバーは、シャットダウン要求の処理中に、IRP の次のメンバーと IRP スタックの場所に設定されている情報を使用できます。

DeviceObject

ターゲットデバイスオブジェクトへのポインター。

Irp- > IoStatus

最後の完了状態と要求された操作に関する情報を受け取る **IO _ 状態 _ ブロック** 構造体へのポインター。

IrpSp- > MajorFunction

IRP _ MJ SET SHUTDOWN を指定し _ ます。

関連項目

[IO _ スタックの _ 場所](#)

[IO _ 状態 _ ブロック](#)

[IoGetLocation](#) **Enti**の場所

[IRP](#)

[IRP _ MJ _ SHUTDOWN \(WDK カーネルリファレンス\)](#)

IRP _ MJ _ WRITE (IFS)

2020/08/04 • [Edit Online](#)

送信時

IRP _ MJ _ 書き込み要求は、i/o マネージャーまたはファイルシステムドライバーによって送信されます。この要求は、たとえば、ユーザーモードアプリケーションがWriteFileなどの Microsoft Win32 関数を呼び出したときや、カーネルモードコンポーネントがzwwritefileを呼び出したときに送信できます。

操作: ファイルシステムドライバー

ファイルシステムドライバーは、パラメーターとマイナー関数コードを決定するために、ファイルオブジェクトを抽出してデコードする必要があります。

MDL 書き込み要求の場合、ファイルシステムはマイナー関数コードを確認して、どの操作が要求されているかを判断する必要があります。有効なマイナー関数コードを次に示します。これらは、キャッシュされたファイル i/o に対してのみ使用できます。

- IRP _ _ 完了完了
- IRP _ _ 完全完了 _ MDL
- IRP _ の _ 包括的な _ MDL _ DPC
- IRP _ _ 圧縮済み
- IRP の全 _ _ DPC
- IRP の全 _ _ MDL
- IRP の全 _ _ MDL _ DPC
- IRP _ _ 通常

この IRP を処理する方法の詳細については、Windows Driver Kit (WDK) に含まれている FASTFAT サンプルを調べてください。

操作: ファイルシステムフィルタードライバー

フィルタードライバーは、必要な処理を実行する必要があります。フィルターの性質に応じて、IRP を完了するか失敗させるか、またはスタック上の次の下位のドライバーに渡します。

パラメーター

ファイルシステムまたはフィルタードライバーは、指定された IRP でlogetlocation entiを呼び出して、irp 内の独自のスタックの場所へのポインターを取得します。次の一覧には、irpspとして示されています。(IRP は、irpとして表示されます)。ドライバーは、create 要求を処理するときに、IRP の次のメンバーと IRP スタックの場所に設定されている情報を使用できます。

DeviceObject

ターゲットデバイスオブジェクトへのポインター。

Irp > *AssociatedIrp.SystemBuffer*

_ _ * > DeviceObject*にバッファリング IO フラグが設定されている場合、中間システムバッファとして使用されるシステム指定のバッファへのポインター。それ以外の場合、このメンバーはNULLに設定されます。

Irp - > *IoStatus*

最終的な完了ステータスと要求された操作に関する情報を受け取るIO _ 状態 _ ブロック構造体へのポインター。IRP _ MJ 書き込み要求が失敗した場合 _、ファイルシステムの書き込みディスパッチルーチンはエラー NTSTATUS 値を返し、irp->iostatusの値を返します。情報は定義されていないため、使用しないでください。

Irpf->mdladdress

データの書き込み先のページを記述するメモリ記述子リスト (MDL) のアドレス。

Irpf->FileObject

DeviceObjectに関連付けられているファイルオブジェクトへのポインター。FO _ 同期 _ IO フラグがirpf->Flagsで設定されている場合、ファイルオブジェクトは同期 i/o 用に開かれています。

*Irpf->FileObject*パラメーターには、関連するFileObjectフィールドへのポインターが含まれています。これは、ファイルオブジェクト構造でも _ あります。ファイルオブジェクト構造の MJ fileobjectフィールド _ は、IRP WRITE の処理中は無効であり、_ 使用できません。

Irpf->フラグ

SL _ FORCE _ ダイレクト _ 書き込みフラグが設定されている場合、カーネルモードドライバーは、直接書き込みをブロックしているために通常は書き込むことができないボリューム領域に書き込むことができます。Windows Vista 以降のオペレーティングシステムでは、セキュリティ上の理由により、直接書き込みのブロックが実装されました。このフラグは、ファイルシステムレイヤーとストレージスタックレイヤーの両方でチェックされます。直接書き込みブロックの詳細については、「[ボリュームとディスクへの直接書き込み操作のブロック](#)」を参照してください。SL _ FORCE _ ダイレクト _ 書き込みフラグは、windows Vista 以降のバージョンの windows で使用できます。

Irpf->MajorFunction

IRP MJ WRITE を指定し _ ます。

Irpf->minorfunction

要求される操作を指定し、次のいずれかを含みます。

- IRP _ _ 完了完了
- IRP _ _ 完全完了 _ MDL
- IRP _ の _ 包括的な _ MDL _ DPC
- IRP _ _ 圧縮済み
- IRP の全 _ _ DPC
- IRP の全 _ _ MDL
- IRP の全 _ _ MDL _ DPC
- IRP _ _ 通常

Irpf->Parameters. 書き込み. ByteOffset

_書き込むデータのファイル内での開始バイトオフセットを指定する大きな整数変数。

特定の状況では、このパラメーターに特別な値が含まれる場合があります。次に例を示します。

- 次の条件に該当する場合は、ファイルの現在の終わりを明示的なファイルオフセット値の代わりに使用する必要があることを示します。

Irpf->Parameters.. ByteOffset. LowPart = _ _ _ _ ファイルの終わりへのファイルの書き込み _ と *irpf* の書き込み。
み。 > highpart = -1

Irpf->Parameters. Write. Key

ターゲットファイルのバイト範囲ロックに関連付けられたキー値。

Irps->Parameters.Write.Length

書き込まれるデータの長さ (バイト単位)。書き込み操作が成功すると、書き込まれたバイト数が、`Information _ _ Irp->IoStatus`によって示された IO 状態ブロック構造体の情報メンバーに返されます。

解説

ファイルシステムは、ファイルの最後に、基になるファイルストレージデバイスのセクターサイズの倍数まで、書き込み操作と読み取り操作を実行します。読み取り前または書き込み前の操作を処理する場合、バッファの割り当てとスワップを行うフィルターでは、割り当てられたバッファのサイズを、関連付けられているデバイスのセクターサイズの倍数まで丸める必要があります。そうでない場合は、基になるファイルシステムから転送されるデータの長さが、バッファに割り当てられた長さを超えます。バッファのスワップの詳細については、「[Swapbuffers のミニサンプル](#)」を参照してください。

関連項目

[CcMdlWriteComplete](#)

[Cc/Mdlwrite](#)

[FLT _ IO _ パラメーター _ ブロック](#)

[IO _ スタックの _ 場所](#)

[IO _ 状態 _ ブロック](#)

[IoGetLocation](#) [Enti](#)の場所

[IRP](#)

[IRP _ MJ の _ 読み取り](#)

[IRP _ MJ _ WRITE \(WDK カーネルリファレンス\)](#)

[ZwWriteFile](#)

FSCTL_オーバーレイコントロールコード_追加する

2020/04/24 • [Edit Online](#)

FSCTL_ADD_オーバーレイコントロールコードは、新しい外部バッキングソースをボリュームの名前空間に追加します。このバッキングソースは、Windows イメージフォーマット (WIM) ファイルにすることができます。

この操作を実行するには、次のパラメーターを使用して`Fltfscontrolfile`または`zwfscontrolfile`を呼び出します。

Parameters

]のインスタンス [

`Fltfscontrolfile`のみ。呼び出し元の非透過インスタンスポインター。このパラメーターは必須であり、NULL にすることはできません。

]の FileObject [

`Fltfscontrolfile`のみ。オーバーレイが追加されるボリュームのファイルポインターオブジェクト。このパラメーターは必須であり、NULL にすることはできません。

]の FileHandle [

`Zwfscontrolfile`のみ。オーバーレイが追加されるボリュームのハンドル。このパラメーターは必須であり、NULL にすることはできません。

]の FsControlCode [

操作の制御コード。この操作には、FSCTL_を使用して `_オーバーレイ` を追加してください。

InputBuffer

入力バッファーへのポインター。これには、`WOF_外部_INFO` 構造体が含まれている必要があります。必要に応じて、追加のプロバイダー固有のデータは、`WOF_外部_情報` に含まれます。

]内の InputBufferLength [

`Sizeof(WOF_外部_INFO)` に、追加のプロバイダー入力データのサイズを加えた値に設定します。

OutputBuffer [out]

追加操作からの結果の情報を格納している出力バッファーへのポインター。

OutputBufferLength [out]

`Outputbuffer`が指すバッファーのサイズ。

ステータス ブロック

`Fltfscontrolfile`または`zwfscontrolfile`は、操作が成功した場合に `STATUS_SUCCESS` を返します。それ以外の場合、適切な関数は、次のいずれかの NTSTATUS 値を返す可能性があります。

II	II
STATUS_ACCESS_DENIED	要求元には管理者特権がありません。
STATUS_BUFFER_TOO_SMALL	<code>Outputbuffer</code> によってポイントされ、 <code>outputbufferlength</code> によって指定された出力バッファーの長さが小さすぎます。

II	II
STATUS_INTERNAL_ERROR	要求されたボリュームにアクセスできません。
STATUS_INVALID_DEVICE_REQUEST	バックギングサービスが存在しないか、開始されていません。

注釈

追加されたバックギングソースが Windows Imaging Format (WIM) ファイルの場合、入力バッファーには、[WOF_外部_INFO](#) 構造体の後に[WIM__プロバイダーが含まれ、_オーバーレイ_入力](#)が追加されます。データ. この場合、*Inputbufferlength*はsizeof([WOF_外部_INFO](#)) + SIZEOF([WIM_プロバイダー__オーバーレイ_入力](#)) を追加します。要求が完了すると、*Outputbuffer*が指すデータには、WIM ファイルの新しいデータソース識別子である1つの大きな_整数値が含まれます。

その他のバックギングプロバイダーは、独自の特定の入力パラメーター構造と出力データ型を定義します。

要件

バージョン	Windows 8.1 更新プログラムから使用できます。
Header	Ntifs (Ntifs または Fltkernel .h を含む)

関連項目

[FSCTL__オーバーレイの削除](#)

[FSCTL_SUSPEND_オーバーレイ](#)

[FSCTL_UPDATE_オーバーレイ](#)

[FSCTL_外部_バックギング_取得する](#)

[外部_バックギング_FSCTL_設定](#)

FSCTL_外部_バックアップコントロールコードの削除

2020/04/24 • [Edit Online](#)

FSCTL_DELETE_外部_バックアップコントロールコードは、ファイルと外部バックアッププロバイダー (Windows イメージフォーマット (WIM) プロバイダーや圧縮ファイルプロバイダーなど) との関連付けを解除します。この操作の結果として、バックアップされたファイルの内容全体が読み取り、圧縮解除され、ファイルに書き込まれます。

この操作を実行するには、次のパラメーターを使用して `Fltfscontrolfile` または `zwfscontrolfile` を呼び出します。

Parameters

]のインスタンス [

`Fltfscontrolfile` のみ。呼び出し元の非透過インスタンスポインター。このパラメーターは必須であり、NULL にすることはできません。

]の FileObject [

`Fltfscontrolfile` のみ。バックアップアソシエーションが削除されるファイルのファイルポインターオブジェクト。このパラメーターは必須であり、NULL にすることはできません。

]の FileHandle [

`Zwfscontrolfile` のみ。バックアップアソシエーションが削除されたファイルのハンドル。このパラメーターは必須であり、NULL にすることはできません。

]の FsControlCode [

操作の制御コード。この操作を行うには、FSCTL_を使用して外部__削除します。

InputBuffer

なし。を NULL に設定します。

]内の InputBufferLength [

を0に設定します。

OutputBuffer [out]

なし。を NULL に設定します。

OutputBufferLength [out]

を0に設定します。

ステータス ブロック

`Fltfscontrolfile` または `zwfscontrolfile` は、操作が成功した場合に STATUS_SUCCESS を返します。それ以外の場合、適切な関数は、次のいずれかの NTSTATUS 値を返す可能性があります。

「	「
STATUS_OBJECT_NOT_EXTERNALLY_BACKED	ファイルが外部でサポートされていません。
STATUS_INVALID_DEVICE_REQUEST	バックアップサービスが存在しないか、開始されていません。

[[	[[
STATUS_ACCESS_DENIED	要求元には、ファイルのバックギアソシエーションを削除するアクセス許可がありません。

注釈

削除操作の結果として、ファイルの内容がバックギアソースから読み取られ、ファイル全体がボリュームに書き込まれます。

要件

バージョン	Windows 8.1 更新プログラムから使用できます。
Header	Ntifs (Ntifs または Fltkernel.h を含む)

関連項目

[FltFsControlFile](#)

[ZwFsControlFile](#)

[外部_バックギア_FSCTL_設定](#)

FSCTL_再解析_ポイント制御コードを削除します

2020/04/24 • [Edit Online](#)

FSCTL_DELETE_再解析_ポイント制御コードは、指定されたファイルまたはディレクトリから再解析ポイントを削除します。
FSCTL_DELETE_再解析_ポイントは、ファイルまたはディレクトリを削除しません。

この操作を実行するには、次のパラメーターを使用して **Zwfscontrolfile** を呼び出します。

再解析ポイントを削除するには、ミニフィルターで FSCTL_DELETE_再解析_ポイントではなく **Fltuntagfile** を使用する必要があります。

再解析ポイントおよび FSCTL_DELETE_再解析_ポイント制御コードの詳細については、Microsoft Windows SDK のドキュメントを参照してください。

Parameters

FileHandle

再解析ポイントの削除元となるファイルまたはディレクトリのファイルハンドル。呼び出し元には、ファイルまたはディレクトリへの書き込みアクセス権が必要です。このパラメーターは必須であり、NULLにすることはできません。

FsControlCode

操作の制御コード。この操作では、FSCTL_を使用して_再解析_ポイントを削除します。

InputBuffer

データ_バッファーまたは**再解析_データ_バッファー**構造 **_GUID**へのポインター。この構造体の **ReparseTag** メンバーで指定されるタグは、削除する再解析ポイントのタグと一致する必要があり、**ReparseDataLength** メンバーはゼロである必要があります。また、再解析ポイントがサードパーティ (Microsoft 以外) の再解析ポイントである場合は、再解析_GUID_データ_バッファー構造の **ReparseGuid** メンバーに指定された guid が、削除する再解析ポイントの guid と一致している必要があります。

InputBufferLength

InputBuffer パラメーターによってポイントされるバッファーのサイズ (バイト単位)。再解析_GUID_データ_バッファー構造の場合、この値は **_GUID_データ_バッファー_のヘッダー_サイズ** に正確に再解析する必要があります。データ_バッファー構造の再解析場合、この値は **_データ_バッファー_ヘッダー_サイズ** に正確に再解析する必要があります。

OutputBuffer

この操作では使用されません。をNULLに設定します。

OutputBufferLength

この操作では使用されません。を0に設定します。

ステータス ブロック

Zwfscontrolfile は、次のいずれかのような状態_SUCCESS または適切な NTSTATUS 値を返します。

状態_IO_再解析_データ_無効

指定されたパラメーター値の1つが無効です。これはエラーコードです。

状態_IO_再解析_タグ_無効

呼び出し元によって指定された再解析タグが無効です。これはエラーコードです。

状態_IO_再解析_タグ_不一致

呼び出し元によって指定された再解析タグが、削除する再解析ポイントのタグと一致しませんでした。これはエラーコードです。

ステータス_再解析_属性_競合

再解析ポイントはサードパーティの再解析ポイントであり、呼び出し元によって指定された再解析 GUID が、削除する再解析ポイ

ントの GUID と一致していませんでした。これはエラーコードです。

要件

Header	Ntifs (Ntifs または Fltkernel .h を含む)

関連項目

[IRP_MJ_ファイルの FLT_パラメーター_システム_コントロール](#)

[FltTagFile](#)

[FltUntagFile](#)

[FSCTL__再解析_ポイントを取得します](#)

[FSCTL_設定_再解析_ポイント](#)

[IRP_MJ_ファイル_システム_コントロール](#)

[IsReparseTagMicrosoft](#)

[IsReparseTagNameSurrogate](#)

[データ_バッファの再解析_](#)

[_GUID_データ_バッファの再解析](#)

[ZwFsControlFile](#)

FSCTL_マウント解除_ボリューム制御コード

2020/04/24 • [Edit Online](#)

ボリュームが使用されているかどうかに関係なくボリュームのマウントを解除しようとすると、FSCTL_ボリューム制御コードがマウント解除されます。

この操作を実行するには、次のパラメーターを使用して`Fltfscontrolfile`または`zwfscontrolfile`を呼び出します。

パラメーター

Instance

`Fltfscontrolfile`のみ。呼び出し元の非透過インスタンスポインター。このパラメーターは必須であり、NULLにすることはできません。

ファ

`Fltfscontrolfile`のみ。マウントを解除するボリュームを指定するファイルポインターオブジェクト。このパラメーターは必須であり、NULLにすることはできません。

]の FileHandle [

`Zwfscontrolfile`のみ。マウントを解除するボリュームのファイルハンドル。このパラメーターは必須であり、NULLにすることはできません。

]の FsControlCode [

操作の制御コード。この操作には、FSCTL_を使用して_ボリュームのマウントを解除してください。

InputBuffer

この操作では使用されません。をNULLに設定します。

]内の InputBufferLength [

この操作では使用されません。を0に設定します。

OutputBuffer [out]

この操作では使用されません。をNULLに設定します。

]の OutputBufferLength [

この操作では使用されません。を0に設定します。

状態ブロック

`Fltfscontrolfile`または`ZWFSCONTROLFILE`は、STATUS_SUCCESS または適切な NTSTATUS 値を返します。

解説

ボリューム制御コード __マウント解除すると、他のプロセスがボリュームを使用しているかどうかに関係なく、ボリュームのマウントを解除しようとします。ボリュームのロックを保持していない場合、これらのプロセスに対して予期しない結果が生じる可能性があります。ボリュームのロックの詳細については、「[FSCTL_LOCK_volume](#)」を参照してください。

オペレーティングシステムは、マウント解除されたボリュームを検出しません。マウント解除されたボリュームにアクセスしようとすると、オペレーティングシステムはボリュームのマウントを試みます。たとえば、`Getlogicaldrives`を呼び出すと、マウント解除されたボリュームをマウントするオペレーティングシステムがトリガーされます。

`Zwfscontrolfile`に渡される *FileHandle* ハンドルは、直接アクセスするために開かれたボリュームへのハンドルである必要があります。ボリュームハンドルを取得するには、*Objectattributes* パラメーターを \\ の形式の *ObjectName* に設定して `zwcreatefile` を呼び出します。|x: ここで、x はボリュームのドライブ文字、フロッピーディスクです。ドライブ、または CD-ROM ドライブ。また、アプリ

ケーションでは、Zwcreatefileの共有 *アクセス*パラメーターで書き込みフラグ__共有_ファイルを共有_指定する必要があります。

指定されたボリュームがシステムボリュームであるか、またはページファイルを含んでいる場合、操作は失敗します。

指定されたボリュームが別のプロセスによってロックされている場合、操作は失敗します。

前提条件

ヘッダー	Ntifs

関連項目

- [FltCreateFile](#)
- [FltFsControlFile](#)
- [ZwCreateFile](#)
- [ZwFsControlFile](#)

FSCTL_複製_エクステント_TO_ファイル_コントロールのコード例

2020/04/24 • [Edit Online](#)

FSCTL_複製_エクステント_TO_ファイル_EX コントロール コード ファイルの範囲をコピーするファイル システムに指示アプリケーションの代わりにバイト数。リンク先のファイルは、同じか、ソース ファイルと異なる可能性があります。

この操作を実行するには、呼び出し、**DeviceIoControl** 関数は次のパラメーター。

```
BOOL
WINAPI
DeviceIoControl( (HANDLE)      hDevice,          // handle to device
                 (DWORD)       FSCTL_DUPLICATE_EXTENTS_TO_FILE_EX, // dwIoControlCode
                 (LPDWORD)      lpInBuffer,      // input buffer
                 (DWORD)        nInBufferSize,   // size of input buffer
                 (LPDWORD)      lpOutBuffer,     // output buffer
                 (DWORD)        nOutBufferSize,  // size of output buffer
                 (LPDWORD)      lpBytesReturned, // number of bytes returned
                 (LPOVERLAPPED) lpOverlapped );  // OVERLAPPED structure
```

パラメーター

hDevice [で]

デバイスへのハンドル。デバイス ハンドルを取得する呼び出し、**CreateFile** 関数。

dwIoControlCode [in]

操作の制御コード。使用 **FSCTL_複製_エクステント_TO_ファイル_EX** この操作。

lpInBuffer

ポインタを複製_エクステント_データ_EXに範囲をコピーするソース ファイル、ソース バイトの範囲、および変換先ファイルを指定する構造体のオフセットします。

nInBufferSize [in]

(バイト単位)、入力バッファのサイズ。

lpOutBuffer [アウト]

この操作で使用できません。NULL に設定します。

nOutBufferSize [in]

この操作で使用できません。ゼロ (0) に設定します。

lpBytesReturned [out]

(バイト単位)、出力バッファに格納されているデータのサイズを受け取る変数へのポインタ。

出力バッファが小さすぎる場合は、呼び出しが失敗した、**GetLastError** 返しますエラー_不十分_バッファ、および *lpBytesReturned* は 0 です。

場合 *lpOverlapped* は NULL、*lpBytesReturned* することはできません NULL します。でもときに操作を返しません出力データと *lpOutBuffer* は NULL、**DeviceIoControl** 活用 *lpBytesReturned* します。このような操作の値の後に *lpBytesReturned* は意味がありません。

場合 *lpOverlapped* ない NULL、*lpBytesReturned* できる NULL します。このパラメーターがない場合 NULL 操作は、そのデータを返します *lpBytesReturned* オーバー ラップ処理が完了するまでは無意味です。返されるバイト数を取得する

[GetOverlappedResult](#)します。場合、*hDevice*パラメーターは、I/O 完了ポートに関連付け、呼び出すことによって返されるバイト数を取得する [GetQueuedCompletionStatus](#)します。

lpOverlapped [in]

ポインター、[OVERLAPPED](#) 構造体。

場合 *hDevice*を指定せずに開かれたファイル_フラグ_[OVERLAPPED](#)、*lpOverlapped*は無視されます。

場合 *hDevice*で開かれた、ファイル_フラグ_[OVERLAPPED](#)フラグは、操作がオーバー ラップ (非同期) 操作として実行します。この場合、*lpOverlapped*有効 をポイントする必要があります [OVERLAPPED](#) イベント オブジェクトへのハンドルを含む構造体。それ以外の場合、関数は、予測できない方法で失敗します。

重複した操作は、[DeviceIoControl](#) を即座に返します、操作が完了したときに、イベント オブジェクトがシグナル状態とします。それ以外の場合、操作が完了したか、エラーが発生するまでには、この関数は返されません。

戻り値

操作が正常に完了した場合 [DeviceIoControl](#) 0 以外の値を返します。

操作は、障害が発生したり、保留中で [DeviceIoControl](#) は 0 を返します。拡張エラー情報を取得するには呼び出します [GetLastError](#)します。

注釈

この操作で重複 I/O の影響の「解説」を参照してください、[DeviceIoControl](#) トピック。

必要条件

Header	WinIoctl.h;Ntifs.h

関連項目

[DeviceIoControl](#)

FSCTL_ENUM_外部_バックアップコントロールコード

2020/04/24 • [Edit Online](#)

FSCTL_ENUM_外部_バックアップコントロールコードは、バックアップソースを持つボリューム上のファイルの列挙を開始または続行します。要求が正常に完了するたびに、バックアップされたファイルの識別子が返されます。サポートされているすべてのファイルは、どの外部プロバイダーによってバックアップされているかに関係なく列挙されます。連続するFSCTL_ENUM_、ボリューム上のすべてのバックアップされたファイルを列挙するために、外部_バックアップ要求が必要です。

この操作を実行するには、次のパラメーターを使用して`Fltfscontrolfile`または`zwfscontrolfile`を呼び出します。

Parameters

]のインスタンス [

`Fltfscontrolfile`のみ。呼び出し元の非透過インスタンスポインター。このパラメーターは必須であり、NULLにすることはできません。

]の FileObject [

`Fltfscontrolfile`のみ。マウントを解除するボリュームを指定するファイルポインターオブジェクト。このパラメーターは必須であり、NULLにすることはできません。

]の FileHandle [

`Zwfscontrolfile`のみ。マウントを解除するボリュームのファイルハンドル。このパラメーターは必須であり、NULLにすることはできません。

]の FsControlCode [

操作の制御コード。この操作を行うには、FSCTL_ENUM_外部_を使用します。

InputBuffer

なし。をNULLに設定します。

]内の InputBufferLength [

を0に設定します。

OutputBuffer [out]

出力バッファーへのポインター。1つ以上のWOF_外部_ファイル_ID構造体を受け取るのに十分なサイズのサイズが必要です。

OutputBufferLength [out]

Outputbufferが指す出力バッファーのサイズ。Outputbufferlength >= sizeof(WOF_外部_ファイル_ID) である必要があります。

長さが [out を返しました]

正常に完了したときにOutputbufferに書き込まれるバイト数を指定します。

ステータス ブロック

`Fltfscontrolfile`または`zwfscontrolfile`は、操作が成功した場合に STATUS_SUCCESS を返します。それ以外の場合、適切な関数は、次のいずれかの NTSTATUS 値を返す可能性があります。

「	「
STATUS_ACCESS_DENIED	要求元には管理者特権がありません。

II	II
STATUS_BUFFER_TOO_SMALL	<i>Outputbuffer</i> によってポイントされ、 <i>outputbufferlength</i> によって指定された出力バッファの長さが小さすぎます。
STATUS_NO_MORE_FILES	ボリューム上のファイルにバッキングソースが1つありません。
STATUS_INTERNAL_ERROR	要求されたボリュームにアクセスできません。
STATUS_INVALID_DEVICE_REQUEST	バッキングサービスが存在しないか、開始されていません。

注釈

*Outputbuffer*に返されるWOF_外部_ファイル_ID構造には、サポートされるファイルの一意のファイル識別子が含まれます。構造体は、ntifs で次のように定義されています。

```
typedef struct _WOF_EXTERNAL_FILE_ID {
    FILE_ID_128 FileId;
} WOF_EXTERNAL_FILE_ID, *PWOFF_EXTERNAL_FILE_ID;
```

外部_バッキング要求_FSCTL_列挙型は、バッキングソースを持つボリューム上の各ファイルの識別子を取得するために、連続して発行されます。すべてのファイルが列挙されると、状態_[_]_ファイルの状態コードが返されます。

要件

バージョン	Windows 8.1 更新プログラムから使用できます。
Header	Ntifs (Ntifs または Fltkernel .h を含む)

関連項目

[FltFsControlFile](#)

[ZwFsControlFile](#)

[FSCTL_外部_バッキング_取得する](#)

FSCTL_ENUM_オーバーレイコントロールコード

2020/04/24 • [Edit Online](#)

FSCTL_ENUM_オーバーレイコントロールコードは、指定されたボリュームのバックアッププロバイダーからすべてのデータソースを列挙します。

この操作を実行するには、次のパラメーターを使用して`Fltfscontrolfile`または`zwfscontrolfile`を呼び出します。

Parameters

]のインスタンス [

`Fltfscontrolfile`のみ。呼び出し元の非透過インスタンスポインター。このパラメーターは必須であり、NULLにすることはできません。

]の FileObject [

`Fltfscontrolfile`のみ。マウントを解除するボリュームを指定するファイルポインターオブジェクト。このパラメーターは必須であり、NULLにすることはできません。

]の FileHandle [

`Zwfscontrolfile`のみ。マウントを解除するボリュームのファイルハンドル。このパラメーターは必須であり、NULLにすることはできません。

]の FsControlCode [

操作の制御コード。この操作には、FSCTL_を使用して_オーバーレイを削除してください。

InputBuffer

入力バッファーへのポインター。これには、`WOF_外部_INFO`構造体が含まれている必要があります。

]内の InputBufferLength [

Sizeof(WOF_外部_INFO) に設定します。

OutputBuffer [out]

ボリュームをバックアップするデータソースの_オーバーレイ_エントリ構造を1つ以上持つ WIM_プロバイダーを受け取る出力バッファーへのポインター。

OutputBufferLength [out]

Outputbufferによってポイントされるバッファーのサイズ (バイト単位)。

長さが [out を返しました]

正常に完了したときにOutputbufferに書き込まれるバイト数を指定します。

ステータス ブロック

`Fltfscontrolfile`または`zwfscontrolfile`は、操作が成功した場合に STATUS_SUCCESS を返します。それ以外の場合、適切な関数は、次のいずれかの NTSTATUS 値を返す可能性があります。

「	「
STATUS_ACCESS_DENIED	要求元には管理者特権がありません。
STATUS_BUFFER_TOO_SMALL	Outputbufferによってポイントされ、outputbufferlengthによって指定された出力バッファーの長さが小さすぎます。

II	II
STATUS_INTERNAL_ERROR	要求されたボリュームにアクセスできません。
STATUS_INVALID_DEVICE_REQUEST	バックギングサービスが存在しないか、開始されていません。

注釈

WIM プロバイダーのデータソースを列挙すると、出力バッファーにwim_プロバイダーの配列_オーバーレイ_エントリ構造が格納されます。出力バッファーのサイズは、すべてのオーバーレイエントリを格納するのに十分な大きさである必要があります。または、呼び出しによってステータス_バッファー__小さすぎます。

追加のバックギングプロバイダーは、独自の特定の列挙構造を定義します。

要件

バージョン	Windows 8.1 更新プログラムから使用できます。
Header	Ntifs (Ntifs または Fltkernel .h を含む)

関連項目

[FltFsControlFile](#)

[ZwFsControlFile](#)

[FSCTL__オーバーレイの追加](#)

[WOF_外部_情報](#)

FSCTL_FILE_LEVEL_TRIM control code

2020/04/24 • [Edit Online](#)

FSCTL_file_LEVEL_trimコントロールコードには、ファイル内のデータ範囲をトリミングするためのメソッドが用意されています。ファイルのトリミング範囲は、基になるストレージデバイスに変換されます。これにより、リソースの組織を最適化し、アクセスのパフォーマンスを向上させることができます。FSCTL_FILE_LEVEL_TRIM要求を使用すると、仮想ディスク上で解放されたデータ範囲に対応するために物理記憶域をトリミングしながら、仮想ディスクファイルを固定サイズで割り当てたままにすることができます。

この操作を実行するには、次のパラメーターを使用して`Fltfscontrolfile`または`zwfscontrolfile`を呼び出します。

パラメーター

]のインスタンス [

`Fltfscontrolfile`のみ。呼び出し元の非透過インスタンスポインター。このパラメーターは必須であり、NULLにすることはできません。

]の FileObject [

`Fltfscontrolfile`のみ。マウントを解除するボリュームを指定するファイルポインターオブジェクト。このパラメーターは必須であり、NULLにすることはできません。

]の FileHandle [

`Zwfscontrolfile`のみ。マウントを解除するボリュームのファイルハンドル。このパラメーターは必須であり、NULLにすることはできません。

]の FsControlCode [

操作の制御コード。この操作には、FSCTL_使用して_オーバーレイを削除してください。

InputBuffer

FSCTL_ファイル_LEVEL_TRIM構造体を含む必要がある入力バッファーへのポインター。

]内の InputBufferLength [

ファイルのトリム範囲の配列を含むtrim 構造_ファイル_レベルへのポインター。

OutputBuffer [out]

省略可能なファイル_レベルへのポインター_trim 操作の結果を受け取る出力構造をします。

OutputBufferLength [out]

*Outputbuffer*パラメーターが指すバッファーのサイズ (バイト単位)。ファイル_レベル_trim_出力がoutputbufferに含まれる場合、この値は少なくともsizeof(ファイル_レベル_trim_出力)である必要があります。それ以外の場合は0に設定されます。

状態ブロック

`Fltfscontrolfile`または`ZWFSCONTROLFILE`は、STATUS_SUCCESS または次のいずれかの値を返します。

「	「
STATUS_INVALID_PARAMETER	トリミングするファイルが圧縮または暗号化されているか、入力バッファーまたは出力バッファーの長さが無効であるか、またはトリム範囲が指定されていません。
STATUS_INSUFFICIENT_RESOURCES	内部リソースの割り当てに失敗しました。

II	II
STATUS_FILE_LOCK_CONFLICT	トリム範囲は、以前にロックされていたバイト範囲の一部です。
STATUS_VOLUME_DISMOUNTED	ファイルが存在するボリュームがマウントされていません。
STATUS_PURGE_FAILED	トリム範囲のキャッシュの消去に失敗しました。
STATUS_NO_RANGES_PROCESSED	Trim 範囲配列内の範囲が処理されませんでした。

解説

特定の記憶装置でトリムを実行すると、将来の書き込みパフォーマンスが大幅に向上する可能性があります。また、Trim では、仮想プロビジョニングされたストレージシステム内の割り当てプールにリソースが返されます。仮想ディスク上のファイルが削除されても、仮想ディスクファイル自体のサイズは変更されません。仮想ディスク上で解放されたデータ範囲は、仮想ディスクファイルが置かれている物理記憶域上では切り捨てられません。仮想ディスクデバイスは、FSCTL_ファイル_レベル_TRIM要求を使用して、仮想ディスクファイル内の特定のデータ範囲を物理記憶装置でトリミングできることをファイルシステムに通知することができます。次に、ファイルシステムは、物理記憶域にトリム要求を発行します。FSCTL_FILE_LEVEL_TRIM要求は、データベースまたはメモリスワップファイルを管理するサービスアプリケーションによって発行されることもあります。

FSCTL_file_LEVEL_trimコントロールコードは、ファイルの選択されたバイト範囲を記憶装置からトリミングしようとしています。バイト範囲は、[ファイル_レベル_TRIM](#)構造体の範囲配列に含まれています。範囲の配列に含まれているのは、[_範囲構造_トリム](#)する1つ以上のファイル_レベルです。

範囲配列に重複する範囲を含めることは、必ずしもエラー条件であるとは限りません。これは、基になるストレージでエクステント処理がどのように処理されるかによって異なります。

切り捨てられた範囲は、ファイルシステムキャッシュからページとして削除されます。キャッシュページのサイズと一致させるために、トリミング範囲の長さはページ_サイズの倍数に調整されます。また、トリミング範囲のオフセットがページの境界から始まらない場合は、次のページの境界に合わせて調整されます。これらの制約を使用すると、オフセットがページに合わせられていない場合、または長さがページサイズが複数でない場合に、トリム範囲の長さが減少します。元の長さが2ページ未満でオフセットがページに合わせられていない場合、トリム範囲の長さは0に短縮される可能性があります。

Trim 範囲が指定されている場合、またはファイルの終わり (EOF) を超えてページが調整された場合、範囲は無視されます。ただし、EOF より前に配置され、長さが EOF を超える範囲オフセットは、ページサイズが複数 <= EOF に調整されます。

圧縮または暗号化されたファイル ([属性_フラグ_圧縮_マスク](#)または[属性_フラグ_暗号化](#)された属性が設定されているファイル) では、ファイルレベルのトリミングはサポートされていません。

ファイルのトリミングは、トランザクションの外部で実行されます。トリム操作をロールバックできません。

スパースファイル ([属性__フラグ](#)が設定されているファイル) を使用すると、ファイルの未割り当て部分のトリム範囲は無視されます。

出力バッファに含まれている場合、numrangesprocessedされた[ファイル_レベル_trim_出力](#)は、正常に処理されたトリミング範囲の数を示します。Trim 範囲の処理中にエラーが発生した場合、Numrangesprocessedは、残りの未処理の範囲の開始インデックスを指定します。これは、[ファイル_レベル](#)のnumofメンバーで終わり_trim-1 です。

前提条件

バージョン	Windows 8 以降のバージョンの Windows で使用できます。

ヘッダー	Ntifs (Ntifs または Fltkernel .h を含む)

関連項目

[ファイル_レベル_TRIM](#)

[ファイル_レベル_TRIM_出力](#)

[ファイル_レベル_TRIM_範囲](#)

[FltCreateFile](#)

[FltFsControlFile](#)

[ZwCreateFile](#)

[ZwFsControlFile](#)

FSCTL__SID 制御コードによって__ファイルを検索する

2020/04/24 • • [Edit Online](#)

_SID 制御コードによって_された_ファイルを検索_は、指定した SID に一致するクリエイターと所有者があるファイルをディレクトリで検索します。

この操作を実行するには、ミニフィルタードライバーは、次のパラメーターを使用して`Fltfscontrolfile`を呼び出します。ファイルシステム、リダイレクター、およびレガシファイルシステムフィルタードライバーは、次のパラメーターを使用して`zwfscontrolfile`を呼び出します。

Parameters

ファ

`Fltfscontrolfile`のみ。検索するディレクトリのファイルオブジェクトポインター。このパラメーターは必須であり、NULLにすることはできません。

FileHandle

`Zwfscontrolfile`のみ。検索するディレクトリのファイルハンドル。このパラメーターは必須であり、NULLにすることはできません。

FsControlCode

操作の制御コード。FSCTL を使用して、この操作の_SID によって_ファイルを検索_します。

InputBuffer

_SID_データ構造によって検索_によって記述された入力バッファーへのポインター。_SID_データ構造による検索_は、次のように定義されています。

```
typedef struct {
    DWORD    ;
    SID      ;
} FIND_BY_SID_DATA, *PFIND_BY_SID_DATA;
```

属する

一度

検索を再開するかどうかを示します。最初の呼び出しでこのメンバーを1に設定して、検索がルートから開始されるようにする必要があります。後続の呼び出しでは、このメンバーを0に設定して、検索が停止した時点で再開されるようにする必要があります。

Sid

作成者と所有者を指定するSID型の構造体。

InputBufferLength

*InputBuffer*のバッファーの長さ (バイト単位)。

OutputBuffer

各ファイルの完全修飾パス名を受け取る_SID_出力構造体を使用して、クワッドアラインされた検索_の呼び出し元が割り当てた配列へのポインター。FIND_BY_SID_の出力構造は次のように定義されています。

```
typedef struct _FIND_BY_SID_OUTPUT {  
    DWORD    ;  
    DWORD    ;  
    DWORD    ;  
    WCHAR    [1];  
} FIND_BY_SID_OUTPUT, *PFIND_BY_SID_OUTPUT;
```

属する

NextEntryOffset

次のレコードに到達するためにスキップする必要があるバイト数。値が0の場合は、これが最後のレコードであることを示します。

Fileindex

ファイルのインデックス。

FileNameLength

ファイル名のサイズ (バイト単位)。

ファイル名

ファイル名を指定する null で終わる文字列。

OutputBufferLength

Outputbuffer パラメーターによって示されるバッファで返されるデータのサイズ (バイト単位)。

注釈

[Fltfscontrolfile](#)と[zwfscontrolfile](#)が `_SID` 制御コードによって `__ファイルを検索` 場合、これらのルーチンはボリューム上のすべてのファイルとディレクトリをチェックします。この操作は、検索するディレクトリが非常に小さい場合でも、ボリュームに多数のファイルがあると、処理が遅くなることがあります。

関連項目

[FltFsControlFile](#)

[SID](#)

[ZwFsControlFile](#)

FSCTL_GET_BOOT_AREA_INFO 制御コードを取得する

2020/04/24 • [Edit Online](#)

FSCTL_GET_BOOT_AREA_INFO 制御コードは、ボリュームのブートセクターの場所を取得します。

この操作を実行するには、次のパラメーターを使用して `Fltfscontrolfile` 関数または `zwfscontrolfile` 関数を呼び出します。

Parameters

ファ

`Fltfscontrolfile` のみ。ブート情報を取得する FSCTL_GET_BOOT_AREA_INFO のボリュームのファイルオブジェクトポインター。このパラメーターは必須であり、NULL にすることはできません。

FileHandle

`Zwfscontrolfile` のみ。ブート情報を取得する、FSCTL_GET_BOOT_AREA_INFO のボリュームのファイルハンドル。このパラメーターは必須であり、NULL にすることはできません。

「ボリューム_名前」のアクセス権を管理するには、このハンドルを SE で開く必要があります。詳細については、「[ファイルセキュリティとアクセス権](#)」を参照してください。

FsControlCode

操作の制御コード。この操作については、FSCTL_GET_BOOT_AREA_INFO を取得してください。

InputBuffer

この操作では使用されません。を NULL に設定します。

InputBufferLength

この操作では使用されません。を 0 に設定します。

OutputBuffer

ボリュームのブートセクターの場所を受け取る、BOOT_AREA_INFO 構造体へのポインター。

OutputBufferLength

出力バッファのサイズ (バイト単位)。

ステータス ブロック

`Fltfscontrolfile` または `zwfscontrolfile` は、次のいずれかのような適切な NTSTATUS 値を返します。

NTSTATUS	説明
STATUS_SUCCESS	操作は成功しました。OutputBuffer には、BOOT_AREA_INFO 構造体へのポインターが含まれています。
STATUS_INVALID_PARAMETER	パラメーターが有効ではありませんでした。たとえば、使用されるハンドルが有効なボリュームハンドルではありません。
STATUS_BUFFER_TOO_SMALL	OutputBuffer は、結果に対して十分な大きさではありません。バッファに情報が書き込まれていません。
STATUS_ACCESS_DENIED	ユーザーには、SE_MANAGE_VOLUME アクセス権がありません。

注釈

FSCTL__ブート_領域の取得_情報制御コードは、FastFAT および exFAT デバイスで使用できます。この機能は、フラッシュドライブなどのデバイスでの BitLocker の使用をサポートしています。

要件

バージョン	Windows Server 2008 R2、Windows 7
Header	Ntifs

関連項目

[DeviceIoControl](#)

FSCTL_外部_バッキングコントロールコード_取得する

2020/04/24 • [Edit Online](#)

FSCTL_GET_外部_バッキングコントロールコードは、外部のバッキングプロバイダーからファイルのバッキング情報を取得します。バッキングプロバイダーには、Windows イメージフォーマット (WIM) プロバイダーまたは個々の圧縮ファイルプロバイダーが含まれます。外部でサポートされるファイルのコンテンツは、照会されたファイルを含むボリューム以外のボリューム上に存在する可能性があります。

この操作を実行するには、次のパラメーターを使用して `Fltfscontrolfile` または `zwfscontrolfile` を呼び出します。

Parameters

]のインスタンス [

`Fltfscontrolfile` のみ。呼び出し元の非透過インスタンスポインター。このパラメーターは必須であり、NULL にすることはできません。

]の FileObject [

`Fltfscontrolfile` のみ。バッキング情報を照会する対象のファイルのファイルポインターオブジェクト。このパラメーターは必須であり、NULL にすることはできません。

]の FileHandle [

`Zwfscontrolfile` のみ。バッキング情報のクエリ対象となるファイルのハンドル。このパラメーターは必須であり、NULL にすることはできません。

]の FsControlCode [

操作の制御コード。この操作を行うには、FSCTL_使用して外部__取得します。

]の InputBuffer [

なし。をNULLに設定します。

]内の InputBufferLength [

を0に設定します。

OutputBuffer [out]

出力バッファーへのポインター。これには、`WOF_外部_INFO` 構造体の後にプロバイダーデータが格納されるのに十分なサイズが必要です。WIM によってサポートされるファイルの場合、`WOF_外部_情報` の後に `wim_プロバイダー_外部_情報` 構造が続きます。個別に圧縮されたファイルの場合、`WOF_外部_情報` の後に `ファイル_プロバイダー_外部_情報_V1` 構造が続きます。

OutputBufferLength [out]

`Outputbuffer` が指すバッファーのサイズ (バイト単位)。

返される長さ

正常に完了したときに `Outputbuffer` に書き込まれるバイト数を指定します。

ステータス ブロック

`Fltfscontrolfile` または `zwfscontrolfile` は、操作が成功した場合に `STATUS_SUCCESS` を返します。それ以外の場合、適切な関数は、次のいずれかの NTSTATUS 値を返す可能性があります。

II	II
STATUS_OBJECT_NOT_EXTERNALLY_BACKED	ファイルが外部でサポートされていません。
STATUS_INVALID_DEVICE_REQUEST	バックアップサービスが存在しないか、開始されていません。

注釈

更新するデータソースのバックアッププロバイダーが WIM ファイルの場合、出力バッファーには、外部 [_INFO](#) 構造体の後に [wim_プロバイダー_外部_info](#) 構造体が含まれます。 *Outputbufferlength* は、少なくとも `sizeof(WOF_外部_info) + sizeof(WIM_PROVIDER_外部_info)` である必要があります。バックアッププロバイダーが個別に圧縮されたファイルの場合、出力バッファーには、外部 [_情報_V1](#) 構造を [ファイル_プロバイダーの __外部の情報](#) 構造が格納されます。

要件

バージョン	Windows 8.1 更新プログラムから使用できます。
Header	Ntifs (Ntifs または Fltkernel .h を含む)

関連項目

[FltFsControlFile](#)

[ZwFsControlFile](#)

[外部_バックアップ_FSCTL_設定](#)

[WIM_プロバイダー_外部_情報](#)

[WOF_外部_情報](#)

FSCTL_再解析_ポイント制御コードを取得します

2020/04/24 • [Edit Online](#)

FSCTL_GET_再解析_ポイント制御コードは、指定したファイルまたはディレクトリに関連付けられている再解析ポイントデータを取得します。

この操作を実行するには、次のパラメーターを使用して`Fltfscontrolfile`または`zwfscontrolfile`を呼び出します。

再解析ポイントおよび FSCTL_GET_再解析_ポイント制御コードの詳細については、Microsoft Windows SDK のドキュメントを参照してください。

Parameters

ファ

`Fltfscontrolfile`のみ。再解析ポイントデータの取得元となるファイルまたはディレクトリのファイルオブジェクトポインター。このパラメーターは必須であり、NULLにすることはできません。

FileHandle

`Zwfscontrolfile`のみ。再解析ポイントデータの取得元となるファイルまたはディレクトリのファイルハンドル。このパラメーターは必須であり、NULLにすることはできません。

FsControlCode

操作の制御コード。この操作を行うには、FSCTL_を使用して_再解析_ポイントを取得してください。

InputBuffer

この操作では使用されません。をNULLに設定します。

InputBufferLength

この操作では使用されません。を0に設定します。

OutputBuffer

再解析ポイントデータを受け取るバッファー構造体 `_データ_バッファー` または `再__解析_GUID` を指すポインター。

OutputBufferLength

`Outputbuffer!`パラメーターが指すバッファーのサイズ (バイト単位)。再解析_GUID_データ_バッファー構造体の場合、この値には少なくとも、_データ_バッファー_のデータ_サイズ、および予想されるユーザー定義データのサイズに対する再解析_GUID を指定する必要があります。、_再解析_データ_バッファー_サイズの最大値以下である必要があります。再解析_データ_バッファー構造体の場合、この値は、少なくともデータ_バッファー_のヘッダー_サイズに加えて、予想されるユーザー定義データのサイズに加え、最大値以下である必要があります_再解析_データ_バッファー_サイズ。

ステータス ブロック

`Fltfscontrolfile`または`zwfscontrolfile`は、次のいずれかのような状態_SUCCESS または適切な NTSTATUS 値を返します。

⌞	⌞

II	II
STATUS_BUFFER_OVERFLOW	<i>Outputbuffer</i> パラメーターが指すバッファは、REPARSE_GUID_DATA_BUFFER または REPARSE_DATA_BUFFER 構造体の固定部分を保持するのに十分な大きさですが、ユーザー定義データは保持できません。この場合は、再解析ポイントデータの固定部分のみが <i>outputbuffer</i> バッファに返されます。 Fltfscontrolfile に返された長さのパラメーターは、返されたデータの実際の長さをバイト単位で受け取ります。これは警告コードです。
STATUS_BUFFER_TOO_SMALL	<i>Outputbuffer</i> パラメーターが指すバッファが、再解析ポイントデータを保持するのに十分な大きさではありません。 Fltfscontrolfile に返された長さ(または、 lowfscontrolfile への <i>iosstatus</i> パラメーターの <code>Length</code> メンバー) は、必要なバッファサイズを受け取ります。この場合、再解析ポイントデータは返されません。これはエラーコードです。
STATUS_IO_REPARSE_DATA_INVALID	指定されたパラメーター値の1つが無効です。これはエラーコードです。
STATUS_NOT_A_REPARSE_POINT	ファイルまたはディレクトリが再解析ポイントではありません。これはエラーコードです。

要件

Header	Ntifs (Ntifs または Fltkernel.h を含む)

関連項目

[FLT_コールバック_データ](#)

[IRP_MJ_ファイルの FLT_パラメーター_システム_コントロール](#)

[FLT_タグ_データ_バッファ](#)

[FltFsControlFile](#)

[FltTagFile](#)

[FltUntagFile](#)

[FSCTL_再解析_ポイントを削除します](#)

[FSCTL_設定_再解析_ポイント](#)

[IRP_MJ_ファイル_システム_コントロール](#)

[IsReparseTagMicrosoft](#)

[IsReparseTagNameSurrogate](#)

[データ_バッファの再解析_](#)

[_GUID_データ_バッファの再解析](#)

FSCTL_取得_取得_ポインター_基本コントロール コード

2020/04/24 • [Edit Online](#)

FSCTL_GET_取得_ポインター_BASEは、ボリュームの先頭に対して相対的なファイルシステムの最初の論理クラスター番号 (LCN) にセクターオフセットを返します。

この操作を実行するには、次のパラメーターを使用して`Fltfscontrolfile`関数または`zwfscontrolfile`関数を呼び出します。

パラメーター

]の FileObject [

`Fltfscontrolfile`のみ。ベースを取得するために、FSCTL_GET_retrieve__ポインターによって取得されるボリュームのファイルオブジェクトポインター。このパラメーターは必須であり、NULLにすることはできません。

FileHandle

`Zwfscontrolfile`のみ。ベースを取得するために、FSCTL_GET_retrieve__ポインターによって取得されるボリュームのファイルハンドル。このパラメーターは必須であり、NULLにすることはできません。

_ボリューム_名前_のアクセス権を管理_には、このハンドルを SE で開く必要があります。詳細については、「[ファイルセキュリティとアクセス権](#)」を参照してください。

]の FsControlCode [

操作の制御コード。この操作では、FSCTL_使用して_取得_ポインター_ベースを取得します。

InputBuffer

この操作では使用されません。をNULLに設定します。

]内の InputBufferLength [

この操作では使用されません。を0に設定します。

OutputBuffer [out]

ボリュームの先頭を基準とした、ファイルシステムの最初の LCN へのセクターオフセットを受け取る、大きな_整数へのポインター。

]の OutputBufferLength [

出力バッファのサイズ (バイト単位)。この値は8である必要があります。

状態ブロック

`Fltfscontrolfile`または`zwfscontrolfile`は、次のいずれかのような状態_SUCCESS または適切な NTSTATUS 値を返します。

「	「
STATUS_SUCCESS	操作は成功しました。OutputBuffer には、ボリュームの開始位置を基準とした、ファイルシステムの最初の LCN に対するセクターオフセットが含まれます。
STATUS_ACCESS_DENIED	ユーザーには、SE_MANAGE_VOLUME アクセス権がありません。
STATUS_BUFFER_TOO_SMALL	OutputBuffer は、結果に対して十分な大きさではありません。バッファに情報が書き込まれていません。

「	「
STATUS_INVALID_PARAMETER	パラメーターが有効ではありませんでした。たとえば、使用されるハンドルが有効なボリュームハンドルではありません。

解説

FSCTL によって取得された値を追加すると、FSCTL_GET__によって取得された値に基づいて取得された_取得__ポインターによって取得されます。_は、ボリューム相対ファイルエクステントオフセットが生成されます。

FastFAT と exFAT デバイスでは、FSCTL_GET_取得_ポインター_基本制御コードを使用できます。この機能は、フラッシュドライブなどのデバイスでの BitLocker の使用をサポートしています。

前提条件

バージョン	Windows Server 2008 R2、Windows 7
ヘッダー	Ntifs

関連項目

[FltFsControlFile](#)

[ZwFsControlFile](#)

FSCTL_取得_取得_ポインター制御コード

2020/04/24 • [Edit Online](#)

FSCTL_GET_取得_ポインター制御コードは、特定のファイルのディスク上の割り当てと場所を記述する可変サイズのデータ構造を取得します。構造体は、仮想クラスター番号 (VCN、ファイル/ストリーム領域内のオフセット)、および論理クラスター番号 (LCN、ボリューム領域内のオフセット) との間のマッピングを記述します。

この操作を実行するには、次のパラメーターを使用して **Fltfscontrolfile** または **zwfscontrolfile** を呼び出します。

再解析ポイントおよび FSCTL_GET_取得_ポインターコントロールコードの詳細については、Microsoft Windows SDK のドキュメントを参照してください。

Parameters

ファ

Fltfscontrolfile のみ。FSCTL_GET_取得_ポインターがマッピングを取得する代替ストリーム、ファイル、またはディレクトリのファイルオブジェクトポインター。このパラメーターは必須であり、NULL にすることはできません。

FileHandle

Zwfscontrolfile のみ。FSCTL_GET_取得_ポインターがマッピングを取得する代替ストリーム、ファイル、またはディレクトリのファイルハンドル。FileHandle の値がボリューム全体のハンドルである場合、ルーチンは、無効なクラスターファイルの vcns とエクステントのマップを返します。このパラメーターは必須であり、NULL にすることはできません。

FsControlCode

操作の制御コード。この操作を行うには、FSCTL_を使用して_取得_ポインターを取得してください。

InputBuffer

代替ストリーム、ファイル、またはディレクトリの先頭をマークする仮想クラスター番号 (VCN) を示す、開始_VCN_入力_バッファー構造体へのポインター。開始_の VCN_入力_バッファー構造体は、次のように定義されています。

```
typedef struct {  
    LARGE_INTEGER ;  
} STARTING_VCN_INPUT_BUFFER, *PSTARTING_VCN_INPUT_BUFFER;
```

属する

StartingVcn

FSCTL_GET_取得_ポインターが、エクステントと、関連付けられた仮想および論理クラスター番号の列挙を開始するときに使用する VCN。_FSCTL のファイルシステム制御コードを使用して **Fltfscontrolfile** または **zwfscontrolfile** を初めて呼び出すときに、_ポインター_取得するには、StartingVcn を 0 に設定する必要があります。

Outputbuffer がファイルの vcns とエクステントのマップ全体を格納するのに十分な大きさではない場合、呼び出し元は、取得__ポインターの nextvcn メンバーで返される値を使用して、より多くのマップデータを要求する必要があります。VCN を開始しています。

InputBufferLength

InputBuffer の入力バッファーの長さ (バイト単位)。

OutputBuffer

代替ストリーム、ファイル、またはディレクトリに対応するディスク上のエクステントの列挙体を含む、型取得_ポインター_ポインターの可変サイズ構造体へのポインター。

```
typedef struct RETRIEVAL_POINTERS_BUFFER {
    ULONG    ExtentCount;
    LARGE_INTEGER    StartingVcn;
    struct {
        LARGE_INTEGER    NextVcn;
        LARGE_INTEGER    Lcn;
    } Extents[1];
} RETRIEVAL_POINTERS_BUFFER, *PRETRIEVAL_POINTERS_BUFFER;
```

属する

ExtentCount

エクステント配列内の要素の数。

StartingVcn

関数呼び出しによって返された VCN を開始しています。これは必ずしも関数呼び出しによって要求される VCN とは限りません。ファイルシステムドライバーは、要求された開始 VCN が検出されたエクステントの最初の VCN に切り捨てられる可能性があるためです。

エクステント

エクステント構造体の配列。配列内のメンバーの数については、「ExtentCount」を参照してください。配列の各メンバーには、次のメンバーがあります。

NextVcn

次のエクステントが開始される VCN。この値は、StartingVcn (最初のエクステント配列メンバーの場合) または配列の前のメンバーのnextvcn (他のすべてのエクステント配列メンバーの場合) のいずれかを差し引いた値で、現在のエクステントのクラスター単位の長さになります。

Lcn

ボリュームで現在のエクステントが開始される LCN。NTFS では、値 (LONGLONG)-1 は、部分的に割り当てられた圧縮ユニット、またはスパースファイルの未割り当て領域のいずれかを示します。

OutputBufferLength

Outputbuffer)パラメーターが指すバッファのサイズ (バイト単位)。

ステータス ブロック

Fltfscontrolfileとzwfscontrolfileはどちらも STATUS_SUCCESS または適切な NTSTATUS エラー値を返します。

VCN/extents マップがOutputbuffer)に収まらない場合、どちらのルーチンも STATUS_BUFFER_OVERFLOW という値を返し、呼び出し元は取得のnextvcnメンバーで返された値を使用して、より多くのマップデータを要求する必要があり_ポインターは、Fltfscontrolfileまたはzwfscontrolfileへの次の呼び出しで、バッファ構造を開始 VCN (StartingVcn) として_します。

StartingVcnで指定された値がファイルの末尾を越えている場合は、_ファイルの_end_が返されます。

注釈

FastFAT と exFAT デバイスでは、FSCTL_GET_取得_ポインター制御コードを使用できます。この機能は、フラッシュドライブなどのデバイスでの BitLocker の使用をサポートしています。

要件

Header	Ntifs (Ntifs を含む)

関連項目

FltFsControlFile

ZwFsControlFile

FSCTL_WOF_バージョン管理コードを取得する

2020/04/24 • [Edit Online](#)

FSCTL_GET_WOF_バージョンの i/o 制御コード (IOCTL) を使用して、特定のプロバイダーをサポートするために使用されるドライバーのバージョンを照会します。

この操作を実行するには、次のパラメーターを使用して [Fltfscontrolfile](#) または [zwfscontrolfile](#) を呼び出します。

```
BOOL
WINAPI
DeviceIoControl( (HANDLE)      hDevice,          // handle to device
                 (DWORD)       FSCTL_GET_WOF_VERSION, // dwIoControlCode
                 (LPDWORD)     lpInBuffer,       // input buffer
                 (DWORD)       nInBufferSize,    // size of input buffer
                 (LPDWORD)     lpOutBuffer,      // output buffer
                 (DWORD)       nOutBufferSize,   // size of output buffer
                 (LPDWORD)     lpBytesReturned,  // number of bytes returned
                 (LPOVERLAPPED) lpOverlapped );  // OVERLAPPED structure
```

パラメーター

hDevice []

デバイスへのハンドル。デバイスハンドルを取得するには、[CreateFile](#) 関数または同様の API を呼び出します。

]の dwIoControlCode [

操作の制御コード。この操作には、FSCTL_ 使用して _WOF_ バージョンを取得してください。

lpInBuffer

操作の入力バッファー。これは、[WOF_外部_情報](#) 構造体へのポインターです。

]の nInBufferSize [

入力バッファーのサイズ (バイト単位)。これは `sizeof(WOF_外部_情報)` にする必要があります。

lpOutBuffer [out]

操作の出力バッファー。これは、[WOF_バージョン_情報](#) 構造体へのポインターです。

]の nOutBufferSize [

出力バッファーのサイズ (バイト単位)。これは `sizeof(WOF_VERSION_INFO)` である必要があります。

lpBytesReturned [out]を返しました

LPDWORD

出力バッファーに格納されているデータのサイズ (バイト単位) を受け取る変数へのポインター。

出力バッファーが小さすぎると、呼び出しが失敗し、[GetLastError](#) がエラー __ バッファーが不足し、*lpbytesreturned* 0 を返します。

lpOverlapped が null の場合、*lpbytesreturned* null にすることはできません。操作によって出力データが返されず、*lpoutbuffer* が NULL の場合でも、[DeviceIoControl](#) は *lpoutbuffer* 使用して返されます。このような操作を行った後、返された *lpbytesreturned* 値は無意味になります。

lpOverlapped が null でない場合は、*lpbytesreturned* null になることがあります。このパラメーターが NULL でなく、操作がデータを返す場合は、重複操作が完了するまで、*lpbytesreturned* 返されることは意味がありません。返されるバイト数を取得するには、[GetOverlappedResult](#) を呼び出します。*hdevice* パラメーターが i/o 完了ポートに関連付けられている場合は、[GetQueuedCompletionStatus](#) を呼び出すことによって返されるバイト数を取得できます。

]の *lpOverlapped* [
LPOVERLAPPED

オーバーラップされた構造体へのポインター。

ファイル__フラグを指定せずに *hdevice*を開いた場合、*lpOverlapped*は無視されます。

*Hdevice*がファイル__フラグを使用して開かれている場合は、重複フラグ_、この操作は、オーバーラップ (非同期) 操作として実行されます。この場合、*lpOverlapped*は、イベントオブジェクトへのハンドルを含む有効な**オーバーラップ**構造体を指す必要があります。それ以外の場合、関数は予期しない方法で失敗します。

オーバーラップ操作の場合、**DeviceIoControl**はすぐに制御を返し、操作が完了するとイベントオブジェクトを通知します。それ以外の場合、この関数は、操作が完了するかエラーが発生するまで、を返しません。

状態ブロック

操作が正常に完了した場合、**DeviceIoControl**は0以外の値を返します。

操作が失敗した場合、または保留中の場合、**DeviceIoControl**は0を返します。エラーの詳細情報を取得するには、**GetLastError**を呼び出します。

前提条件

バージョン	Windows 10 以降で使用できます。
ヘッダー	Ntifs (Ntifs または Fltkernel .h を含む)

FSCTL_ボリューム制御コードの無効化

2020/04/24 • [Edit Online](#)

FSCTL_は、指定されたファイルオブジェクトまたはハンドルによって表されるデバイス上にマウントされているすべてのボリュームを検出して削除_ボリューム制御コードが検出し、削除します。

この操作を実行するには、ミニフィルタードライバーが`Fltfscontrolfile`を呼び出します。また、ファイルシステム、リダイレクター、およびレガシファイルシステムフィルタードライバーは、次のパラメーターを使用して`Zwfscontrolfile`を呼び出します。

パラメーター

ファ

デバイスへのハンドル。デバイスハンドルを取得するには、`CreateFile`関数を呼び出します。

FileHandle

デバイスへのハンドル。デバイスハンドルを取得するには、`CreateFile`関数を呼び出します。

FsControlCode

操作の制御コード。この操作では、FSCTL_を使用して_ボリュームを無効にします。

InputBuffer

この操作では使用されません。をNULLに設定します。

InputBufferLength

この操作では使用されません。を0に設定します。

OutputBuffer

この操作では使用されません。をNULLに設定します。

OutputBufferLength

この操作では使用されません。を0に設定します。

状態ブロック

操作が成功した場合、または適切な NTSTATUS 値が返された場合、`Fltfscontrolfile`と`ZWFSCONTROLFILE` STATUS_SUCCESS を返します。

解説

FSCTL_は、ボリュームデバイスオブジェクトではなく、ファイルシステムのコントロール (名前付き) デバイスオブジェクトに_ボリュームが送信されます。コントロールデバイスオブジェクトの詳細については、「[コントロールデバイスオブジェクトの作成](#)」を参照してください。

FAT および NTFS ファイルシステムは、IRP_MJ_PNP/_IRP に応答することによって、突然の削除を処理します。これにより、突然_削除が_されます。

前提条件

ヘッダー	Ntifs (Ntifs を含む)

関連項目

[FltFsControlFile](#)

FSCTL_は、有効なコントロールコード_パス名です

2020/04/24 • [Edit Online](#)

FSCTL_は_pathname_有効なコントロールコードが指定されたパス名の静的分析を実行し、パス名が正しい形式であるかどうかを示す状態値を返します (たとえば、無効な文字、許容されるパスは使用できません)。長さなど)。この分析ではボリュームのコンテンツは考慮されないため、"偽陽性" が示されることがあります。つまり、パス名が適切な形式であることが分析によって示されている可能性があります。負の結果はより信頼性が高くなりますが、正確であるとは限りません。

この制御コードは、高速な FAT ファイルシステムではサポートされていません。NTFS や UDF では意味のある操作ではありません。NTFS および UDF では、このような広範なコードによって、任意の文字列が有効なパス名であることがサポートされています。

この操作を実行するには、次のパラメーターを使用して`Fltfscontrolfile`または`zwfscontrolfile`を呼び出します。

パラメーター

ファ

`Fltfscontrolfile`のみ。使用されません。

FileHandle

`Fltfscontrolfile`のみ。使用されません。

FsControlCode

操作の制御コード。FSCTL を使用する_は、この操作に対して有効なパス名__ます。

InputBuffer

確認する NULL で終わるパス名文字列を含むバッファー。

InputBufferLength

パス名の長さ (バイト単位)。

OutputBuffer

使用されません。

OutputBufferLength

使用されません。

状態ブロック

`Fltfscontrolfile`または`zwfscontrolfile`は、パス名が正しい形式である場合、STATUS_SUCCESS を返します。それ以外の場合、使用されるルーチンは適切な NTSTATUS エラーコードを返します。

前提条件

ヘッダー	Ntifs (Ntifs または Fltkernel.h を含む)

FSCTL_は_ボリューム_ダーティコントロールコード

2020/04/24 • [Edit Online](#)

FSCTL_は_ボリューム_ダーティコントロールコードによって、指定されたボリュームがダーティかどうか判断されます。

ボリューム情報ファイルが破損している場合は、NTFS によってステータス_ファイル_破損_エラーが返されます。

この操作を実行するには、ミニフィルタードライバーは、次のパラメーターを使用して`Fltfscontrolfile`を呼び出します。ファイルシステム、リダイレクター、およびレガシファイルシステムフィルタードライバーは、次のパラメーターを使用して`zwfscontrolfile`を呼び出します。

Parameters

ファ

`Fltfscontrolfile`のみ。ボリュームのファイルオブジェクトポインター。このパラメーターは、マウントされたファイルシステムボリュームを開いているユーザーボリュームを表す必要があります。このパラメーターは必須であり、NULLにすることはできません。

FileHandle

`Zwfscontrolfile`のみ。ボリュームのハンドル。このパラメーターは、マウントされたファイルシステムボリュームを開いているユーザーボリュームのハンドルである必要があります。このパラメーターは必須であり、NULLにすることはできません。

FsControlCode

操作の制御コード。この操作では、ボリューム_ダーティ__を使用します。

InputBuffer

この操作では使用されません。をNULLに設定します。

InputBufferLength

この操作では使用されません。を0に設定します。

OutputBuffer

ボリュームが現在ダーティであるかどうかを示すフラグの ULONG ビットマスクを受け取る、呼び出し元が割り当てた32ビットのアラインされたバッファーへのポインター。次の表に示す1つ以上のフラグを設定できます。

VALUE	II
VOLUME_IS_DIRTY	ボリュームがダーティです。
VOLUME_UPGRADE_SCHEDULED	この値は現在使用されていません。
その他のすべての値	将来使用するために予約されています。

OutputBufferLength

`Outputbuffer`パラメーターによってポイントされるバッファーのサイズ (バイト単位)。このサイズは、少なくとも `sizeof (ULONG)` である必要があります。

ステータス ブロック

`Fltfscontrolfile`または`zwfscontrolfile`は、操作が成功した場合に `STATUS_SUCCESS` を返します。それ以外の場合、適

切な関数は次のいずれかの NTSTATUS 値を返す可能性があります。

⌘	⌘
STATUS_INSUFFICIENT_RESOURCES	ファイルシステムでブール割り当てエラーが発生しました。これはエラーコードです。
STATUS_INVALID_PARAMETER	<i>Outputbuffer</i> パラメーターが指すバッファーが NULL であるか、 <i>FileHandle</i> または <i>FileObject</i> パラメーターが、開いているユーザーボリュームを表していません。これはエラーコードです。
STATUS_INVALID_USER_BUFFER	<i>Outputbuffer</i> パラメーターが指すバッファーが、再解析ポイントデータを保持するのに十分な大きさではありません。または、 <i>FileHandle</i> または <i>FileObject</i> パラメーターが、開いているユーザーボリュームを表していません。これはエラーコードです。
STATUS_VOLUME_DISMOUNTED	ボリュームがマウントされていません。これはエラーコードです。

要件

Header	Ntifs (Ntifs または Fltkernel.h を含む)

関連項目

- [FLT_コールバック_データ](#)
- [IRP_MJ_ファイルの FLT_パラメーター_システム_コントロール](#)
- [FtFsControlFile](#)
- [IRP_MJ_ファイル_システム_コントロール](#)
- [ZwFsControlFile](#)

FSCTL_LMR_リンクを取得する_情報制御コードを追跡する

2020/04/24 • [Edit Online](#)

FSCTL_LMR_GET_link_は、_情報制御コードは、ファイルのリンク追跡情報を取得します。

この操作を実行するには、次のパラメーターを使用して`Fltfscontrolfile`または`zwfscontrolfile`を呼び出します。

Parameters

フア

`Fltfscontrolfile`のみ。リモートボリュームのファイルオブジェクトポインター。このパラメーターは必須であり、NULLにすることはできません。

FileHandle

`Zwfscontrolfile`のみ。リモートボリュームのハンドル。このパラメーターは必須であり、NULLにすることはできません。

FsControlCode

操作の制御コード。この操作の_情報を追跡_には、FSCTL_LMR_ 使用して_リンクを取得します。

InputBuffer

なし。

InputBufferLength

使用しません。

OutputBuffer

ファイルのリンク追跡情報を格納している_情報の構造を追跡_リンク。

```
typedef struct _LINK_TRACKING_INFORMATION {
    LINK_TRACKING_INFORMATION_TYPE Type;
    UCHAR VolumeId[16];
} LINK_TRACKING_INFORMATION, *PLINK_TRACKING_INFORMATION;
```

各種

ファイルが格納されているファイルシステムの種類を指定する_情報_型情報の列挙値を追跡_リンク。このメンバーがDfsLinktrackinginformationの値を保持している場合、ファイルは分散ファイルシステム上に存在します。このメンバーがNtfsLinkTrackingInformationの値を保持している場合、ファイルは NTFS ファイルシステム上に存在します。

Volumeld

ボリューム識別子を保持する符号なし文字配列。

OutputBufferLength

*Outputbufferl*パラメーターが指すバッファのサイズ (バイト単位)。

ステータス ブロック

`Fltfscontrolfile`または`zwfscontrolfile`は、操作が成功した場合に STATUS_SUCCESS を返します。それ以外の場合、適切な関数は、適切な NTSTATUS エラーコードを返します。

要件

Header	Ntifs (Ntifs を含む)

FSCTL__システム_HIVE 制御コードとしてマークする

2020/04/24 • [Edit Online](#)

FSCTL_マーク__システム_HIVE コントロールコードとしてマークすると、指定したファイルにレジストリのシステムハイクが含まれていることがファイルシステムに通知されます。ファイルシステムは、デッドロックを回避し、データの整合性を確保するために、すぐにシステム hive データをディスクにフラッシュする必要があります。このファイルシステム制御コードは、レジストリのシステムハイクを含むファイル以外のファイルでは使用しないでください。この制御コードは、ディレクトリまたはボリュームハンドルでは機能しません。リモートコンピューター上のファイルにアクセスするファイルシステムのリダイレクターは、この制御コードを操作なしとして扱います。

カーネルレベルのコンポーネントのみが、このファイルシステム制御コードを使用できます。

この操作を実行するには、次のパラメーターを使用して **Fltfscontrolfile** または **zwfscontrolfile** を呼び出します。

Parameters

ファ

Fltfscontrolfile のみ。ユーザーファイルのファイルオブジェクトポインター。このパラメーターは必須であり、NULL にすることはできません。

FileHandle

Zwfscontrolfile のみ。ユーザーファイルのハンドル。このパラメーターは必須であり、NULL にすることはできません。

FsControlCode

操作の制御コード。この操作には、FSCTL_ 使用して _ を _ システム_HIVE としてマークしてください。

InputBuffer

使用しません。このパラメーターに NULL 値を割り当てます。

InputBufferLength

使用しません。

OutputBuffer

使用しません。このパラメーターに NULL 値を割り当てます。

OutputBufferLength

使用しません。

ステータス ブロック

Fltfscontrolfile または **zwfscontrolfile** は、操作が成功した場合に STATUS_SUCCESS を返します。それ以外の場合、適切な関数は、適切な NTSTATUS エラーコードを返します。

要件

Header	Ntifs (Ntifs を含む)

FSCTL_MARK_ボリューム_ダーティコントロール コード

2020/04/24 • [Edit Online](#)

FSCTL_MARK_ボリューム_ダーティコントロールコードは、指定されたボリュームをダーティとしてマークします。これにより、次のシステムの再起動時に autochk.exe がボリューム上で実行されます。

この操作を実行するには、次のパラメーターを使用して **Fltfscontrolfile** または **zwfscontrolfile** を呼び出します。

Parameters

Instance

Fltfscontrolfile のみ。FSCTL 要求を開始するミニフィルタードライバインスタンスへの非透過インスタンスポインター。

ファ

Fltfscontrolfile のみ。ダーティとマークするボリュームを指定するファイルポインターオブジェクト。このパラメーターは必須であり、NULL にすることはできません。

FileHandle

Zwfscontrolfile のみ。ダーティとマークされるボリュームへのハンドル。このパラメーターは必須であり、NULL にすることはできません。

FsControlCode

操作の制御コード。この操作では、FSCTL_を使用して_ボリューム_ダーティとマークします。

InputBuffer

この操作では使用されません。をNULLに設定します。

InputBufferLength

この操作では使用されません。を0に設定します。

OutputBuffer

この操作では使用されません。をNULLに設定します。

OutputBufferLength

この操作では使用されません。を0に設定します。

ステータス ブロック

Fltfscontrolfile または **zwfscontrolfile** ルーチンは、STATUS_SUCCESS または適切な NTSTATUS 値を返します。

「	「
STATUS_INVALID_PARAMETER	<i>FileObject</i> または <i>FileHandle</i> が有効なボリュームハンドルを表していないか、別のパラメーターが無効です。
STATUS_ACCESS_DENIED	呼び出し元には、SE_MANAGE_VOLUME アクセス権がありません。
STATUS_VOLUME_DISMOUNTED	ファイルシステムボリュームがマウント解除されています。

STATUS_TOO_LATE	ファイルシステムボリュームがシャットダウンされています。
STATUS_MEDIA_WRITE_PROTECTED	ファイルシステムボリュームは読み取り専用です。

注釈

**ReFS: **このコードはサポートされていません。

要件

Header	Ntifs (FltKernel .h または Ntifs を含む)

関連項目

[FltFsControlFile](#)

[FSCTL_が_ボリューム_ダーティ](#)

[ZwFsControlFile](#)

FSCTL_クエリ_ボリューム_NUMA_情報制御コード

2020/04/24 • [Edit Online](#)

FSCTL_QUERY_volume_numa_INFO control code は、ボリュームの現在の Non-uniform Memory ARCHITECTURE (numa) ノードインデックスを検索します。

この操作を実行するには、次のパラメーターを使用して [DeviceIoControl](#) 関数を呼び出します。

```
BOOL
WINAPI
DeviceIoControl( (HANDLE)      hDevice,
                 (DWORD)       FSCTL_QUERY_VOLUME_NUMA_INFO, // dwIoControlCode
                 (LPDWORD)     lpInBuffer,    // input buffer
                 (DWORD)       nInBufferSize, // size of input buffer
                 (LPDWORD)     lpOutBuffer,    // output buffer
                 (DWORD)       nOutBufferSize, // size of output buffer
                 (LPDWORD)     lpBytesReturned, // number of bytes returned
                 (LPOVERLAPPED) lpOverlapped ); // OVERLAPPED structure
```

パラメーター

Hdevice []

デバイスへのハンドル。デバイスハンドルを取得するには、[CreateFile](#) 関数を呼び出します。

これは、デバイス内のファイルまたはディレクトリへのハンドルにすることもできます。この FSCTL は、ファイルまたはディレクトリを含むボリュームの NUMA ノードを返します。

] の *Dwiocontrolcode* []

操作の制御コード。この操作には、FSCTL__ボリューム_NUMA_INFO を使用してください。

Lpinbuffer

この操作では使用されません。NULL に設定

] の *nInBufferSize* []

この操作では使用されません。0 に設定します。

Lpoutbuffer [out]

Non-uniform Memory Architecture (NUMA) ボリュームの現在のノードを指定する [numa_INFO_出力構造](#) [_FSCTL_QUERY_volume](#) を受け取るバッファーへのポインター。

] の *Noutbuffersize* []

出力バッファーのサイズ (バイト単位)。

Lpbytesreturned [out] を返しました

出力バッファーに格納されているデータのサイズ (バイト単位) を受け取る変数へのポインター。

出力バッファーが小さすぎると、呼び出しが失敗し、[GetLastError](#) がエラー __ バッファーが不足し、*lpbytesreturned* 0 を返します。

LpOverlapped が null の場合、*lpbytesreturned* null にすることはできません。操作によって出力データが返されず、*Lpoutbuffer* が NULL の場合でも、[DeviceIoControl](#) は *Lpoutbuffer* 使用して返されます。このような操作を行った後、返された *Lpbytesreturned* 値は無意味になります。

LpOverlapped が null でない場合は、*lpbytesreturned* null になることがあります。このパラメーターが NULL でなく、操作がデータを返す場合は、重複操作が完了するまで、*lpbytesreturned* 返されることは意味がありません。返されるバイト数を取得する

には、[GetOverlappedResult](#)を呼び出します。*Hdevice*パラメーターが i/o 完了ポートに関連付けられている場合は、[GetQueuedCompletionStatus](#)を呼び出すことによって返されるバイト数を取得できます。

] の*lpOverlapped* [
[オーバーラップ](#)された構造体へのポインター。

*ファイル__フラグ*を指定せずに*hdevice*を開いた場合、*lpOverlapped*は無視されます。

*Hdevice*が*ファイル_フラグ*を使用して開かれている場合は、重複フラグ_、この操作は、オーバーラップ (非同期) 操作として実行されます。この場合、*lpOverlapped*は、イベントオブジェクトへのハンドルを含む有効な[オーバーラップ](#)構造体を指す必要があります。それ以外の場合、関数は予期しない方法で失敗します。

オーバーラップ操作の場合、[DeviceIoControl](#)はすぐに制御を返し、操作が完了するとイベントオブジェクトを通知します。それ以外の場合、この関数は、操作が完了するかエラーが発生するまで、を返しません。

戻り値

操作が正常に完了した場合、[DeviceIoControl](#)は0以外の値を返します。

操作が失敗した場合、または保留中の場合、[DeviceIoControl](#)は0を返します。エラーの詳細情報を取得するには、[GetLastError](#)を呼び出します。

要件

Header	Winioctl. h;Ntifs

関連項目

[DeviceIoControl](#)

[FSCTL_クエリ_ボリューム_NUMA_情報_出力](#)

FSCTL_オフロード_読み取り制御コード

2020/04/24 • [Edit Online](#)

FSCTL_オフロード_読み取り制御コードは、オフロード読み取りプリミティブをサポートするストレージシステム内のデータブロックに対してオフロード読み取りを開始します。

この操作を実行するには、次のパラメーターを使用して`Fltfscontrolfile`または`zwfscontrolfile`を呼び出します。

Parameters

]のインスタンス [

`Fltfscontrolfile`のみ。呼び出し元の非透過インスタンスポインター。このパラメーターは必須であり、NULL にすることはできません。

]の FileObject [

`Fltfscontrolfile`のみ。読み取り元のファイルを指定するファイルポインターオブジェクト。このパラメーターは必須であり、NULL にすることはできません。

]の FileHandle [

`Zwfscontrolfile`のみ。読み取り元のファイルのファイルハンドル。このパラメーターは必須であり、NULL にすることはできません。

]の FsControlCode [

操作の制御コード。この操作には、FSCTL_OFFLOAD_READを使用してください。

InputBuffer

読み取るデータブロックのサイズとオフセットを格納している、FSCTL_オフロード_読み取り_入力 構造体へのポインター。

]内の InputBufferLength [

*InputBuffer*が指すバッファのサイズ (バイト単位)。この値はsizeof(FSCTL_OFFLOAD_READ_INPUT) です。

OutputBuffer [out]

オフロード読み取り操作の結果を受け取る、FSCTL_オフロード_読み取り_出力 構造体へのポインター。

OutputBufferLength [out]

*Outputbuffer*パラメーターが指すバッファのサイズ (バイト単位)。この値は、少なくともsizeof(FSCTL_OFFLOAD_読み取り_出力) である必要があります。

ステータス ブロック

`Fltfscontrolfile`または`zwfscontrolfile`は、操作が成功した場合に STATUS_SUCCESS を返します。それ以外の場合、適切な関数は、次のいずれかの NTSTATUS 値を返す可能性があります。

「	「
STATUS_INVALID_DEVICE_REQUEST	指定されたハンドルは、有効なファイルハンドルではありません。

II	II
STATUS_INVALID_PARAMETER	ファイルサイズが PAGE_SIZE 未満です。 \- または - <i>Inputbufferlength</i> < sizeof(FSCTL_OFFLOAD_READ_INPUT)。 \- または - FSCTL_OFFLOAD_READ_INPUTの1つ以上のメンバーが正しくありません。 Fileoffsetは、ボリュームの論理セクターサイズの倍数ではありません。Copylengthは、ボリュームの論理セクターサイズの倍数ではありません。Sizeは、FSCTL_OFFLOAD_READ_INPUT構造体のサイズではありません。Fileoffset + Copylength > MAXULONGLONG。
STATUS_VOLUME_DISMOUNTED	ファイルシステムボリュームがマウント解除されています。
STATUS_NOT_SUPPORTED	オフロード読み取り操作は、このボリュームではサポートされていません。
STATUS_OFFLOAD_READ_FILE_NOT_SUPPORTED	要求されたファイルの種類はサポートされていません。次のファイルの種類では、オフロード操作はサポートされていません。 • トランザクションファイル (TxF) • 非ユーザーファイル • 圧縮されたファイル • 暗号化されたファイル • スパース ファイル • NTFS Metatdata ファイル
STATUS_FILE_DELETED	このファイルのデータストリームは無効です。
STATUS_FILE_CLOSED	ファイルハンドルは閉じられています。
STATUS_INVALID_HANDLE	指定されたファイルハンドルは無効です。
STATUS_FILE_LOCK_CONFLICT	現在のファイルロック状態のため、読み取りアクセスが不十分です。
STATUS_END_OF_FILE	FSCTL_OFFLOAD_READ_INPUT のfileoffsetメンバーは、ファイルの終わり (EOF) の後に始まります。
STATUS_DISMOUNTED_VOLUME	マウント解除されたボリュームでオフロード読み取りを行うことはできません。
STATUS_INSUFFICIENT_RESOURCES	要求を完了するために使用できるリソースが不足しています。

㊦	㊦
STATUS_BUFFER_TOO_SMALL	FSCTL_OFFLOAD_READ_OUTPUT 構造体を受け取る <i>Outputbuffer</i> の <i>outputbufferlength</i> が小さすぎます。

注釈

オフロード読み取りは、通常のファイルに対してのみ使用できます。サポートされていないファイルの種類の一覧については、[ステータス_オフロード_読み取り__ファイル]の説明を参照してください。

読み取りが有効なデータ長 (VDL) を超えて開始される可能性があります、EOF を超えることはできません。

要件

バージョン	Windows 8 以降で使用できます。
Header	Ntifs (Ntifs または Fltkernel.h を含む)

関連項目

[FltFsControlFile](#)

[ZwFsControlFile](#)

[FSCTL_オフロード_読み取り_入力](#)

[FSCTL_オフロード_読み取り_出力](#)

FSCTL_オフロード_書き込み制御コード

2020/04/24 • [Edit Online](#)

FSCTL_offload_書き込み制御コードは、オフロード書き込みプリミティブをサポートするストレージシステム内のデータブロックに対してオフロード書き込みを開始します。

この操作を実行するには、ミニフィルタードライバーは、次のパラメーターを使用して **Fltfscontrolfile** を呼び出します。ファイルシステム、リダイレクター、およびレガシファイルシステムフィルタードライバーは、次のパラメーターを使用して **zwfscontrolfile** を呼び出します。

Parameters

]のインスタンス [

Fltfscontrolfile のみ。呼び出し元の非透過インスタンスポインター。このパラメーターは必須であり、NULL にすることはできません。

]の FileObject [

Fltfscontrolfile のみ。書き込み先のファイルを指定するファイルポインターオブジェクト。このパラメーターは必須であり、NULL にすることはできません。

]の FileHandle [

Zwfscontrolfile のみ。書き込み先のファイルのファイルハンドル。このパラメーターは必須であり、NULL にすることはできません。

]の FsControlCode [

操作の制御コード。この操作には、FSCTL_オフロード_書き込みを使用します。

InputBuffer

読み取り対象のデータブロックのサイズとオフセットを格納している、**FSCTL_オフロード_書き込み_入力** 構造体へのポインター。

]内の InputBufferLength [

InputBuffer が指すバッファのサイズ (バイト単位)。この値は `sizeof(FSCTL_OFFLOAD_WRITE_INPUT)` です。

OutputBuffer [out]

読み取り対象のデータブロックのサイズとオフセットを格納している、**FSCTL_オフロード_書き込み_入力** 構造体へのポインター。

OutputBufferLength [out]

Outputbuffer パラメーターが指すバッファのサイズ (バイト単位)。この値は、少なくとも `sizeof(FSCTL_OFFLOAD_READ_OUTPUT)` である必要があります。

ステータス ブロック

Fltfscontrolfile または **zwfscontrolfile** は、操作が成功した場合に STATUS_SUCCESS を返します。それ以外の場合、適切な関数は、次のいずれかの NTSTATUS 値を返す可能性があります。

「	「
STATUS_INVALID_DEVICE_REQUEST	指定されたハンドルは、有効なファイルハンドルではありません。

II	II
STATUS_INVALID_PARAMETER	ファイルサイズが PAGE_SIZE 未満です。 \- または - <i>Inputbufferlength</i> < sizeof(FSCTL_OFFLOAD_WRITE_INPUT)。 \- または - FSCTL_OFFLOAD_WRITE_INPUTの1つ以上のメンバーが正しくありません。 Fileoffsetは、ボリュームの論理セクターサイズの倍数ではありません。Copylengthは、ボリュームの論理セクターサイズの倍数ではありません。Transferoffsetは、ボリュームの論理セクターサイズの倍数ではありません。Sizeは、FSCTL_OFFLOAD_WRITE_INPUT構造体のサイズではありません。Fileoffset > ファイルの有効なデータ長 (VDL) です。Fileoffset + Copylength > MAXULONGLONG。
STATUS_NOT_SUPPORTED	オフロード読み取り操作は、このボリュームではサポートされていません。
STATUS_OFFLOAD_WRITE_FILE_NOT_SUPPORTED	要求されたファイルの種類はサポートされていません。次のファイルの種類では、オフロード操作はサポートされていません。 • トランザクションファイル (TxF) • 非ユーザーファイル • 圧縮されたファイル • スパース ファイル • 暗号化されたファイル • NTFS Metatdata ファイル
STATUS_TOO_LATE	マウント解除後にボリュームに書き込み操作を実行しようとした。
STATUS_FILE_DELETED	このファイルのデータストリームは無効です。
STATUS_FILE_CLOSED	ファイルハンドルは閉じられています。
STATUS_INVALID_HANDLE	指定されたファイルハンドルは無効です。
STATUS_FILE_LOCK_CONFLICT	現在のファイルロック状態のため、読み取りまたは書き込みアクセス権を付与できません。
STATUS_END_OF_FILE	FSCTL_OFFLOAD_WRITE_INPUTのfileoffsetメンバーは、ファイルの終わり (EOF) の後に始まります。
STATUS_DISMOUNTED_VOLUME	マウント解除されたボリュームでオフロード書き込みを行うことはできません。

STATUS_MEDIA_WRITE_PROTECTED	ボリュームは読み取り専用です。
STATUS_INSUFFICIENT_RESOURCES	要求を完了するために使用できるリソースが不足しています。
STATUS_BUFFER_TOO_SMALL	<i>InputBuffer</i> がFSCTL_OFFLOAD_WRITE_INPUT構造体を含むようにするには、 <i>inputbufferlength</i> が小さすぎます。 \\- または - FSCTL_OFFLOAD_WRITE_OUTPUT構造体を受け取る <i>Outputbuffer</i> の <i>outputbufferlength</i> が小さすぎます。

注釈

オフロード読み取りは、通常のファイルに対してのみ使用できます。サポートされていないファイルの種類の一覧については、[ステータス_オフロード_書き込み__ファイルを書き込む]の説明を参照してください。

要件

バージョン	Windows 8以降で使用できます。
Header	Ntifs (Ntifs または Fltkernel.h を含む)

関連項目

[FltFsControlFile](#)

[ZwFsControlFile](#)

[FSCTL_オフロード_書き込み_入力](#)

[FSCTL_オフロード_書き込み_出力](#)

FSCTL_OPBATCH_ACK_終了_保留中の制御コード

2020/04/24 • [Edit Online](#)

FSCTL_OPBATCH_ACK_閉じる_保留中のコントロールコードは、ファイルに対する排他 (レベル1、バッチ、またはフィルター) 便宜ロック (oplock) が解除されたという通知に応答します。クライアントアプリケーションは、oplock の解除を確認し、ファイルハンドルを閉じようとしていることを示すために、この制御コードを送信します。

バッチまたはフィルター oplock 解除の場合、呼び出し元は、このコントロールコードを送信した後にファイルハンドルを閉じる必要があります。それ以外の場合は、ファイルハンドルが閉じられるのを待機している間、システムによってブロックされます。

この制御コードは、レベル1の oplock で使用するものではありません。それにもかかわらず、レベル1の oplock が解除されると、システムはこの制御コードを中断の完全な受信確認として扱い、呼び出し元はファイルハンドルを閉じる必要がありません。

この制御コードはほとんど使用されません。クライアントアプリケーションにファイルの oplock 解除が通知され、ファイルのハンドルが閉じられると、ファイルハンドルは oplock ブレークの完全な確認として扱われます。このため、この制御コードを送信する必要はありません。

この制御コードを処理するために、ミニフィルターは次のパラメーターを使用して **FltOplockFsctrl** を呼び出します。ファイルシステムまたはレガシフィルタードライバは、**FsRtlOplockFsctrl** を呼び出します。

便宜的ロックおよび FSCTL_OPBATCH_ACK の詳細については、_保留中の制御コードを閉じる_については、Microsoft Windows SDK のドキュメントを参照してください。

Parameters

Oplock

ファイルの不透明な oplock オブジェクトポインターです。

データの対

FltOplockFsctrlのみです。IRP_MJ_ファイル_システム_CONTROL FSCTL request のコールバックデータ (**FLT_callback_data**) 構造。操作の *Fscontrolcode* パラメーターを FSCTL_opbatch_ACK_CLOSE_PENDING を閉じる必要があります。

Irp

FsRtlOplockFsctrlのみです。Irp_MJ_ファイル_システム_CONTROL FSCTL 要求の IRP。操作の *Fscontrolcode* パラメーターを FSCTL_opbatch_ACK_CLOSE_PENDING を閉じる必要があります。

OpenCount

この操作では使用されません。を0に設定します。

ステータス ブロック

FltOplockFsctrlは、この操作の FLT_preop_COMPLETE を常に返します。

FsRtlOplockFsctrlは、この操作に対して次のいずれかの NTSTATUS 値を返します。

STATUS_SUCCESS	このハンドルによって保持されている oplock が壊れています。
STATUS_INVALID_OPLOCK_PROTOCOL	このハンドルで oplock が保持されていないか、または oplock ブレークが現在実行中ではありません。これはエラーコードです。

要件

Header	Ntifs (Ntifs または Fltkernel .h を含む)

関連項目

- [FLT_コールバック_データ](#)
- [IRP_MJ_ファイルの FLT_パラメーター_システム_コントロール](#)
- [FltOplockFsctrl](#)
- [FSCTL_OPLOCK_中断_ACK_NO_2](#)
- [FSCTL_OPLOCK_中断_確認](#)
- [FSCTL_OPLOCK_中断_通知](#)
- [FSCTL_要求_バッチ_OPLOCK](#)
- [FSCTL_要求_フィルター_OPLOCK](#)
- [FSCTL_要求_OPLOCK_レベル_1](#)
- [FSCTL_要求_OPLOCK_レベル_2](#)
- [FsRtlOplockFsctrl](#)
- [IRP_MJ_ファイル_システム_コントロール](#)

FSCTL_OPLOCK_中断_ACK_2 制御コードなし

2020/04/24 • [Edit Online](#)

FSCTL_OPLOCK_ は、ファイルに対する排他 (レベル1、バッチ、またはフィルター) 便宜ロック (oplock) が解除されたという通知に対して、_2 の制御コードが応答しなくなったことを **中断__** ます。

クライアントアプリケーションは、oplock の解除を確認することを示すためにこの制御コードを送信します。また、oplock がレベル2に違反したレベル1の oplock の場合、レベル2の oplock は必要ありません。

この制御コードを処理するために、ミニフィルターは次のパラメーターを使用して **FltOplockFsctrl** を呼び出します。ファイルシステムまたはレガシフィルタードライバーは、**FsRtlOplockFsctrl** を呼び出します。

便宜的ロックの詳細と、FSCTL_OPLOCK_ の詳細については、_2 の制御コードを使用せずに __ACK を _ 中断するには、Microsoft Windows SDK のドキュメントを参照してください。

Parameters

Oplock

ファイルの不透明な oplock オブジェクトポインターです。

データの対

FltOplockFsctrl のみです。IRP_MJ_ファイル_システム_CONTROL FSCTL request のコールバックデータ (**FLT_callback_data**) 構造。操作の *Fscontrolcode* パラメーターを FSCTL_OPLOCK_、_2 ではなく __ACK を中断する必要があります。

Irp

FsRtlOplockFsctrl のみです。Irp_MJ_ファイル_システム_CONTROL FSCTL 要求の IRP。操作の *Fscontrolcode* パラメーターを FSCTL_OPLOCK_、_2 ではなく __ACK を中断する必要があります。

OpenCount

この操作では使用されません。を0に設定します。

ステータス ブロック

FltOplockFsctrl は、この操作の FLT_preop_COMPLETE を常に返します。

FsRtlOplockFsctrl は、この操作に対して次のいずれかの NTSTATUS 値を返します。

II	II
STATUS_SUCCESS	Oplock の解除が確認されます。残りの oplock は保持されません。
STATUS_INVALID_OPLOCK_PROTOCOL	このハンドルで oplock が保持されていないか、または oplock ブレークが現在実行中ではありません。これはエラーコードです。

要件

Header	Ntifs (Ntifs または Fltkernel .h を含む)

関連項目

[FLT_コールバック_データ](#)

[FLT_パラメーター](#)

[IRP_MJ_ファイルの FLT_パラメーター_システム_コントロール](#)

[FltOplockFsctrl](#)

[FSCTL_OPBATCH_ACK_終了_保留中](#)

[FSCTL_OPLOCK_中断_確認](#)

[FSCTL_OPLOCK_中断_通知](#)

[FSCTL_要求_バッチ_OPLOCK](#)

[FSCTL_要求_フィルター_OPLOCK](#)

[FSCTL_要求_OPLOCK_レベル_1](#)

[FSCTL_要求_OPLOCK_レベル_2](#)

[FsRtlOplockFsctrl](#)

[IRP_MJ_ファイル_システム_コントロール](#)

FSCTL_OPLOCK_制御コードの中断_確認

2020/04/24 • [Edit Online](#)

FSCTL_OPLOCK_、ファイルに対する排他 (レベル1、バッチ、またはフィルター) 便宜ロック (oplock) が解除されたことを示す通知に制御コードが応答することを_ます。

クライアントアプリケーションは、oplock の解除を確認することを示すためにこの制御コードを送信します。また、oplock がレベル2に違反したレベル1の oplock の場合、レベル2の oplock が必要になります。

この制御コードを処理するために、ミニフィルターは次のパラメーターを使用して**FltOplockFsctrl**を呼び出します。ファイルシステムまたはレガシフィルタードライバは、**FsRtlOplockFsctrl**を呼び出します。

便宜的ロックおよびFSCTL_OPLOCKの詳細については_制御コードの_を確認するには Microsoft Windows SDK のドキュメントを参照してください。

Parameters

Oplock

ファイルの不透明な oplock オブジェクトポインターです。

データの対

FltOplockFsctrlのみです。IRP_MJ_ファイル_システム_CONTROL FSCTL request のコールバックデータ (**FLT_callback_data**) 構造。操作の*Fscontrolcode*パラメーターを FSCTL_OPLOCK_て、_の確認を中断する必要があります。

Irp

FsRtlOplockFsctrlのみです。Irp_MJ_ファイル_システム_CONTROL FSCTL 要求の IRP。操作の*Fscontrolcode*パラメーターを FSCTL_OPLOCK_て、_の確認を中断する必要があります。

OpenCount

この操作では使用されません。を0に設定します。

ステータス ブロック

レベル1の oplock がレベル2に分割され、レベル2の oplock が許可されている場合、**FltOplockFsctrl**はこの操作に対して FLT_preop_PENDING を返します。それ以外の場合は、FLT_PREOP_COMPLETE が返されます。

FsRtlOplockFsctrlは、この操作に対して次のいずれかの NTSTATUS 値を返します。

「	「
STATUS_SUCCESS	Oplock の解除が確認されます。残りの oplock は保持されません。
STATUS_INVALID_OPLOCK_PROTOCOL	このハンドルで oplock が保持されていないか、または oplock ブレークが現在実行中ではありません。これはエラーコードです。
■	Oplock の解除が確認されます。返されると、FSCTL_OPLOCK_BREAK_ACKNOWLEDGEコントロールコードの送信者はレベル2の OPLOCK を保持します。これは成功コードです。

要件

Header	Ntifs (Ntifs または Fltkernel .h を含む)

関連項目

- [FLT_コールバック_データ](#)
- [FLT_パラメーター](#)
- [IRP_MJ_ファイルの FLT_パラメーター_システム_コントロール](#)
- [FltOplockFsctrl](#)
- [FSCTL_OPBATCH_ACK_終了_保留中](#)
- [FSCTL_OPLOCK_中断_ACK_NO_2](#)
- [FSCTL_OPLOCK_中断_通知](#)
- [FSCTL_要求_バッチ_OPLOCK](#)
- [FSCTL_要求_フィルター_OPLOCK](#)
- [FSCTL_要求_OPLOCK_レベル_1](#)
- [FSCTL_要求_OPLOCK_レベル_2](#)
- [FsRtlOplockFsctrl](#)
- [IRP_MJ_ファイル_システム_コントロール](#)

FSCTL_OPLOCK_中断_通知制御コード

2020/04/24 • [Edit Online](#)

FSCTL_OPLOCK_中断_通知制御コードでは、呼び出し元のアプリケーションが便宜的ロック (oplock) 中断の完了を待機することを許可します。

この操作は、呼び出し元のハンドルを開いたときに既に開始されていた oplock の中断に対してのみ有効です。ハンドルが OPLOCKED_場合は、ファイル_完了_で開かれている必要があります。この操作は、oplock が現在保持されていて、oplock の中断が開始されていない場合は意味がありません。

この制御コードを処理するために、ミニフィルターは次のパラメーターを使用して **FltOplockFsctrl** を呼び出します。ファイルシステムまたはレガシフィルタードライバーは、**FsRtlOplockFsctrl** を呼び出します。

便宜的ロックの詳細および FSCTL_OPLOCK の詳細については、制御コードの通知_中断するには Microsoft Windows SDK のドキュメントを参照してください。

Parameters

Oplock

ファイルの不透明な oplock オブジェクトポインターです。

データの対

FltOplockFsctrlのみです。IRP_MJ_ファイル_システム_CONTROL FSCTL request のコールバックデータ (**FLT_callback_data**) 構造。操作の *Fscontrolcode* パラメーターを FSCTL_OPLOCK_中断_通知にする必要があります。

Irp

FsRtlOplockFsctrlのみです。Irp_MJ_ファイル_システム_CONTROL FSCTL 要求の IRP。操作の *Fscontrolcode* パラメーターを FSCTL_OPLOCK_中断_通知にする必要があります。

OpenCount

この操作では使用されません。を0に設定します。

ステータス ブロック

Oplock の中断が実行されている場合、**FltOplockFsctrl**は FLT_preop_PENDING を返し、oplock の解除が完了すると IRP は完了します。(この場合、IRP は、正常に完了したか、状態_取り消された状態で完了_ます)。それ以外の場合、**FltOplockFsctrl**は FLT_preop_COMPLETE を返します。

FsRtlOplockFsctrlは、この操作に対して次のいずれかの NTSTATUS 値を返します。

「	「
STATUS_SUCCESS	このハンドルで oplock が保持されていないか、oplock が保持されており、oplock ブレークが開始されていません。
STATUS_INVALID_OPLOCK_PROTOCOL	IRP は、FSCTL_OPLOCK_BREAK_NOTIFY 操作が完了する前に取り消されました。

⌈	⌈
■	Oplock の中断が進行中です。Oplock の解除が完了すると、IRP は完了します。IRP は、STATUS_SUCCESS または STATUS_CANCELLED のいずれかを使用して最終的に完了できます。これは成功コードです。

要件

Header	Ntifs (Ntifs または Fltkernel .h を含む)

関連項目

- [FLT_コールバック_データ](#)
- [FLT_パラメーター](#)
- [IRP_MJ_ファイルの FLT_パラメーター_システム_コントロール](#)
- [FtOplockFsctrl](#)
- [FSCTL_OPBATCH_ACK_終了_保留中](#)
- [FSCTL_OPLOCK_中断_ACK_NO_2](#)
- [FSCTL_OPLOCK_中断_確認](#)
- [FSCTL_要求_バッチ_OPLOCK](#)
- [FSCTL_要求_フィルター_OPLOCK](#)
- [FSCTL_要求_OPLOCK_レベル_1](#)
- [FSCTL_要求_OPLOCK_レベル_2](#)
- [FsRtlOplockFsctrl](#)
- [IRP_MJ_ファイル_システム_コントロール](#)

FSCTL_クエリ_ファイル_レイアウトコントロール コード

2020/04/24 • [Edit Online](#)

FSCTL_クエリ_ファイル_レイアウトコントロールコードは、ファイルシステムボリュームのファイルレイアウト情報を取得します。この要求の結果は、ファイルレイアウト情報のエントリのコレクションです。返されるエントリの種類は、[クエリ_ファイル_レイアウト_入力](#)構造に含めるフラグを設定することによって制御されます。必要に応じて、一連のファイル範囲を指定して、レイアウト情報の選択を制限することで、結果をフィルター処理できます。

この操作を実行するには、次のパラメーターを使用して[Fltfscontrolfile](#)または[zwfscontrolfile](#)を呼び出します。

Parameters

ファ

[Fltfscontrolfile](#)のみ。ファイルシステムボリュームのファイルオブジェクトポインター。このパラメーターは必須であり、NULLにすることはできません。

FileHandle

[Zwfscontrolfile](#)のみ。ファイルシステムボリュームのファイルハンドル。このパラメーターは必須であり、NULLにすることはできません。

FsControlCode

操作の制御コード。この操作には、FSCTL_クエリ_ファイル_レイアウトコントロールコードを使用します。

InputBuffer

呼び出し元によって割り当てられた[クエリ_ファイル_レイアウト_入力](#)構造体へのポインター。この構造体には、選択オプションが含まれています。[クエリ_ファイル_レイアウト_入力](#)の後に、オプションのファイルのエクステントが含まれます。

InputBufferLength

*InputBuffer*パラメーターが指すバッファのサイズ (バイト単位)。 *Numberofpairs* > 0 の場合、 *InputBuffer*のサイズは少なくとも `sizeof(クエリ_ファイル_LAYOUT_INPUT) + (sizeof(Filter) * (numberofpairs))` である必要があります。それ以外の場合、 *Inputbufferlength* = `sizeof(クエリ_ファイル_レイアウト_入力)` に設定します。

OutputBuffer

呼び出し元によって割り当てられた[クエリ_ファイル_レイアウト_出力](#)構造体へのポインター。これは、このバッファで後に続くファイルレイアウトエントリのヘッダーです。

OutputBufferLength

*Outputbuffer*パラメーターが指すバッファのサイズ (バイト単位)。 *Outputbuffer*のサイズは、返されるファイルレイアウトおよびストリームレイアウト構造と共に、[_レイアウト_出力_ファイル](#)を格納するのに十分な大きさである必要があります。

ステータス ブロック

[Fltfscontrolfile](#)または[Zwfscontrolfile](#)は、次のいずれかの値のように、STATUS_SUCCESS または適切な NTSTATUS 値を返します。

「	「
STATUS_INVALID_PARAMETER	ファイルシステムボリュームがオープンユーザーボリュームではないか、バッファ長が正しくないか、または無効なクエリオプションが設定されています。

㊦	㊦
STATUS_ACCESS_DENIED	呼び出し元がファイルシステムボリュームにアクセスできません。
STATUS_INVALID_USER_BUFFER	<i>InputBuffer</i> または <i>outputbuffer</i> のポインターが適切にアラインされていません。
STATUS_BUFFER_TOO_SMALL	<i>Outputbufferlength</i> の値は、 <i>outputbuffer</i> が少なくとも1つのレイアウトエントリを格納するには小さすぎることを示します。
STATUS_END_OF_FILE	<i>InputBuffer</i> または <i>outputbuffer</i> のポインターが適切にアラインされていません。

注釈

ボリュームのすべてのレイアウトエントリを取得するために、FSCTL_クエリ_ファイル_レイアウト要求は、_ファイルのステータス_終了_が返されるまで複数回発行されます。状態_SUCCESSが返されますが、呼び出し元は、残りのレイアウトエントリに対して_レイアウト要求に対して、FSCTL_クエリ_ファイルの送信を続行できます。

レイアウトエントリの列挙は、クエリ_ファイル_レイアウト_入力のFlagsメンバーにクエリ_ファイル_レイアウト_再起動フラグを含めることによって、いつでも再起動できます。また、FSCTL_QUERY_FILE_LAYOUT が_ファイルの終了_を_ 返した後、次の FSCTL にクエリ_ファイル_レイアウト_再起動フラグを含める必要があり @no新しいレイアウトエントリの列挙を開始するには、クエリ_ファイル_レイアウト要求を実行します。

**ReFS: **このコードはサポートされていません。

要件

バージョン	Windows 8 以降で使用できます。
Header	Ntifs (Ntifs を含む)

関連項目

[クエリ_ファイル_レイアウト_入力](#)

[クエリ_ファイル_レイアウト_出力](#)

[FtFsControlFile](#)

[ZwFsControlFile](#)

FSCTL_クエリ_永続_ボリューム_状態制御コード

2020/04/24 • [Edit Online](#)

FSCTL_クエリ_永続的_ボリューム_状態制御コードは、ファイルシステムボリュームの永続的な設定を取得します。永続設定は、コンピューターの再起動の間にファイルシステムボリュームに保持されます。

この操作を実行するには、次のパラメーターを使用して`Fltfscontrolfile`または`zwfscontrolfile`を呼び出します。

Parameters

フア

`Fltfscontrolfile`のみ。ファイルシステムボリュームのファイルオブジェクトポインター。このパラメーターは必須であり、NULLにすることはできません。

FileHandle

`Zwfscontrolfile`のみ。ファイルシステムボリュームのファイルハンドル。このパラメーターは必須であり、NULLにすることはできません。

FsControlCode

操作の制御コード。この操作には、FSCTL_クエリ_永続的_ボリューム_状態を使用してください。

InputBuffer

__FS によって割り当てられたファイルへのポインターは、`永続_ボリューム_情報`の構造を保持します。

InputBufferLength

`InputBuffer`パラメーターが指すバッファのサイズ (バイト単位)。

OutputBuffer

__FS によって割り当てられたファイルへのポインターは、ファイルシステムボリュームの永続的な設定を受け取る`永続的な_ボリューム_情報`構造を保持します。

OutputBufferLength

`Outputbuffer`パラメーターが指すバッファのサイズ (バイト単位)。

ステータス ブロック

`Fltfscontrolfile`または`zwfscontrolfile`は、次のいずれかのような状態_SUCCESS または適切な NTSTATUS 値を返します。

「	「
STATUS_NOT_SUPPORTED	呼び出し元が、 <code>FILE_FS_PERSISTENT_VOLUME_INFORMATION</code> のversionメンバーで正しくないバージョン番号を指定しました。
STATUS_INVALID_PARAMETER	ファイルシステムボリュームがオープンユーザーボリュームではないか、呼び出し元が <code>FILE_FS_PERSISTENT_VOLUME_INFORMATION</code> のflagmaskメンバーに無効なフラグを指定しました。

II	II
STATUS_BUFFER_TOO_SMALL	<i>InputBuffer</i> パラメーターが指すバッファーが十分な大きさではありません (つまり、バッファーがsizeof(FILE_FS_PERSISTENT_VOLUME_INFORMATION)未満)。この場合、永続的な設定データは返されません。これはエラーコードです。
STATUS_ACCESS_DENIED	呼び出し元がファイルシステムボリュームにアクセスできません。
STATUS_VOLUME_DISMOUNTED	ファイルシステムボリュームがマウント解除されています。
STATUS_TOO_LATE	ファイルシステムボリュームがシャットダウンされています。
STATUS_MEDIA_WRITE_PROTECTED	ファイルシステムボリュームは読み取り専用です。

要件

バージョン	Windows 7 以降で使用できます。
Header	Ntifs (Ntifs を含む)

関連項目

[ファイル_FS_永続的な_ボリューム_情報](#)

[FltFsControlFile](#)

[ZwFsControlFile](#)

FSCTL_クエリ_REFS_SMR_ボリューム_情報制御 コード

2020/04/24 • [Edit Online](#)

FSCTL_クエリ_REFS_SMR_ボリューム_情報制御コードは、領域とガベージコレクションアクティビティの現在の状態を Shingled 磁気記録 (smr) ボリュームに照会します。

この操作を実行するには、次のパラメーターを使用して **DeviceIoControl** 関数を呼び出します。

```
BOOL
DeviceIoControl( (HANDLE)      hDevice,          // handle to device
                 (DWORD)       FSCTL_QUERY_REFS_SMR_VOLUME_INFO, // dwIoControlCode
                 (LPDWORD)     lpInBuffer,       // input buffer
                 (DWORD)       nInBufferSize,    // size of input buffer
                 (LPDWORD)     lpOutBuffer,       // output buffer
                 (DWORD)       nOutBufferSize,    // size of output buffer
                 (LPDWORD)     lpBytesReturned,   // number of bytes returned
                 (LPOVERLAPPED) lpOverlapped );   // OVERLAPPED structure
```

パラメーター

Hdevice []

デバイスへのハンドル。デバイスハンドルを取得するには、**CreateFile**関数を呼び出します。

] の *Dwiocontrolcode* [

操作の制御コード。この操作には、FSCTL_クエリ_REFS_SMR_ボリューム_情報を使用します。

Lpinbuffer

この操作では使用されません。をNULLに設定します。

] の *nInBufferSize* [

この操作では使用されません。を0に設定します。

Lpoutbuffer [out]

領域およびガベージコレクションアクティビティのボリュームの現在の状態を指定する参照 **_SMR_ボリューム_情報_出力** 構造体を受け取るバッファーへのポインター。

] の *Noutbuffersize* [

出力バッファーのサイズ (バイト単位)。

Lpbytesreturned [out] を返しました

出力バッファーに格納されているデータのサイズ (バイト単位) を受け取る変数へのポインター。

出力バッファーが小さすぎると、呼び出しが失敗し、**GetLastError**がエラー **_バッファーが不足し、lpbytesreturned0** を返します。

*LpOverlapped*がnullの場合、*lpbytesreturned* nullにすることはできません。操作によって出力データが返されず、*Lpoutbuffer*がNULLの場合でも、**DeviceIoControl**は*lpoutbuffer*を使用して返されます。このような操作を行った後、返された *Lpbytesreturned* 値は無意味になります。

*LpOverlapped*がnullでない場合は、*lpbytesreturned* nullになることがあります。このパラメーターがNULLでなく、操作がデータを返す場合は、重複操作が完了するまで、*lpbytesreturned* 返されることは意味がありません。返されるバイト数を取得するには、**GetOverlappedResult**を呼び出します。*Hdevice*パラメーターが i/o 完了ポートに関連付けられている場合は、

[GetQueuedCompletionStatus](#)を呼び出すことによって返されるバイト数を取得できます。

] の *lpOverlapped* [
[オーバーラップ](#)された構造体へのポインター。

ファイル__フラグを指定せずに *hdevice*を開いた場合、*lpOverlapped*は無視されます。

*Hdevice*がファイル_フラグを使用して開かれている場合は、重複フラグ_、この操作は、オーバーラップ (非同期) 操作として実行されます。この場合、*lpOverlapped*は、イベントオブジェクトへのハンドルを含む有効な[オーバーラップ](#)構造体を指す必要があります。それ以外の場合、関数は予期しない方法で失敗します。

オーバーラップ操作の場合、[DeviceIoControl](#)はすぐに制御を返し、操作が完了するとイベントオブジェクトを通知します。それ以外の場合、この関数は、操作が完了するかエラーが発生するまで、を返しません。

戻り値

操作が正常に完了した場合、[DeviceIoControl](#)は0以外の値を返します。

操作が失敗した場合、または保留中の場合、[DeviceIoControl](#)は0を返します。エラーの詳細情報を取得するには、[GetLastError](#)を呼び出します。

要件

バージョン	Windows 10 バージョン1709以降で使用できます。
Header	Winioctl. h

関連項目

[DeviceIoControl](#)

[REFS_SMR_ボリューム_情報_出力](#)

[REFS_SMR_ボリューム_GC_状態](#)

FSCTL_クエリ_取得_ポインター制御コード

2020/04/24 • [Edit Online](#)

FSCTL_クエリ_取得_ポインター制御コードは、仮想クラスター番号 (VCN、ファイル/ストリーム領域内のオフセット)、および論理クラスター番号 (LCN、ボリューム領域内のオフセット) 間のマッピングを取得します。*InputBuffer*で指定されたマップサイズまでのファイルの先頭から開始します。

FSCTL_クエリ_取得_ポインターは、[ポインターを取得](#) 取得する_に似ています。ただし、FSCTL_クエリ_取得_ポインターは、ローカルページングファイルまたはシステムハイブのカーネルモードでのみ機能します。ページングファイルには、ボリューム内の VCN から、基になる物理記憶域をより直接参照する LCN まで、1対1のマッピングがあることが保証されています。FSCTL_使用しないでください。この場合、ページファイル以外のファイルを使用したポインター_ポインター_取得する必要があります。これは、Vcns から Lcns への一対多マッピングを持つボリューム (ミラーボリュームなど) に存在する可能性があるためです。

この操作を実行するには、次のパラメーターを使用して [Fltfscontrolfile](#) または [zwfscontrolfile](#) を呼び出します。

パラメーター

フラ

[Fltfscontrolfile](#)のみ。ページングファイルまたは休止状態ファイルのファイルオブジェクトポインター。このパラメーターは必須であり、NULLにすることはできません。

FileHandle

[Zwfscontrolfile](#)のみ。ページングファイルまたは休止状態ファイルのファイルハンドル。このパラメーターは必須であり、NULLにすることはできません。

FsControlCode

操作の制御コード。FSCTL を使用して、この操作の_ポインターを取得_取得_します。

InputBuffer

マップサイズを指定する quadlet へのポインターを格納するユーザー指定のバッファー。マップサイズは、マップするクラスターの数です。

InputBufferLength

*InputBuffer*の入力バッファーの長さ (バイト単位)。

OutputBuffer

次の型の要素の配列を含むページプールメモリのバッファーへのポインター。

```
struct {
    LONGLONG SectorLengthInBytes;
    LONGLONG StartingLogicalOffsetInBytes;
} MappingPair;
```

この quadlet のペアの配列は、ファイルのディスク実行を定義します。配列の最後の要素のSectorLengthInBytesメンバーの値が0です。

OutputBufferLength

*Outputbufferl*パラメーターが指すバッファーのサイズ (バイト単位)。

状態ブロック

[Fltfscontrolfile](#)と[zwfscontrolfile](#)はどちらも STATUS_SUCCESS または適切な NTSTATUS エラー値を返します。

解説

**ReFS: **このコードはサポートされていません。

前提条件

ヘッダー	Ntifs (Ntifs を含む)

関連項目

[FltFsControlFile](#)

[FSCTL_取得_ポインター_取得する](#)

[ZwFsControlFile](#)

FSCTL_オーバーレイコントロールコードの削除

2020/04/24 • [Edit Online](#)

FSCTL_REMOVE_オーバーレイコントロールコードは、バックギングソースをボリュームから削除します。

この操作を実行するには、次のパラメーターを使用して`Fltfscontrolfile`または`zwfscontrolfile`を呼び出します。

パラメーター

]のインスタンス [

`Fltfscontrolfile`のみ。呼び出し元の非透過インスタンスポインター。このパラメーターは必須であり、NULL にすることはできません。

]の FileObject [

`Fltfscontrolfile`のみ。オーバーレイの削除元となるボリュームのファイルポインターオブジェクト。このパラメーターは必須であり、NULL にすることはできません。

]の FileHandle [

`Zwfscontrolfile`のみ。オーバーレイが削除されるボリュームのハンドル。このパラメーターは必須であり、NULL にすることはできません。

]の FsControlCode [

操作の制御コード。この操作には、FSCTL_を使用して_オーバーレイを削除してください。

InputBuffer

入力バッファーへのポインター。これには、`WOF_外部_INFO`構造体が含まれている必要があります。必要に応じて、追加のプロバイダー固有のデータは、`WOF_外部_情報`に含まれます。プロバイダーが WIM ファイルの場合、`wim_プロバイダー_削除_オーバーレイ_入力`構造は、`WOF_外部_情報`に含まれています。

]内の InputBufferLength [

Sizeof(`WOF_外部_INFO`) に、追加のプロバイダー入力データのサイズを加えた値に設定します。

OutputBuffer [out]

使用されません。NULL に設定されている。

]の OutputBufferLength [

を0に設定します。

状態ブロック

`Fltfscontrolfile`または`zwfscontrolfile`は、操作が成功した場合に STATUS_SUCCESS を返します。それ以外の場合、適切な関数は、次のいずれかの NTSTATUS 値を返す可能性があります。

「	「
STATUS_ACCESS_DENIED	要求元には管理者特権がありません。
STATUS_BUFFER_TOO_SMALL	<i>Outputbuffer</i> によってポイントされ、 <i>outputbufferlength</i> によって指定された出力バッファーの長さが小さすぎます。
STATUS_INTERNAL_ERROR	要求されたボリュームにアクセスできません。

㊦	㊦
STATUS_INVALID_DEVICE_REQUEST	バックギグサービスが存在しないか、開始されていません。

解説

削除するバックギグソースが Windows Imaging Format (WIM) ファイルの場合、入力バッファーには、[WOF_外部_INFO](#) 構造体の後に[WIM__プロバイダー](#)が含まれます。これにより、オーバーレイ_入力が削除されます。データ_ この場合、*Inputbufferlength*はsizeof(WOF_外部_INFO) + sizeof(WIM_プロバイダー__オーバーレイ_入力) を削除します。Wim_PROVIDER のDataSourceId値 __オーバーレイの削除_入力は、[FSCTL_オーバーレイ](#) 要求の追加で以前に追加された wim ファイルに対して行う必要があります。_

追加のバックギグプロバイダーは、独自の特定の入力パラメーター構造を定義します。

前提条件

バージョン	Windows 8.1 更新プログラムから使用できます。
ヘッダー	Ntifs (Ntifs または Fltkernel .h を含む)

関連項目

[FSCTL_SUSPEND_オーバーレイ](#)

[FSCTL_UPDATE_オーバーレイ](#)

[FSCTL_外部_バックギグ_取得する](#)

[外部_バックギグ_FSCTL_設定](#)

FSCTL_要求_バッチ_OPLOCK 制御コード

2020/04/24 • [Edit Online](#)

FSCTL_要求_batch_oplock制御コードは、ファイルにバッチ便宜ロック (oplock) を要求します。

この制御コードを処理するために、ミニフィルターは次のパラメーターを使用して**FltOplockFsctrl**を呼び出します。ファイルシステムまたはレガシフィルタードライバは、**FsRtlOplockFsctrl**を呼び出します。

便宜的ロックの詳細、およびバッチ_OPLOCK_要求の詳細については、Microsoft Windows SDK のドキュメントを参照してください。

パラメーター

Oplock

ファイルの不透明な oplock オブジェクトポインターです。

データの対

FltOplockFsctrlのみです。IRP_MJ_ファイル_システム_CONTROL FSCTL request のコールバックデータ (**FLT_callback_data**) 構造。操作の*Fscontrolcode*パラメーターは、BATCH_OPLOCK_FSCTL_要求である必要があります。

Irp

FsRtlOplockFsctrlのみです。Irp_MJ_ファイル_システム_CONTROL FSCTL 要求の IRP。操作の*Fscontrolcode*パラメーターは、BATCH_OPLOCK_FSCTL_要求である必要があります。

OpenCount

ファイルのユーザーハンドルの数。

状態ブロック

Oplock が付与されている場合、**FltOplockFsctrl**は、この操作に対して FLT_preop_PENDING を返します。それ以外の場合は、FLT_PREOP_COMPLETE が返されます。

FsRtlOplockFsctrlは、この操作に対して次のいずれかの NTSTATUS 値を返します。

「	「
■	Oplock が付与されました。これは成功コードです。
STATUS_CANCELLED	IRP は、FSCTL_REQUEST_BATCH_OPLOCK 操作が完了する前に取り消されました。これはエラーコードです。
STATUS_OPLOCK_NOT_GRANTED	Oplock を許可できませんでした。これはエラーコードです。

前提条件

ヘッダー	Ntifs (Ntifs または Fltkernel .h を含む)

関連項目

FLT_コールバック_データ

FLT_パラメーター

IRP_MJ_ファイルの FLT_パラメーター_システム_コントロール

FltOplockFsctrl

FSCTL_OPBATCH_ACK_終了_保留中

FSCTL_OPLOCK_中断_ACK_NO_2

FSCTL_OPLOCK_中断_確認

FSCTL_OPLOCK_中断_通知

FSCTL_要求_フィルター_OPLOCK

FSCTL_要求_OPLOCK_レベル_1

FSCTL_要求_OPLOCK_レベル_2

FsRtlOplockFsctrl

IRP_MJ_ファイル_システム_コントロール

FSCTL_要求_フィルター_OPLOCK 制御コード

2020/04/24 • [Edit Online](#)

FSCTL_要求_フィルター_oplock制御コードは、ファイルに対してフィルター便宜のロック (oplock) を要求します。

この制御コードを処理するために、ミニフィルターは次のパラメーターを使用して **FltOplockFsctrl** を呼び出します。ファイルシステムまたはレガシフィルタードライバーは、**FsRtlOplockFsctrl** を呼び出します。

便宜のロックおよびFSCTL_要求_フィルター_OPLOCKの詳細については、Microsoft Windows SDK のドキュメントを参照してください。

パラメーター

Oplock

ファイルの不透明な oplock オブジェクトポインターです。

データの対

FltOplockFsctrlのみです。IRP_MJ_ファイル_システム_CONTROL FSCTL request のコールバックデータ (**FLT_callback_data**) 構造。操作の *Fscontrolcode* パラメーターは、FSCTL_REQUEST_FILTER_OPLOCK である必要があります。

Irp

FsRtlOplockFsctrlのみです。Irp_MJ_ファイル_システム_CONTROL FSCTL 要求の IRP。操作の *Fscontrolcode* パラメーターは、FSCTL_REQUEST_FILTER_OPLOCK である必要があります。

OpenCount

ファイルのユーザーハンドルの数。

状態ブロック

Oplock が付与されている場合、**FltOplockFsctrl** は、この操作に対して FLT_preop_PENDING を返します。それ以外の場合は、FLT_PREOP_COMPLETE が返されます。

FsRtlOplockFsctrl は、この操作に対して次のいずれかの NTSTATUS 値を返します。

■	■
■	Oplock が付与されました。これは成功コードです。
STATUS_CANCELLED	IRP は、FSCTL_REQUEST_BATCH_OPLOCK 操作が完了する前に取り消されました。これはエラーコードです。
STATUS_OPLOCK_NOT_GRANTED	Oplock を許可できませんでした。これはエラーコードです。

前提条件

ヘッダー	Ntifs (Ntifs または Fltkernel .h を含む)

関連項目

[FLT_コールバック_データ](#)

[FLT_パラメーター](#)

[IRP_MJ_ファイルの FLT_パラメーター_システム_コントロール](#)

[FltOplockFsctrl](#)

[FSCTL_OPBATCH_ACK_終了_保留中](#)

[FSCTL_OPLOCK_中断_ACK_NO_2](#)

[FSCTL_OPLOCK_中断_確認](#)

[FSCTL_OPLOCK_中断_通知](#)

[FSCTL_要求_バッチ_OPLOCK](#)

[FSCTL_要求_OPLOCK_レベル_1](#)

[FSCTL_要求_OPLOCK_レベル_2](#)

[FsRtlOplockFsctrl](#)

[IRP_MJ_ファイル_システム_コントロール](#)

FSCTL_要求_OPLOCK 制御コード

2020/04/24 • [Edit Online](#)

FSCTL_要求_oplock制御コードは、ファイルに対して便宜的ロック (oplock) を要求するか、oplock の解除が発生したことを確認します。

便宜的ロックの詳細については、Windows デスクトップのドキュメントの「[便宜的ロック](#)」を参照してください。ユーザーモードの OPLOCK コントロールの詳細については、Windows デスクトップのドキュメントの「[ファイル管理制御コード](#)」を参照してください。

この制御コードを処理するために、ファイルシステムまたはフィルタードライバーは、次のパラメーターを使用して [FsRtlOplockFsctrlEx](#) を呼び出します。

Parameters

Oplock

ファイルの不透明な oplock オブジェクトポインターです。

Irp

IRP_MJ_ファイル_システム_CONTROL FSCTL 要求の IRP へのポインター。操作の *Fscontrolcode* パラメーターは、FSCTL_要求_OPLOCK である必要があります。

OpenCount

要求が排他的な oplock の場合は、ファイルのユーザーハンドルの数。要求が共有可能な oplock に対するものである場合、ファイルにバイト範囲ロックが存在しない場合、*Opencount* は 0 になります。それ以外の場合、*Opencount* は 0 以外になります。呼び出し元は、IRP で [FsRtlOplockIsSharedRequest](#) ルーチンを呼び出して、要求が共有可能な oplock 用かどうかを判断できます。

示す

関連付けられた oplock 操作のビットマスク。ファイルシステムまたはフィルタードライバーは、ビットを設定して [FsRtlOplockFsctrlEx](#) の動作を指定します。*Flags* パラメーターには、次のオプションがあります。

OPLOCK_FSCTRL_フラグ_すべての__キーに一致する (0x00000001)

現在開いているハンドルですべての便宜的ロックキーが一致することをファイルシステムが検証したことを示します。このフラグを指定することにより、oplock パッケージは、ファイルに対して複数の開いているハンドルが存在する場合に、レベル RW または RWH の oplock を与えることができます。Oplock の種類の詳細については、「[概要](#)」を参照してください。

ステータス ブロック

[FsRtlOplockFsctrlEx](#) は、この操作に対して次のいずれかの NTSTATUS 値を返します。

「	「
STATUS_PENDING	Oplock が付与されました。これは成功コードです。
STATUS_CANCELLED	IRP は、FSCTL_REQUEST_OPLOCK 操作が完了する前に取り消されました。これはエラーコードです。
STATUS_OPLOCK_NOT_GRANTED	Oplock を許可できませんでした。これはエラーコードです。

要件

ヘッダー	Ntifs (Ntifs または Fltkernel .h を含む)

参照

[FsRtlOplockFsctrlEx](#)

[FsRtlOplockIsSharedRequest](#)

[IRP_MJ_ファイル_システム_コントロール](#)

FSCTL_要求_OPLOCK_レベル_1 コントロールコード

2020/04/24 • [Edit Online](#)

FSCTL_要求_OPLOCK_レベル_1 コントロールコードは、ファイルにレベル1の便宜ロック (oplock) を要求します。

この制御コードを処理するために、ミニフィルターは次のパラメーターを使用して **FltOplockFsctrl** を呼び出します。ファイルシステムまたはレガシフィルタードライバーは、**FsRtlOplockFsctrl** を呼び出します。

便宜的ロックの詳細およびFSCTL_要求_OPLOCK_レベル_1 コントロールコードについては、Microsoft Windows SDK のドキュメントを参照してください。

Parameters

Oplock

ファイルの不透明な oplock オブジェクトポインターです。

データの対

FltOplockFsctrl のみです。IRP_MJ_ファイル_システム_CONTROL FSCTL request のコールバックデータ (**FLT_callback_data**) 構造。操作の *Fscontrolcode* パラメーターは、FSCTL_要求_OPLOCK_レベル_1 である必要があります。

Irp

FsRtlOplockFsctrl のみです。Irp_MJ_ファイル_システム_CONTROL FSCTL 要求の IRP。操作の *Fscontrolcode* パラメーターは、FSCTL_要求_OPLOCK_レベル_1 である必要があります。

OpenCount

ファイルのユーザーハンドルの数。

ステータス ブロック

Oplock が付与されている場合、**FltOplockFsctrl** は、この操作に対して FLT_preop_PENDING を返します。それ以外の場合は、FLT_PREOP_COMPLETE が返されます。

FsRtlOplockFsctrl は、この操作に対して次のいずれかの NTSTATUS 値を返します。

「	「
■	Oplock が付与されました。これは成功コードです。
STATUS_CANCELLED	IRP は、FSCTL_REQUEST_OPLOCK_LEVEL_1 操作が完了する前に取り消されました。これはエラーコードです。
STATUS_OPLOCK_NOT_GRANTED	Oplock を許可できませんでした。これはエラーコードです。

要件

Header	Ntifs (Ntifs または Fltkernel.h を含む)

関連項目

[FLT_コールバック_データ](#)

[FLT_パラメーター](#)

[IRP_MJ_ファイルの FLT_パラメーター_システム_コントロール](#)

[FltOplockFsctrl](#)

[FSCTL_OPBATCH_ACK_終了_保留中](#)

[FSCTL_OPLOCK_中断_ACK_NO_2](#)

[FSCTL_OPLOCK_中断_確認](#)

[FSCTL_OPLOCK_中断_通知](#)

[FSCTL_要求_バッチ_OPLOCK](#)

[FSCTL_要求_フィルター_OPLOCK](#)

[FSCTL_要求_OPLOCK_レベル_2](#)

[FsRtlOplockFsctrl](#)

[IRP_MJ_ファイル_システム_コントロール](#)

FSCTL_要求_OPLOCK_レベル_2 制御コード

2020/04/24 • [Edit Online](#)

FSCTL_要求_OPLOCK_レベル_2 制御コードは、ファイルにレベル2の便宜的ロック (oplock) を要求します。

この制御コードを処理するために、ミニフィルターは次のパラメーターを使用して **FltOplockFsctrl** を呼び出します。ファイルシステムまたはレガシフィルタードライバは、**FsRtlOplockFsctrl** を呼び出します。

便宜的ロックの詳細およびFSCTL_要求_OPLOCK_レベル_2 コントロールコードについては、Microsoft Windows SDK のドキュメントを参照してください。

Parameters

Oplock

ファイルの不透明な oplock オブジェクトポインターです。

データの対

FltOplockFsctrl のみです。IRP_MJ_ファイル_システム_CONTROL FSCTL request のコールバックデータ (**FLT_callback_data**) 構造。操作の *Fscontrolcode* パラメーターは、FSCTL_REQUEST_OPLOCK_レベル_2 である必要があります。

Irp

FsRtlOplockFsctrl のみです。Irp_MJ_ファイル_システム_CONTROL FSCTL 要求の IRP。操作の *Fscontrolcode* パラメーターは、FSCTL_REQUEST_OPLOCK_レベル_2 である必要があります。

OpenCount

ファイルのロック状態を指定します。ファイルにバイト範囲ロックがある場合は、このパラメーターに0以外の ULONG 値を設定します。それ以外の場合は0を設定します。

ステータス ブロック

Oplock が付与されている場合、**FltOplockFsctrl** は、この操作に対して FLT_preop_PENDING を返します。それ以外の場合は、FLT_PREOP_COMPLETE が返されます。

FsRtlOplockFsctrl は、この操作に対して次のいずれかの NTSTATUS 値を返します。

「	「
■	Oplock が付与されました。これは成功コードです。
STATUS_CANCELLED	IRP は、FSCTL_REQUEST_OPLOCK_LEVEL_2 操作が完了する前に取り消されました。これはエラーコードです。
STATUS_OPLOCK_NOT_GRANTED	Oplock を許可できませんでした。これはエラーコードです。

要件

Header	Ntifs (Ntifs または Fltkernel .h を含む)

関連項目

[FLT_コールバック_データ](#)

[FLT_パラメーター](#)

[IRP_MJ_ファイルの FLT_パラメーター_システム_コントロール](#)

[FltOplockFsctrl](#)

[FSCTL_OPBATCH_ACK_終了_保留中](#)

[FSCTL_OPLOCK_中断_ACK_NO_2](#)

[FSCTL_OPLOCK_中断_確認](#)

[FSCTL_OPLOCK_中断_通知](#)

[FSCTL_要求_バッチ_OPLOCK](#)

[FSCTL_要求_フィルター_OPLOCK](#)

[FSCTL_要求_OPLOCK_レベル_1](#)

[FsRtlOplockFsctrl](#)

[IRP_MJ_ファイル_システム_コントロール](#)

FSCTL_設定_外部_バッキングコントロールコード

2020/04/24 • [Edit Online](#)

_外部制御コード_設定された FSCTL_は、外部バッキングプロバイダーによって、Windows イメージフォーマット (WIM) ファイルや圧縮ファイルなどのファイルのバッキングソースを設定します。外部でサポートされるファイルのコンテンツは、ファイルが存在するボリューム以外のボリュームで発生する可能性があります。

この操作を実行するには、次のパラメーターを使用して **Fltfscontrolfile** または **zwfscontrolfile** を呼び出します。

Parameters

]のインスタンス [

Fltfscontrolfileのみ。呼び出し元の非透過インスタンスポインター。このパラメーターは必須であり、NULL にすることはできません。

]の FileObject [

Fltfscontrolfileのみ。バッキングが設定されているファイルのファイルポインターオブジェクト。このパラメーターは必須であり、NULL にすることはできません。

]の FileHandle [

Zwfscontrolfileのみ。バッキングが設定されているファイルのハンドル。このパラメーターは必須であり、NULL にすることはできません。

]の FsControlCode [

操作の制御コード。この操作を行うには、FSCTL_を使用して **_外部_**を設定します。

InputBuffer

入力バッファーへのポインター。これには、**WOF_外部_INFO**構造体の後にプロバイダーデータが格納されます。WIM によってサポートされるファイルの場合、**WOF_外部_情報**の後に**wim_プロバイダー_外部_情報**構造が続きます。

]内の InputBufferLength [

*InputBuffer*で提供されるデータのサイズ。

OutputBuffer [out]

なし。を NULL に設定します。

]の OutputBufferLength [

を0に設定します。

ステータス ブロック

Fltfscontrolfileまたは**zwfscontrolfile**は、操作が成功した場合に STATUS_SUCCESS を返します。それ以外の場合は、適切な NTSTATUS 値が返されます。

注釈

追加されたデータソースのバッキングプロバイダーが WIM プロバイダーである場合、入力バッファーには、**外部_情報**構造に加えて、**wim_プロバイダー_外部_情報**構造が含まれます。この場合、*Inputbufferlength*は `sizeof(WOF_外部_info) + sizeof(WIM_PROVIDER_外部_INFO)` になります。

個別に圧縮されたファイルは、変更されないデータ (実行可能ファイルを含む) に適した圧縮を提供します。これらが書き込み用に開かれている場合、ファイルは透過的に圧縮解除されます。個別に圧縮されたファイルを指定するために、入力バッファーには、**外部_情報_V1 構造_ファイル_プロバイダー_、_の_外部の情報**構造が含まれます。この場合、*Inputbufferlength*は `sizeof(WOF_外部_info) + sizeof(ファイル_プロバイダー_外部_情報_V1)` になります。個々の圧

縮ファイルは、Windows 10 以降で使用できます。

要件

バージョン	Windows 8.1 更新プログラムから使用できます。
Header	Ntifs (Ntifs または Fltkernel .h を含む)

関連項目

[FltFsControlFile](#)

[ZwFsControlFile](#)

[FSCTL__外部_の削除](#)

[FSCTL_外部_バッキング_取得する](#)

[WIM_プロバイダー_外部_情報](#)

[WOF_外部_情報](#)

FSCTL_設定_永続的_ボリューム_状態制御コード

2020/04/24 • [Edit Online](#)

FSCTL_SET_永続_ボリューム_状態制御コードは、ファイルシステムボリュームの永続的な設定を設定します。永続設定は、コンピューターの再起動の間にファイルシステムボリュームに保持されます。

この操作を実行するには、次のパラメーターを使用して`Fltfscontrolfile`または`zwfscontrolfile`を呼び出します。

Parameters

フア

`Fltfscontrolfile`のみ。ファイルシステムボリュームのファイルオブジェクトポインター。このパラメーターは必須であり、NULLにすることはできません。

FileHandle

`Zwfscontrolfile`のみ。ファイルシステムボリュームのファイルハンドル。このパラメーターは必須であり、NULLにすることはできません。

FsControlCode

操作の制御コード。この操作には、FSCTL_使用して_永続的_ボリューム_状態を設定します。

InputBuffer

_FS によって割り当てられたファイルへのポインターは、ファイルシステムボリュームの永続的な設定を含む永続的な_ボリューム_情報構造を保持します。

InputBufferLength

`InputBuffer`パラメーターが指すバッファのサイズ (バイト単位)。

OutputBuffer

この操作では使用されません。をNULLに設定します。

OutputBufferLength

この操作では使用されません。を0に設定します。

ステータス ブロック

`Fltfscontrolfile`または`zwfscontrolfile`は、次のいずれかのような状態_SUCCESS または適切な NTSTATUS 値を返します。

「	「
STATUS_NOT_SUPPORTED	要求されたレジストリ設定がボリュームごとではないか、または呼び出し元がFILE_FS_PERSISTENT_VOLUME_INFORMATIONのversionメンバーで正しくないバージョン番号を指定しました。
STATUS_INVALID_PARAMETER	ファイルシステムボリュームがオープンユーザーボリュームではないか、呼び出し元がFILE_FS_PERSISTENT_VOLUME_INFORMATIONのflagmaskメンバーに無効なフラグを指定しました。

II	II
STATUS_BUFFER_TOO_SMALL	<i>InputBuffer</i> パラメーターが指すバッファが、永続的な設定データを保持するのに十分な大きさではありません。この場合、永続的な設定データは設定されません。これはエラーコードです。
STATUS_ACCESS_DENIED	呼び出し元がファイルシステムボリュームにアクセスできません。
STATUS_VOLUME_DISMOUNTED	ファイルシステムボリュームがマウント解除されています。
STATUS_TOO_LATE	ファイルシステムボリュームがシャットダウンされています。
STATUS_MEDIA_WRITE_PROTECTED	ファイルシステムボリュームは読み取り専用です。

要件

バージョン	Windows 7 以降で使用できます。
Header	Ntifs (Ntifs を含む)

関連項目

[ファイル_FS_永続的な_ボリューム_情報](#)

[FltFsControlFile](#)

[ZwFsControlFile](#)

FSCTL_REFS_SMR_ボリューム_GC_パラメーター制御コードに設定します。

2020/04/24 • [Edit Online](#)

FSCTL_SET_REFS_SMR_ボリューム_GC_PARAMETERS制御コードは、Shingled 磁気記録 (SMR) ボリュームでガベージコレクションを制御します。

```
BOOL
DeviceIoControl( (HANDLE)      hDevice,          // handle to volume
                  FSCTL_SET_REFS_SMR_VOLUME_GC_PARAMETERS, // dwIoControlCode
                  (LPDWORD)    lpInBuffer,       // input buffer
                  (DWORD)      nInBufferSize,    // size of input buffer
                  NULL,        // output buffer
                  0,           // size of output buffer
                  (LPDWORD)    lpBytesReturned,   // number of bytes returned
                  (LPOVERLAPPED) lpOverlapped ); // OVERLAPPED structure
```

パラメーター

Hdevice []

デバイスへのハンドル。デバイスハンドルを取得するには、[CreateFile](#)関数を呼び出します。

] の *Dwiocontrolcode* [

操作の制御コード。この操作には、FSCTL_使用して、_REFS_SMR_ボリューム_GC_パラメーターを設定します。

Lpinbuffer

呼び出し元が割り当てた[REFS_SMR_ボリューム_GC_PARAMETERS](#)構造体へのポインター。

] の *nInBufferSize* [

入力バッファのサイズ (バイト単位)。

Lpoutbuffer [out]

この操作では使用されません。をNULLに設定します。

] の *Noutbuffersize* [

この操作では使用されません。を0に設定します。

Lpbytesreturned [out] を返しました

この操作では使用されません。をNULLに設定します。

] の *lpOverlapped* [

[オーバーラップ](#)された構造体へのポインター。

ファイル_フラグを指定せずに*hdevice*を開いた場合、*lpOverlapped*は無視されます。

*Hdevice*がファイル_フラグを使用して開かれている場合は、重複フラグ_、この操作は、オーバーラップ (非同期) 操作として実行されます。この場合、*lpOverlapped*は、イベントオブジェクトへのハンドルを含む有効な[オーバーラップ](#)構造体を指す必要があります。それ以外の場合、関数は予期しない方法で失敗します。

オーバーラップ操作の場合、[DeviceIoControl](#)はすぐに制御を返し、操作が完了するとイベントオブジェクトを通知します。それ以外の場合、この関数は、操作が完了するかエラーが発生するまで、を返しません。

戻り値

操作が正常に完了した場合、[DeviceIoControl](#)は0以外の値を返します。

操作が失敗した場合、または保留中の場合、[DeviceIoControl](#)は0を返します。エラーの詳細情報を取得するには、[GetLastError](#)を呼び出します。

要件

バージョン	Windows 10 バージョン1709以降で使用できます。
Header	Winioctl. h

関連項目

[DeviceIoControl](#)

FSCTL_再解析_ポイント制御コードの設定

2020/04/24 • [Edit Online](#)

再解析_ポイント制御コード_設定された FSCTL_は、ファイルまたはディレクトリの再解析ポイントを設定します。

この操作を実行するには、次のパラメーターを使用して **Zwfscontrolfile** を呼び出します。

再解析ポイントを設定するには、FSCTL_SET_の代わりに **Flttagfile** を使用して再解析_ポイントを設定する必要があります。

再解析ポイントと FSCTL_設定_再解析_ポイント制御コードの詳細については、Microsoft Windows SDK のドキュメントを参照してください。

Parameters

FileHandle

再解析ポイントを設定するファイルまたはディレクトリのファイルハンドル。このパラメーターは必須であり、NULLにすることはできません。

FsControlCode

操作の制御コード。この操作では、FSCTL_使用して、再解析_ポイント_設定します。

InputBuffer

再解析ポイントデータを格納しているバッファー構造_データ_バッファーまたは **再解析__データ** を格納する、呼び出し元によって割り当てられた **再解析_GUID** を指すポインター。既存の再解析ポイントが変更されている場合は、この構造体の **ReparseTag** メンバーに指定されているタグが、変更する再解析ポイントのタグと一致している必要があります。また、再解析ポイントがサードパーティ (Microsoft 以外) の再解析ポイントである場合、構造体の **ReparseGuid** メンバーに指定されている GUID は再解析_GUID_データ_バッファー構造は再解析ポイントの guid と一致している必要があります変更する。

InputBufferLength

InputBuffer パラメーターによってポイントされるバッファーのサイズ (バイト単位)。再解析_GUID_データ_バッファー構造体の場合、この値には少なくとも、GUID_データ_バッファー_の_のサイズとユーザー定義データのサイズを加えた再解析_GUID を指定する必要があります。、_再解析_データ_バッファー_サイズの最大値以下である必要があります。再解析_データ_バッファー構造の場合、この値は_少なくともデータ_バッファー_のヘッダー_サイズに加え、ユーザー定義データのサイズに加えて、最大値以下である必要があります_再解析_データ_バッファー_サイズ。

OutputBuffer

この操作では使用されません。をNULLに設定します。

OutputBufferLength

この操作では使用されません。を0に設定します。

ステータス ブロック

Zwfscontrolfile は、次のいずれかのような状態_SUCCESS または適切な NTSTATUS 値を返します。

II	II
STATUS_DIRECTORY_NOT_EMPTY	空でないディレクトリに再解析ポイントを設定することはできません。これはエラーコードです。
STATUS_EAS_NOT_SUPPORTED	この要求がトランザクション内にある場合、ファイルに再解析ポイントを設定することはできません。これはエラーコードです。

㊦	㊦
STATUS_IO_REPARSE_DATA_INVALID	指定されたパラメーター値の1つが無効です。これはエラーコードです。
STATUS_IO_REPARSE_TAG_MISMATCH	呼び出し元によって指定された再解析タグが、変更する再解析ポイントのタグと一致しませんでした。これはエラーコードです。
STATUS_NOT_A_REPARSE_POINT	ファイルまたはディレクトリが再解析ポイントではありません。これはエラーコードです。
STATUS_REPARSE_ATTRIBUTE_CONFLICT	再解析ポイントはサードパーティの再解析ポイントであり、呼び出し元によって指定された再解析 GUID が、変更する再解析ポイントの GUID と一致しませんでした。これはエラーコードです。

要件

Header	Ntifs (Ntifs または Fltkernel .h を含む)

関連項目

[IRP_MJ_ファイルの FLT_パラメーター_システム_コントロール](#)

[FltTagFile](#)

[FltUntagFile](#)

[FSCTL__再解析_ポイントを削除します](#)

[FSCTL__再解析_ポイントを取得します](#)

[IRP_MJ_ファイル_システム_コントロール](#)

[IsReparseTagMicrosoft](#)

[IsReparseTagNameSurrogate](#)

[データ_バッファの再解析_](#)

[_GUID_データ_バッファの再解析](#)

[ZwFsControlFile](#)

FSCTL__ゼロ_データ制御コードに設定

2020/04/24 • [Edit Online](#)

FSCTL__ゼロ(0)のファイルの指定された範囲をFILL_データコントロールコードに設定します。ファイルがスパースまたは圧縮されている場合は、NTFS ファイルシステムによってファイル内のディスク領域の割り当てが解除されることがあります。これにより、ファイルサイズを拡張せずに、バイトの範囲がゼロ(0)に設定されます。

ドライバからこの操作を実行するには、次のパラメーターを使用して`Fltfscontrolfile`を呼び出します。

Parameters

Instance

呼び出し元の非透過インスタンスポインター。このパラメーターは必須であり、NULLにすることはできません。

File

0を書き込むファイルへのファイルオブジェクトポインター。このパラメーターは必須であり、NULLにすることはできません。

FsControlCode

操作の制御コード。

この操作では、FSCTL_使用して__ゼロ_データを設定します。

InputBuffer

ファイルへのポインターは、0に設定するファイルの範囲を指定するデータ_情報または`ファイル__ゼロ_データ情報`またはファイルを__ます。

Fileoffsetメンバーは、ゼロ(0)に設定する最初のバイトのバイトオフセットです。また、BeyondFinalZeroメンバーは、最後のゼロ(0)バイトを超える最初のバイトのバイトオフセットです。

ファイルのFlagsメンバー__0_データ_情報_EXは操作の修飾子を指定します。たとえば、フラグがfile_0_data_INFORMATION_フラグに設定され__キャッシュされた_データ_保持する場合、ファイルのこの範囲に対応するキャッシュの内容は削除されません。

InputBufferLength

入力バッファのサイズ(バイト単位)。

OutputBuffer

この操作では使用されません。をNULLに設定します。

OutputBufferLength

この操作では使用されません。を0に設定します。

ステータス ブロック

`Fltfscontrolfile`は、STATUS_SUCCESSまたは適切な NTSTATUS 値を返します。

- 操作を完了するために必要なメモリが不足している場合は、状態 __リソースが返されます。
- ステータス_無効_パラメーターが返されるのは、Inputbufferlengthがファイルのサイズよりも小さい場合__0_データ_情報_構造体、または指定されたファイルがシステムメタデータファイルである場合です。またはディレクトリ。
- ファイル_0_データ_情報_フラグ__キャッシュされている_データがユーザーモードから設定されている場合、__アクセス_拒否された状態が返されます。
- ボリュームが現在書き込み保護されている場合、状態_メディア_書き込み_保護されます。

要件

Header	Ntifs (Ntifs を含む)

関連項目

[FltFsControlFile](#)

[ファイル_ゼロ_データ_情報](#)

[ファイル_ゼロ_データ_情報_EX](#)

FSCTL_SUSPEND_オーバーレイコントロールコード

2020/04/24 • [Edit Online](#)

FSCTL_SUSPEND_オーバーレイコントロールコードは、ボリュームに接続されているバックギグソースを中断し、バックギグソースへのアクセスを禁止して、変更または削除できるようにします。

この操作を実行するには、次のパラメーターを使用して**Fltfscontrolfile**または**zwfscontrolfile**を呼び出します。

```
BOOL
WINAPI
DeviceIoControl( (HANDLE)      hDevice,          // handle to device
                 (DWORD)      FSCTL_SUSPEND_OVERLAY, // dwIoControlCode
                 (LPDWORD)    lpInBuffer,        // input buffer
                 (DWORD)      nInBufferSize,    // size of input buffer
                 (LPDWORD)    lpOutBuffer,       // output buffer
                 (DWORD)      nOutBufferSize,    // size of output buffer
                 (LPDWORD)    lpBytesReturned,   // number of bytes returned
                 (LPOVERLAPPED) lpOverlapped ); // OVERLAPPED structure
```

Parameters

]のインスタンス [

Fltfscontrolfileのみ。呼び出し元の非透過インスタンスポインター。このパラメーターは必須であり、NULL にすることはできません。

]の FileObject [

Fltfscontrolfileのみ。オーバーレイを更新するボリュームのファイルポインターオブジェクト。このパラメーターは必須であり、NULL にすることはできません。

]の FileHandle [

Zwfscontrolfileのみ。オーバーレイが更新されるボリュームのハンドル。このパラメーターは必須であり、NULL にすることはできません。

]の FsControlCode [

操作の制御コード。この操作には、FSCTL_使用して_オーバーレイを中断します。

InputBuffer

入力バッファーへのポインター。これには、**WOF_外部_INFO**構造体が含まれている必要があります。必要に応じて、追加のプロバイダー固有のデータは、**WOF_外部_情報**に含まれます。プロバイダーが WIM ファイルの場合、**wim_プロバイダー_中断_オーバーレイ_入力**構造は、**WOF_外部_情報**に含まれています。

]内の InputBufferLength [

Sizeof(**WOF_外部_INFO**) に、追加のプロバイダー入力データのサイズを加えた値に設定します。

OutputBuffer [out]

使用しません。を NULL に設定します。

]の OutputBufferLength [

を0に設定します。

ステータス ブロック

[Fltfscontrolfile](#)または[zwfscontrolfile](#)は、操作が成功した場合に STATUS_SUCCESS を返します。それ以外の場合、適切な関数は、次のいずれかの NTSTATUS 値を返す可能性があります。

NTSTATUS	説明
STATUS_ACCESS_DENIED	要求元には管理者特権がありません。
STATUS_BUFFER_TOO_SMALL	<i>InputBuffer</i> によってポイントされ、 <i>inputbufferlength</i> によって指定された入力バッファの長さが小さすぎます。
STATUS_INTERNAL_ERROR	要求されたボリュームにアクセスできません。
STATUS_INVALID_DEVICE_REQUEST	バックギングサービスが存在しないか、開始されていません。

注釈

削除するバックギングソースが Windows Imaging Format (WIM) ファイルの場合、入力バッファには、[WOF_外部_INFO](#)構造体の後に[WIM__プロバイダー](#)が含まれます。これにより、オーバーレイ_の入力が中断されます。データ. この場合、*Inputbufferlength*はsizeof(WOF_外部_INFO) + sizeof([WIM__プロバイダー__オーバーレイ_入力](#)) を削除します。Wim_PROVIDER のDataSourceId値_中断_オーバーレイ_入力は、[FSCTL_オーバーレイ](#)要求の追加で以前に追加された wim ファイルに対して行う必要があります。

追加のバックギングプロバイダーは、独自の特定の入力パラメーター構造を定義します。

要件

Header	Ntifs (Ntifs または Fltkernel .h を含む)

関連項目

[FSCTL__オーバーレイの削除](#)

[FSCTL_UPDATE_オーバーレイ](#)

[FSCTL_外部_バックギング_取得する](#)

[外部_バックギング_FSCTL_設定](#)

FSCTL_UPDATE_オーバーレイコントロールコード

2020/04/24 • [Edit Online](#)

FSCTL_UPDATE_オーバーレイコントロールコードは、ボリュームにアタッチされているバックギングソースの新しいデータソース識別子を更新します。

この操作を実行するには、次のパラメーターを使用して **Fltfscontrolfile** または **zwfscontrolfile** を呼び出します。

Parameters

]のインスタンス [

Fltfscontrolfile のみ。呼び出し元の非透過インスタンスポインター。このパラメーターは必須であり、NULL にすることはできません。

]の FileObject [

Fltfscontrolfile のみ。オーバーレイを更新するボリュームのファイルポインターオブジェクト。このパラメーターは必須であり、NULL にすることはできません。

]の FileHandle [

Zwfscontrolfile のみ。オーバーレイが更新されるボリュームのハンドル。このパラメーターは必須であり、NULL にすることはできません。

]の FsControlCode [

操作の制御コード。この操作には、FSCTL_を使用して **_オーバーレイを更新** してください。

InputBuffer

入力バッファーへのポインター。これには、**WOF_外部_INFO** 構造体が含まれている必要があります。必要に応じて、追加のプロバイダー固有のデータは、**WOF_外部_情報** に含まれます。プロバイダーが WIM ファイルの場合、**wim_プロバイダー_更新_オーバーレイ_入力** 構造は、**WOF_外部_情報** に含まれています。

]内の InputBufferLength [

Sizeof(WOF_外部_INFO) に、追加のプロバイダー入力データのサイズを加えた値に設定します。

OutputBuffer [out]

使用しません。を NULL に設定します。

]の OutputBufferLength [

を0に設定します。

ステータス ブロック

Fltfscontrolfile または **zwfscontrolfile** は、操作が成功した場合に STATUS_SUCCESS を返します。それ以外の場合、適切な関数は、次のいずれかの NTSTATUS 値を返す可能性があります。

「	「
STATUS_ACCESS_DENIED	要求元には管理者特権がありません。
STATUS_BUFFER_TOO_SMALL	<i>InputBuffer</i> によってポイントされ、 <i>inputbufferlength</i> によって指定された入力バッファーの長さが小さすぎます。

エラーコード	説明
STATUS_INTERNAL_ERROR	要求されたボリュームにアクセスできません。
STATUS_INVALID_DEVICE_REQUEST	バックギンサービスが存在しないか、開始されていません。

注釈

更新するデータソースが Windows Imaging Format (WIM) ファイルの場合、入力バッファには、[WOF_外部_INFO](#) 構造体の後に [wim_プロバイダー_更新_オーバーレイ_入力](#) が格納されます。データ. この場合、*Inputbufferlength* は `sizeof(WOF_外部_INFO) + sizeof(WIM_PROVIDER_UPDATE_オーバーレイ_入力)` になります。Wim_PROVIDER_UPDATE_オーバーレイの *DataSourceId* 値は、[FSCTL_オーバーレイ](#) 要求の追加で以前に追加された wim ファイルの_入力である必要があります。

その他のバックギンプロバイダーは、独自の特定の入力パラメーター構造を定義します。

要件

バージョン	Windows 8.1 更新プログラムから使用できます。
Header	Ntifs (Ntifs または Fltkernel.h を含む)

関連項目

[FSCTL_オーバーレイの追加](#)

[FSCTL_オーバーレイの削除](#)

[FSCTL_SUSPEND_オーバーレイ](#)

[FSCTL_外部_バックギン_取得する](#)

[外部_バックギン_FSCTL_設定](#)

CcIsThereDirtyLoggedPages ルーチン

2020/04/24 • [Edit Online](#)

CcIsThereDirtyLoggedPages ルーチンは、ボリュームがシステム キャッシュのダーティ ログ データを格納しているすべてのファイルを含むかどうかを決定します。

構文

```
BOOLEAN CcIsThereDirtyLoggedPages(  
    _In_      PDEVICE_OBJECT DeviceObject,  
    _In_opt_ PULONG          NumberOfDirtyPages  
);
```

パラメーター

DeviceObject [in]

確認するボリュームに関連付けられているデバイスのオブジェクトへのポインター。

NumberOfDirtyPages [で、省略可能]

ポインターを省略可能なULONGダーティ ログ ページに関連付けられているボリューム上の数を受け取るバッファー デバイス オブジェクトします。

戻り値

CcIsThereDirtyLoggedPages ルーチンを返します。TRUEボリュームにログ データをキャッシュに変更されたが、まだディスクにフラッシュされていない 1 つまたは複数のキャッシュされたファイルが含まれます。このルーチンを返しますそれ以外の場合、FALSEします。

注釈

このルーチンは返しますTRUEログのダーティ ページが存在しない場合。返しますTRUEログ ページが現在、ボリュームにキューがある場合。

異なり [CcIsThereDirtyDataEx](#)、CcIsThereDirtyLoggedPages ルーチンでは、ファイル システムのデバイス オブジェクトを使用して、ダーティを確認するボリューム キャッシュ情報を検索するにはページにログインします。

必要条件

対象プラットフォーム	ユニバーサル
バージョン	Windows 8 および Windows の以降のバージョンで使用できます。
Header	Ntifs.h (Ntifs.h または FltKernel.h を含む)
Library	NtosKrnl.lib

DLL	NtosKrnل.exe
IRQL	PASSIVE_LEVEL

関連項目

[CcFlushCache](#)

[CcPurgeCacheSection](#)

[CclIsThereDirtyDataEx](#)

CcSetLogHandleForFileEx ルーチン

2020/04/24 • [Edit Online](#)

CcSetLogHandleForFileEx ルーチンは、ファイルと追跡のコールバックをログ ハンドル ファイル ログの機能を設定します。

構文

```
VOID CcSetLogHandleForFileEx(  
    _In_ PFILE_OBJECT      FileObject,  
    _In_ PVOID              LogHandle,  
    _In_ PFLUSH_TO_LSN      FlushToLsnRoutine,  
    _In_ PQUERY_LOG_USAGE    QueryLogUsageRoutine  
);
```

パラメーター

FileObject [で]ログ ハンドルを格納するファイルのファイル オブジェクトへのポインター。

LogHandle [で]が格納されるログのハンドルへのポインター。

FlushToLsnRoutine [で]ログ ファイルへのポインターがこのファイルのバッファーをフラッシュする前に呼び出すコールバック ルーチンをフラッシュします。このルーチンが呼び出されてフラッシュされるすべてのバッファー制御ブロック (BCB) の最新の論理シーケンス番号 (LSN) をログ ファイルがフラッシュされるようにします。このルーチンは、次のように宣言されます。

```
typedef  
VOID (*PFLUSH_TO_LSN) (  
    IN PVOID LogHandle,  
    IN LARGE_INTEGER Lsn  
);
```

LogHandle

このクライアントを識別するために使用される非透過構造へのポインター。

Lsn

このコールバック ルーチンからの戻り値ではディスクにする必要がある LSN です。

QueryLogUsageRoutine [で]このファイルのログの使用率を取得するために呼び出すクライアントコールバック ルーチンへのポインター。このルーチンは、確認のダーティ ページの書き込みを開始するしきい値が満たされたかどうか呼び出されます。このルーチンは、次のように宣言されます。

```
typedef  
VOID (*PQUERY_LOG_USAGE) (  
    IN PVOID LogHandle,  
    OUT PUSHORT PercentageFull  
);
```

LogHandle

このクライアントを識別するために使用される非透過構造へのポインター。

PercentageFull

ログの使用量の割合を示す 0 から 100 までの値。

戻り値

なし

注釈

CcSetLogHandleForFileEx後続の呼び出しで使用するためのファイルのログ ハンドルを設定します [CcGetDirtyPages](#)します。

コールバックを*FlushToLsnRoutine*と*QueryLogUsageRoutine*が必要です。これらの値では、NULL は指定できません。

必要条件

対象プラットフォーム	ユニバーサル
バージョン	Windows 8 以降で利用できます。
Header	Ntifs.h (Ntifs.h を含む)
Library	NtosKrnl.lib
DLL	NtosKrnl.exe
IRQL	任意のレベル

関連項目

[CcGetDirtyPages](#)

[CcSetDirtyPinnedData](#)

CcSetLoggedDataThreshold ルーチン

2020/04/24 • [Edit Online](#)

CcSetLoggedDataThreshold ルーチンは、ログのダーティ ページのスクランが遅延書き込みを開始する場合のしきい値を設定します。

構文

```
VOID CcSetLoggedDataThreshold(  
    _In_ PVOID LogHandle,  
    _In_ ULONG NumberOfPages  
);
```

パラメーター

LogHandle [in]

新しいしきい値のログのハンドル。

NumberOfPages [で]

指定されたログのログのダーティ ページの数をしきい値 *LogHandle* します。

戻り値

なし

注釈

にしきい値 *NumberOfPages* で値が返される場合にのみ使用されます、*PercentageUsed* のパラメーター、*QueryLogUsageRoutine* コールバック ルーチンは 0 です。*QueryLogUsageRoutine* への呼び出しにコールバックが設定されて **CcSetLogHandleForFileEx** します。

CcSetLoggedDataThreshold ルーチンは、ログの完全な割合を使用しない場合、ログのダーティ ページの数の固定値を設定するために使用します。

要件

対象プラットフォーム	ユニバーサル
バージョン	Windows 8 および Windows の以降のバージョンで使用できます。
Header	Ntifs.h (Ntifs.h または FltKernel.h を含む)
Library	NtosKrnl.lib
DLL	NtosKrnl.exe

IRQL	PASSIVE_LEVEL

関連項目

[CcSetLogHandleForFileEx](#)

CcSetReadAheadGranularityEx ルーチン

2020/04/24 • [Edit Online](#)

CcSetReadAheadGranularityEx 日常的な先行読み取りの粒度の設定やパイプライン処理の先読みのキャッシュされたファイルを使用します。

構文

```
VOID CcSetReadAheadGranularityEx(  
    _In_ PFILE_OBJECT FileObject,  
    _In_ PREAD_AHEAD_PARAMETERS ReadAheadParameters  
);
```

パラメーター

FileObject [in]

ファイル オブジェクトへのポインターのキャッシュされたファイルが先行読み取りの粒度を設定するがします。

ReadAheadParameters [in]

先行読み取りのパラメーターを指定します。参照してください[READ_AHEAD_PARAMETERS](#)詳細についてはします。

戻り値

なし

注釈

呼び出すCcSetReadAheadGranularityEx内のファイル オブジェクトの先行読み取り要求をパイプライン処理が有効になります *FileObject*します。適切な値を選択する*PipelinedRequestSize*は分割先行読み取りの要求をより小さな複数の並列要求。呼び出し元CcSetReadAheadGranularityEx先行読み取りのパフォーマンスを調整することで調整できる*PipelinedRequestSize*します。

後 [CcInitializeCacheMap](#) する前に、ファイルをキャッシュするために呼び出されるCcSetReadAheadGranularityExはキャッシュされたファイル、既定の先行読み取りの粒度に対して呼び出されますキャッシュされたファイルはページに等しくなります_サイズ。

要件

対象プラットフォーム	ユニバーサル
バージョン	Windows 8 および Windows の以降のバージョンで使用できます。
Header	Ntifs.h (Ntifs.h を含む)
Library	NtosKrnl.lib

DLL	NtosKrnل.exe

関連項目

[CcInitializeCacheMap](#)

[CcReadAhead](#)

[CcScheduleReadAhead](#)

[CcSetAdditionalCacheAttributes](#)

READ_AHEAD_PARAMETERS 構造体

2020/04/24 • [Edit Online](#)

READ_AHEAD_PARAMETERS 構造体には、一般公開されている読み取り先行パラメーターが含まれています。

構文

```
typedef struct _READ_AHEAD_PARAMETERS {  
  
    CSHORT NodeByteSize;  
    ULONG Granularity;  
    ULONG PipelinedRequestSize;  
    ULONG ReadAheadGrowthPercentage;  
  
} READ_AHEAD_PARAMETERS, *PREAD_AHEAD_PARAMETERS;
```

Members

NodeByteSize [で]

(バイト単位) のノードのサイズ。

粒度 [で]

先行読み取りの粒度です。この値は 2 より大きい、または PAGE_SIZE 等しくの偶数乗である必要があります。

PipelinedRequestSize [で]

要求のサイズ (バイト単位) を実行するときに使用するには、先行読み取りがパイプライン処理します。各先読みをパイプライン処理要求が小さいに分割 *PipelinedRequestSize* 要求のサイズします。これは、1 つの 1 つの主なものではなく複数使いを並列化によって、スループットを向上のために通常使用します。

NOTE

旧バージョンとの互換性のためこの値が 0 の場合すべて先行読み取りの要求は、2 つに分割されます。

ReadAheadGrowthPercentage [in]

これまでに、アプリケーションで準備済みデータの割合として事前読み取りの増加。

注釈

なし

必要条件

対象プラットフォーム	ユニバーサル
バージョン	Windows 8 および Windows の以降のバージョンで使用できます。
Header	Ntifs.h (Ntifs.h を含む)

Library	NtosKrnl.lib
DLL	NtosKrnl.exe

関連項目

[CcSetReadAheadGranularityEx](#)

[CcReadAhead](#)

[CcScheduleReadAhead](#)

CcUnmapFileOffsetFromSystemCache ルーチン

2020/04/24 • [Edit Online](#)

CcUnmapFileOffsetFromSystemCache ルーチン システム キャッシュからキャッシュされたファイルの一部を削除します。

構文

```
VOID CcUnmapFileOffsetFromSystemCache(  
    _In_ PFILE_OBJECT    FileObject,  
    _In_ PLARGE_INTEGER FileOffset,  
    _In_ ULONG           Length,  
    _In_ ULONG           Flags  
);
```

パラメーター

FileObject [in]

キャッシュ メモリがマップされますが、キャッシュされたファイルのファイル オブジェクトへのポインター。

FileOffset [in]

システム キャッシュからのマッピング解除を開始するファイルの場所へのポインター。

長さ [で]

キャッシュからマップ解除するファイルの部分の長さ。場合 長さ 0 の場合、ファイルから始まるメモリ セクションの残りの部分は、*FileOffset*、マップ解除されています。

フラグ [で]

使用されていません。設定 フラグを 0 にします。

戻り値

なし

注釈

両方を設定 *FileOffset* と 長さを 0 にすると、**CcUnmapFileOffsetFromSystemCache** 全体のメモリのセクションをマップ解除するにはキャッシュされたファイル。

CcUnmapFileOffsetFromSystemCache ファイル システム ドライバー ファイルの部分の数が多いがシステムのキャッシュに保持されているときに、キャッシュの使用率を削減することができます。

要件

対象プラットフォーム	ユニバーサル
バージョン	Windows 8 および Windows の以降のバージョンで使用できます。

Header	Ntifs.h (Ntifs.h を含む)
Library	NtosKrn.lib
DLL	NtosKrn.exe

ExDisableResourceBoost

2020/04/24 • [Edit Online](#)

このルーチンは、システムの使用に予約されています。ドライバーは、このルーチンを使わないでください。

FsRtlAllocatePool

2020/04/24 • [Edit Online](#)

Fsrtlallocatepoolルーチンは互換性のために残されていますが、既存のドライババイナリをサポートするためにエクスポートされています。代わりに、[Fsrtlallocatepoolwithtag](#)を使用してください。

FsRtlAllocatePoolWithQuota

2020/04/24 • [Edit Online](#)

Fsrtlallocatepoolwithquotaルーチンは互換性のために残されていますが、既存のドライババイナリをサポートするためにエクスポートされています。代わりに、[Fsrtlallocatepoolwithquotatag](#)を使用してください。

FsRtlEnterFileSystem 関数

2020/08/04 • [Edit Online](#)

Fsrtlenterfilesystem マクロは、通常のカーネルモードの非同期プロシージャ呼び出し (APC) の配信を一時的に無効にします。特別なカーネルモードの Apc は引き続き配信されます。

構文

```
VOID FsRtlEnterFileSystem(  
    VOID  
);
```

パラメーター

None

戻り値

この関数は値を返しません。

注釈

すべてのファイルシステムドライバーのエントリポイントルーチンは、ファイル i/o 要求を実行するために必要なリソースを取得する直前にFsrtlenterfilesystemを呼び出し、その後すぐにFsrtlenterfilesystemを呼び出す必要があります。これにより、ルーチンを実行中に中断したり、他のファイル i/o 要求をブロックしたりすることはできません。

Fsrtlenterfilesystemの呼び出しが成功するたびに、後続のFsrtlenterfilesystemの呼び出しで一致する必要があります。

ファイルシステムフィルタードライバーは、FsrtlenterfilesystemまたはKeLeaveCriticalRegionが同じディスパッチルーチン内にある場合にのみ、FsrtlenterfilesystemまたはIoCallDriverの前にKeEnterCriticalRegionを呼び出すことによって、通常のカーネル apc の配信を無効にすることができます。IoCallDriverの前にFsrtlenterfilesystemまたはKEENTERCRITICALREGIONを呼び出して、IRP の完了ルーチンでfsrtlenterfilesystemまたはKeLeaveCriticalRegionを呼び出すことはできません。Driver Verifier には、この状況を検出するための規則があります。

ファイルシステムフィルタードライバーは、リソースを取得する前に通常のカーネル Apc を無効にする必要があります。ファイルシステムフィルタードライバーは、次のルーチンを使用してリソースを取得します。

- [ExAcquireResourceExclusive](#)
- [ExAcquireResourceExclusiveLite](#)
- [ExAcquireResourceShared](#)
- [ExAcquireResourceSharedLite](#)
- [ExAcquireSharedStarveExclusive](#)
- [ExAcquireSharedWaitForExclusive](#)

[FltAcquireResourceExclusive](#)、[FltAcquireResourceShared](#)、および[FltReleaseResource](#)ルーチンは、fsrtlenterfilesystemの代わりに、リソースを取得して解放するときに、apc を正しく処理するために使用できます。

要件

ターゲットプラットフォーム: デスクトップ

ヘッダー: Ntifs (Ntifs を含む)

IRQL: <= APC_LEVEL

関連項目

[ExAcquireResourceExclusive](#)

[ExAcquireResourceExclusiveLite](#)

[ExAcquireResourceShared](#)

[ExAcquireResourceSharedLite](#)

[ExAcquireSharedWaitForExclusive](#)

[ExAcquireSharedStarveExclusive](#)

[ExReleaseResource](#)

[ExReleaseResourceLite](#)

[ExTryToAcquireFastMutex](#)

[FltAcquireResourceExclusive](#)

[FltAcquireResourceShared](#)

[FltReleaseResource](#)

[FsRtlExitFileSystem](#)

[IoCallDriver](#)

[KeEnterCriticalRegion](#)

[KeRaiseIrqlToDpcLevel](#)

FsRtlExitFileSystem 関数

2020/04/24 • [Edit Online](#)

Fsrtlexitfilesystem マクロは、[Fsrtlexitfilesystem](#) の前の呼び出しによって無効にされた通常のカーネルモード apc の配信を再度有効にします。

構文

```
VOID FsRtlExitFileSystem(  
    VOID  
);
```

パラメーター

なし

戻り値

この関数は値を返しません。

注釈

すべてのファイルシステムドライバーのエントリーポイントルーチンは、ファイル i/o 要求を実行するために必要なリソースを取得する直前に[Fsrtlenterfilesystem](#)を呼び出し、その後すぐにFsrtlenterfilesystemを呼び出す必要があります。これにより、ルーチンを実行中に中断したり、他のファイル i/o 要求をブロックしたりすることはできません。

[Fsrtlenterfilesystem](#) の呼び出しが成功するたびに、後続のFsrtlenterfilesystemの呼び出しで一致する必要があります。

ローカルファイルシステムやネットワークリダイレクターとは異なり、ファイルシステムフィルタードライバーは通常のカーネル Apc の配信を無効にしないようにする必要があります。これは、[Fsrtlenterfilesystem](#) または [KeEnterCriticalRegion](#) を呼び出すか、または IRQL APC の _レベルにすることによって行います。) を呼び出してIoCallDriverします。

ファイルシステムフィルタードライバーが通常のカーネル Apc を無効にする必要があるのは、[ExAcquireResourceExclusive](#)、[ExAcquireResourceExclusiveLite](#)、[ExAcquireResourceShared](#)を呼び出す直前だけです。

[ExAcquireResourceSharedLite](#) または [ExAcquireSharedStarveExclusive](#)。 [ExReleaseResource](#) または [ExReleaseResourceLite](#) を呼び出した後、フィルタードライバーは通常のカーネル apc の配信を直ちに有効にする必要があります。 [FltAcquireResourceExclusive](#)、[FltAcquireResourceShared](#)、および [FltReleaseResource](#) ルーチンは、[fsrtlenterfilesystem](#) の代わりに使用できます。このルーチンでは、とを取得するときに、apc を正しく処理します。リソースを解放しています。

[ExAcquireSharedWaitForExclusive](#) を呼び出す前に通常のカーネル apc を無効にする必要はありません。このルーチンは [KeRaiseIrqlToDpcLevel](#) を呼び出します。これにより、通常のカーネル apc と特殊なカーネル apc の両方が無効になります。また、[ExAcquireFastMutex](#) または [ExAcquireResourceExclusive](#) を呼び出す前にこれを行う必要はありません。これらのルーチンは通常のカーネル apc を無効にするためです。

要件

対象プラットフォーム	Desktop
Header	Ntifs (Ntifs を含む)
IRQL	<= APC_LEVEL

関連項目

- [ExAcquireFastMutex](#)
- [ExAcquireResourceExclusive](#)
- [ExAcquireResourceExclusiveLite](#)
- [ExAcquireResourceShared](#)
- [ExAcquireResourceSharedLite](#)
- [ExAcquireSharedWaitForExclusive](#)
- [ExAcquireSharedStarveExclusive](#)
- [ExReleaseResource](#)
- [ExReleaseResourceLite](#)
- [ExTryToAcquireFastMutex](#)
- [FltAcquireResourceExclusive](#)
- [FltAcquireResourceShared](#)
- [FltReleaseResource](#)
- [FsRtlEnterFileSystem](#)
- [IoCallDriver](#)
- [KeLeaveCriticalRegion](#)
- [KeRaiseIrqlToDpcLevel](#)

FltAcquireResourceExclusive ルーチン

2020/08/04 • [Edit Online](#)

FltAcquireResourceExclusiveルーチンは、呼び出し元のスレッドによって排他的にアクセスするために、指定されたリソースを取得します。

構文

```
VOID FltAcquireResourceExclusive(  
    _Inout_ PERESOURCE Resource  
);
```

パラメーター

リソース [in, out]

不透明な構造体を指すポインターです。この構造体は、呼び出し元によって非ページブルから割り当てられ、[Exinitializer eresourcelite](#)または[Exreinitializer eresourcelite](#)を呼び出すことによって初期化される必要があります。

戻り値

なし

解説

このルーチンは、Windows XP Service Pack 2 (SP2)、Windows Server 2003 Service Pack 1 (SP1)、およびそれ以降のバージョンの Windows で使用できます。

FltAcquireResourceExclusiveは、呼び出し元のスレッドによって排他的にアクセスするために、指定されたリソースを取得します。

次の状況では、指定されたリソースへの排他アクセスが呼び出し元に付与されているかどうかを判断します。

- リソースが現在所有されていない場合は、現在のスレッドに対して排他アクセスが直ちに許可されます。
- 呼び出し元が既に排他アクセス用にリソースを取得している場合、現在のスレッドは同じ種類のアクセスを再帰的に許可されます。
- リソースへの共有アクセス権を持つ呼び出し元は、ロックを解放してから排他的に再取得する必要があります。
- リソースが別のスレッドによって排他的に所有されている場合、または呼び出し元がリソースへの共有アクセスだけを持っている場合は、リソースを取得できるようになるまで、現在のスレッドは待機状態になります。

NOTE

2つのスレッドが同じリソースに対して共有ロックを保持し、共有ロックを解放せずにロックを排他的に取得しようとすると、デッドロックが発生します。つまり、各スレッドは、他のスレッドがロックに対して共有ホールドを解放するまで待機し、それ以外の場合は共有ホールドを解放しません。

FltAcquireResourceExclusiveは、通常のカーネル APC 配信を無効にする[ExAcquireResourceExclusiveLite](#)のラッパーです。

FltAcquireResourceExclusiveは通常のカーネル APC 配信を無効にするため、FltAcquireResourceExclusiveを呼び出す前に[KeEnterCriticalRegion](#)または[fsrtlenterfilesystem](#)を呼び出す必要はありません。

リソースを取得した後に解放するには、[FltReleaseResource](#)を呼び出します。FltAcquireResourceExclusiveの呼び出しが成功するたびに、FltReleaseResourceへの後続の呼び出しで一致する必要があります。

共有アクセス用のリソースを取得するには、[FltAcquireResourceShared](#)を呼び出します。

システムのリソースリストからリソースを削除するには、[ExDeleteResourceLite](#)を呼び出します。

再利用するためにリソースを初期化するには、[Exreinitializer Eresourcelite](#)を呼び出します。

÷の構造体の詳細については、「カーネルアーキテクチャの設計ガイド」の「[のについて](#)」を[参照](#)してください。

必要条件

ターゲット プラットフォーム	ユニバーサル
バージョン	Windows XP SP2、Windows Server 2003 SP1、およびそれ以降のバージョンのすべての Windows オペレーティングシステムで使用できます。
Header	Fltkernel.h (Fltkernel.h を含む)
ライブラリ	FltMgr.lib
IRQL	<= APC_LEVEL

関連項目

- [ExAcquireResourceExclusiveLite](#)
- [ExDeleteResourceLite](#)
- [Ex/Eresourcelite](#)
- [Exreinitializer Eresourcelite](#)
- [FltAcquireResourceShared](#)
- [FltReleaseResource](#)
- [FsRtlEnterFileSystem](#)
- [KeEnterCriticalRegion](#)

FltAcquireResourceShared ルーチン

2020/04/24 • [Edit Online](#)

FltAcquireResourceSharedルーチンは、呼び出し元のスレッドによって共有アクセス用に指定されたリソースを取得します。

構文

```
VOID FltAcquireResourceShared(  
    _Inout_ PERESOURCE Resource  
);
```

パラメーター

リソース[in, out]

不透明な構造体へのポインター。この構造体は、呼び出し元によって非ページプールから割り当てられ、[Exinitializer eresourcelite](#)または[Exreinitializer eresourcelite](#)を呼び出すことによって初期化される必要があります。

戻り値

なし

注釈

このルーチンは、Microsoft Windows XP SP2、Microsoft Windows Server 2003 SP1 以降で使用できます。

FltAcquireResourceSharedルーチンは、呼び出し元のスレッドによって共有アクセス用に指定されたリソースを取得します。

指定したリソースへの共有アクセスが呼び出し元に付与されているかどうかは、次のことに依存します。

- リソースが現在所有されていない場合は、現在のスレッドに対して共有アクセスが直ちに付与されます。
- 呼び出し元が既にリソースを取得している場合（共有または排他アクセスの場合）、現在のスレッドは同じ種類のアクセスを再帰的に許可されます。この呼び出しを行っても、特定のリソースの呼び出し元の排他所有権が共有に変換されることはないことに注意してください。
- リソースが別のスレッドによって共有されているものの、リソースへの排他アクセスを待機しているスレッドがない場合、共有アクセスはすぐに呼び出し元に付与されます。排他的な待機が発生した場合、呼び出し元は待機状態になります。
- リソースが別のスレッドによって排他的に所有されている場合、または排他的アクセスを待機している別のスレッドが存在し、呼び出し元がリソースへの共有アクセスをまだ持っていない場合は、リソースが取得されるまで、現在のスレッドは待機状態になります。

FltAcquireResourceSharedは、通常のカーネル APC 配信を無効にする[ExAcquireResourceSharedLite](#)のラッパーです。

FltAcquireResourceSharedは通常のカーネル APC 配信を無効にするため、FltAcquireResourceSharedを呼び出す前に[KeEnterCriticalRegion](#)または[fsrtlenterfilesystem](#)を呼び出す必要はありません。

リソースを取得した後に解放するには、[FltReleaseResource](#)を呼び出します。FltAcquireResourceSharedの呼び出

しが成功するたびに、`FltReleaseResource`への後続の呼び出しで一致する必要があります。

排他アクセスのためにリソースを取得するには、`FltAcquireResourceExclusive`を呼び出します。

システムのリソースリストからリソースを削除するには、`ExDeleteResourceLite`を呼び出します。

再利用するためにリソースを初期化するには、`Exreinitializer Eresourcelite`を呼び出します。

÷の構造体の詳細については、「カーネルアーキテクチャの設計ガイド」の「のについて」を参照してください。

要件

対象プラットフォーム	ユニバーサル
Header	Fltkernel.h (Fltkernel.hを含む)
Library	fltMgr.lib
IRQL	<= APC_LEVEL

関連項目

[ExAcquireResourceSharedLite](#)

[ExDeleteResourceLite](#)

[Ex/Eresourcelite](#)

[Exreinitializer Eresourcelite](#)

[FltAcquireResourceExclusive](#)

[FltReleaseResource](#)

[FsRtlEnterFileSystem](#)

[KeEnterCriticalRegion](#)

FltReleaseResource ルーチン

2020/04/24 • [Edit Online](#)

FltReleaseResource ルーチンは、現在のスレッドが所有する指定されたリソースを解放します。

構文

```
VOID FltReleaseResource(  
    _Inout_ PERESOURCE Resource  
);
```

パラメーター

リソース[in, out]

解放されるリソースの不透明なリソース構造体へのポインター。

戻り値

なし

注釈

FltReleaseResource は、[FltAcquireResourceExclusive](#) または [FltAcquireResourceShared](#) を呼び出すことによって既に取得されたリソースを解放します。

FltReleaseResource は、通常のカネル APC 配信を再び実現する [ExReleaseResourceLite](#) のラッパーです。

FltReleaseResource は通常のカネル APC 配信を再び行うため、FltReleaseResource を呼び出した後に [KeLeaveCriticalRegion](#) または [fsrtlEXITFILESYS](#) を呼び出す必要はありません。

排他アクセスのためにリソースを取得するには、[FltAcquireResourceExclusive](#) を呼び出します。

共有アクセス用のリソースを取得するには、[FltAcquireResourceShared](#) を呼び出します。

システムのリソースリストからリソースを削除するには、[ExDeleteResourceLite](#) を呼び出します。

再利用するためにリソースを初期化するには、[Exreinitializer Eresourcelite](#) を呼び出します。

÷ の構造体の詳細については、「カネルアーキテクチャの設計ガイド」の「のについて」を[参照](#)してください。

要件

対象プラットフォーム	ユニバーサル
バージョン	このルーチンは、Microsoft Windows XP SP2、Microsoft Windows Server 2003 SP1 以降で使用できます。
Header	Fltkernel.h (Fltkernel.h を含む)

Library	fltMgr.lib
DLL	Fltmgr.sys
IRQL	<= DISPATCH_LEVEL

関連項目

- [ExDeleteResourceLite](#)
- [Ex/Eresourcelite](#)
- [Exreinitializer Eresourcelite](#)
- [ExReleaseResourceLite](#)
- [FltAcquireResourceExclusive](#)
- [FltAcquireResourceShared](#)
- [FsRtlExitFileSystem](#)
- [KeLeaveCriticalRegion](#)

KeAcquireQueuedSpinLock

2020/04/24 • [Edit Online](#)

KeAcquireQueuedSpinLockルーチンはシステム用に予約されています。

MRxCollapseOpen ルーチン

2020/04/24 • [Edit Online](#)

MRxCollapseOpen ルーチンは [RDBSS](#) によって呼び出され、ネットワークミニリダイレクターが開いているファイルシステム要求を既存の SRV_オープン構造に折りたたんでいることを要求します。

構文

```
PMRX_CALLDOWN MRxCollapseOpen;  
  
NTSTATUS MRxCollapseOpen(  
    _Inout_ PRX_CONTEXT RxContext  
)  
{ ... }
```

parameters

RxContext [in, out]

RX_コンテキスト構造体へのポインター。このパラメーターには、操作を要求している IRP が含まれています。

戻り値

MRxCollapseOpen は正常に完了した状態_成功したか、次のような適切な NTSTATUS 値を返します。

NTSTATUS	説明
STATUS_INSUFFICIENT_RESOURCES	操作を完了するためのリソースが不足しています。

解説

MRxCollapseOpen は、SRV_開いている構造体をローカルに折りたたむために RDBSS によって呼び出されます。ネットワークミニリダイレクターは、折りたたみが可能かどうかを判断するために調査されるため、ネットワークミニリダイレクターを2回呼び出す理由はありません。ネットワークミニリダイレクターが SRV_開いている構造を折りたたむ場合は、その操作を行い、返却可能の状態を返します。STATUS_SUCCESS の戻り値は、終了する戻り値です。異なる戻り値 (たとえば、STATUS_MORE_PROCESSING_REQUIRED) は、終了しない戻り値と見なされます。

MRxCollapseOpen を呼び出す前に、RDBSS は、*RxContext* パラメーターによって示される RX_コンテキスト構造内の次のメンバーを変更します。

pRelevantSrvOpen は、折りたたまれるように SRV_オープン構造に設定されています。

Create. pSrvCall は、開いている srv_に関連付けられている SRV_呼び出し構造に設定されます。

ネットワークミニリダイレクターが SRV_開いている構造体を折りたたむ場合は、RX_コンテキスト構造の Srvopen メンバーを、折りたたまれた SRV_open 構造体に設定する必要があります。

前提条件

ターゲットプラットフォーム	デスクトップ
ヘッダー	Mrx.h (Mrx を含む)

関連項目

[MRxAreFilesAliased](#)

[MRxCleanupFobx](#)

[MRxCloseSrvOpen](#)

[MRxCreate](#)

[MRxDeallocateForFcb](#)

[MRxDeallocateForFobx](#)

[MRxExtendForCache](#)

[MRxExtendForNonCache](#)

[MRxFlush](#)

[MRxForceClosed](#)

[MRxIsLockRealizable](#)

[MRxShouldTryToCollapseThisOpen](#)

[MRxTruncate](#)

[MRxZeroExtend](#)

MRxCreate ルーチン

2020/04/24 • [Edit Online](#)

MRxCreate ルーチンは、ネットワークミニリダイレクターがファイルシステムオブジェクトを作成するように要求するために、*RDBSS* によって呼び出されます。

構文

```
PMRX_CALLDOWN MRxCreate;  
  
NTSTATUS MRxCreate(  
    _Inout_ PRX_CONTEXT RxContext  
)  
{ ... }
```

parameters

RxContext [in, out]

RX_コンテキスト構造体へのポインター。このパラメーターには、操作を要求している IRP が含まれています。

戻り値

MRxCreate は正常に完了した状態_成功したか、または次のいずれかのような NTSTATUS 値を返します。

NTSTATUS	説明
STATUS_INSUFFICIENT_RESOURCES	操作を完了するためのリソースが不足しています。
STATUS_NETWORK_ACCESS_DENIED	ネットワークアクセスが拒否されました。このエラーは、ネットワークミニリダイレクターが読み取り専用の共有で新しいファイルを開くように要求された場合に返されます。
STATUS_NOT_IMPLEMENTED	リモートブートやリモートページファイルなど、要求された機能は実装されていません。
STATUS_NOT_SUPPORTED	拡張属性など、要求された機能はサポートされていません。
STATUS_OBJECT_NAME_COLLISION	ネットワークミニリダイレクターは、既に存在するファイルを作成するように求められました。
STATUS_OBJECT_NAME_NOT_FOUND	オブジェクト名が見つかりませんでした。このエラーは、ネットワークミニリダイレクターが、存在しないファイルを開くように要求された場合に返されます。

エラーコード	説明
STATUS_OBJECT_PATH_NOT_FOUND	オブジェクトのパスが見つかりませんでした。このエラーは、NTFS ストリームオブジェクトが要求され、リモートファイルシステムでストリームがサポートされていない場合に返されます。
STATUS_REPARSE	シンボリックリンクを処理するには、再解析が必要です。
STATUS_RETRY	操作を再試行する必要があります。ネットワークミニリダイレクターで共有違反またはアクセス拒否エラーが発生した場合、このエラーが返されることがあります。

解説

*MRxCreate*は、ネットワーク上でファイルシステムオブジェクトを開くようにネットワークミニリダイレクターに要求するために、RDBSS によって呼び出されます。この呼び出しは、[IRP_MJ_CREATE](#) 要求の受信に응答して、RDBSS によって発行されます。

*MRxCreate*を呼び出す前に、RDBSS は、*RxContext*パラメーターによって示される RX_コンテキスト構造内の次のメンバーを変更します。

*pRelevantSrvOpen*は、オープン構造の *SRV_*に設定されます。

*Create. pSrvCall*は *SRV_*呼び出し構造に設定されます。

作成します。 *NtCreateParameters*は、要求された NT_作成_パラメーターに設定されます。

ネットワークミニリダイレクターのコンテキストでは、ファイルオブジェクトは、関連付けられたファイルコントロールブロック (FCB) とファイルオブジェクト拡張 (FOBx) 構造体を参照します。ファイルオブジェクトと FOBxs の間には1対1の対応があります。多くのファイルオブジェクトは、リモートサーバー上の1つのファイルを表す同じ FCB を参照します。クライアントは、同じ FCB で複数の異なる open requests (*NtCreateFile* requests) を持つことができ、これらはそれぞれ新しいファイルオブジェクトを作成します。RDBSS とネットワークミニリダイレクターは、受信した *NtCreateFile* 要求よりも*MRxCreate*要求を送信することを選択できます。これは、*SRV_*オープン構造体を複数の FOBxs 間で共有する場合に有効です。

*MRxCreate*要求がファイルの上書き用であり、*MRxCreate*が状態_SUCCESS を返した場合、RDBSS はページング i/o リソースを取得し、ファイルを切り詰めます。ファイルがキャッシュマネージャーによってキャッシュされている場合、RDBSS は、サーバーから受け取ったものと共に、キャッシュマネージャーが持つサイズを更新します。

MRxCreate を返す前に、*RxContext*パラメーターによって示される RX_コンテキスト構造のメンバーを > 設定する必要があります。

前提条件

ターゲットプラットフォーム	デスクトップ
ヘッダー	<i>Mrx.h</i> (<i>Mrx</i> を含む)

関連項目

[MRxAreFilesAliased](#)

[MRxCleanupFobx](#)

MRxCloseSrvOpen

MRxCollapseOpen

MRxCreate

MRxDeallocateForFcb

MRxDeallocateForFobx

MRxExtendForCache

MRxExtendForNonCache

MRxFlush

MRxForceClosed

MRxIsLockRealizable

MRxShouldTryToCollapseThisOpen

MRxTruncate

MRxZeroExtend

MRxDevFcbXXXControlFile ルーチン

2020/04/24 • [Edit Online](#)

MRxDevFcbXXXControlFile ルーチンは、デバイス FCB コントロール要求 (IOCTL または FSCTL 要求) をネットワークミニリダイレクターに渡すために、[RDBSS](#)によって呼び出されます。

構文

```
PMRX_CALLDOWN MRxDevFcbXXXControlFile;  
  
NTSTATUS MRxDevFcbXXXControlFile(  
    _Inout_ PRX_CONTEXT RxContext  
)  
{ ... }
```

parameters

RxContext [in, out]

RX_コンテキスト構造体へのポインター。このパラメーターには、操作を要求している IRP が含まれています。

戻り値

MRxDevFcbXXXControlFile は正常に完了した状態_成功したか、または次のいずれかのような NTSTATUS 値を返します。

NTSTATUS	説明
STATUS_ACCESS_DENIED	ネットワークミニリダイレクターを停止または開始するための要求が行われましたが、呼び出し元にはこの操作に対する適切なセキュリティがありませんでした。
STATUS_INVALID_DEVICE_REQUEST	無効なデバイス要求がネットワークミニリダイレクターに送信されました。
STATUS_REDIRECTOR_HAS_OPEN_HANDLES	これはネットワークミニリダイレクターを停止する要求でしたが、リダイレクターには、この時点で停止できないハンドルが開かれています。
STATUS_REDIRECTOR_NOT_STARTED	これはネットワークミニリダイレクターを停止する要求でしたが、リダイレクターが開始されていません。
STATUS_REDIRECTOR_STARTED	これはネットワークミニリダイレクターを開始する要求でしたが、リダイレクターは既に開始されています。

解説

MRxDevFcbXXXControlFile は、ネットワークミニリダイレクターに送信されるデバイス FCB に関連する IOCTL および FSCTL 要求を処理します。

*MRxDevFcbXXXControlFile*を呼び出す前に、RDBSS は、*RxContext*パラメーターによって示される RX_コンテキスト構造内の次のメンバーを変更します。

MajorFunctionは、IRP の主要な関数に設定されています。

これが IRP_MJ_FILE_SYSTEM_CONTROL 要求であった場合、*RxContext*パラメーターによって示される RX_CONTEXT 構造体の次のメンバーが RDBSS によって変更されます。

FsCtlは、FsCtl コードのマイナー関数コードに設定されていることを示します。

FsCtlが、IRP の FsCtl コードに設定されていることを示します。

これが IRP_MJ_デバイス_CONTROL または IRP_MJ_内部_デバイス_コントロール要求であった場合、RDBSS は RxContext が指す RX_CONTEXT 構造体の次のメンバーを変更します。パラメーター:

FsCtlは、IRP の制御コードに設定されていることを示します。

*MRxDevFcbXXXControlFile*が STATUS_SUCCESS を返した場合、ルーチンは成功しました。その他の戻り値は、エラーが発生したことを示します。

前提条件

ターゲットプラットフォーム	デスクトップ
ヘッダー	Mrx.h (Mrx を含む)

関連項目

[MRxStart](#)

[MRxStop](#)

MRxExtendForNonCache ルーチン

2020/04/24 • [Edit Online](#)

MRxExtendForNonCache ルーチンは、ファイルがキャッシュマネージャーによってキャッシュされていない場合に、ネットワークミニリダイレクターがファイルを拡張するよう要求するために、[RDBSS](#)によって呼び出されます。

構文

```
PMRX_EXTENDFILE_CALLDOWN MRxExtendForNonCache;  
  
ULONG MRxExtendForNonCache(  
    _Inout_ PRX_CONTEXT    RxContext,  
    _Inout_ PLARGE_INTEGER pNewFileSize,  
    _Out_    PLARGE_INTEGER pNewAllocationSize  
)  
{ ... }
```

パラメーター

RxContext [in, out]

RX_コンテキスト構造体へのポインター。このパラメーターには、操作を要求している IRP が含まれています。

Pnewfilesize サイズの [in, out]

新しいファイルサイズのバイト数を示す大きな_整数値へのポインター。

*Pnew*のサイズ[out]

[MRxExtendForCache](#)が返されたときに、新しい割り当てサイズを格納するための大きな_整数へのポインター。

戻り値

*MRxExtendForNonCache*は、成功した場合は状態_成功した場合は、失敗した場合はエラーコードを返します。

注釈

*MRxExtendForNonCache*は、キャッシュされていない i/o 用にファイルを拡張するネットワーク要求を処理します。

*MRxExtendForNonCache*を呼び出す前に、RDBSS は、*RxContext*パラメーターによって示される RX_コンテキスト構造内の次のメンバーを変更します。

Lowiocontext. 操作は LOWIO_OP_WRITE に設定されています

Lowiocontext . READWRITEFLAG には、_の_のビットセットを拡張する LOWIO_があります

ファイルまたはディレクトリの情報をキャッシュするネットワークミニリダイレクターは、ファイルが拡張されるときに、キャッシュ情報を無効にすることが必要になる場合があります。

要件

対象プラットフォーム	Desktop

Header	Mrx.h (Mrx を含む)

関連項目

[MRxAreFilesAliased](#)

[MRxCleanupFobx](#)

[MRxCloseSrvOpen](#)

[MRxCollapseOpen](#)

[MRxCreate](#)

[MRxDeallocateForFcb](#)

[MRxDeallocateForFobx](#)

[MRxExtendForCache](#)

[MRxFlush](#)

[MRxForceClosed](#)

[MRxIsLockRealizable](#)

[MRxShouldTryToCollapseThisOpen](#)

[MRxTruncate](#)

[MRxZeroExtend](#)

MRxFlush ルーチン

2020/04/24 • [Edit Online](#)

MRxFlush ルーチンは、ネットワークミニリダイレクターがファイルシステムオブジェクトの内容をストレージに書き込むように要求するために、[RDBSS](#)によって呼び出されます。RDBSS は、[IRP_MJ_FLUSH_BUFFERS](#) 要求の受信に応答して、この呼び出しを発行します。

構文

```
PMRX_CALLDOWN MRxFlush;  
  
NTSTATUS MRxFlush(  
    _Inout_ PRX_CONTEXT RxContext  
)  
{ ... }
```

パラメーター

RxContext [in, out]

RX_コンテキスト構造体へのポインター。このパラメーターには、操作を要求している IRP が含まれています。

戻り値

MRxFlush は正常に完了した状態_成功したか、次のような適切な NTSTATUS 値を返します。

NTSTATUS	説明
STATUS_INSUFFICIENT_RESOURCES	操作を完了するためのリソースが不足しています。

注釈

MRxFlush は、ファイルフラッシュのネットワーク要求を処理します。

要件

対象プラットフォーム	Desktop
Header	Mrx.h (Mrx を含む)

関連項目

[MRxAreFilesAliased](#)

[MRxCleanupFobx](#)

[MRxCloseSrvOpen](#)

[MRxCollapseOpen](#)

MRxCreate

MRxDeallocateForFcb

MRxDeallocateForFobx

MRxExtendForCache

MRxExtendForNonCache

MRxFlush

MRxForceClosed

MRxIsLockRealizable

MRxShouldTryToCollapseThisOpen

MRxTruncate

MRxZeroExtend

MRxLowIOSubmit[LOWIO_OP_EXCLUSIVELOCK]

ルーチン

2020/04/24 • [Edit Online](#)

MRxLowIOSubmit[LOWIO_OP_EXCLUSIVELOCK] ルーチンは、ネットワークミニリダイレクターがファイルオブジェクトに対して排他ロックを開くように要求するためにRDBSSによって呼び出されます。

構文

```
PMRX_CALLDOWN MRxLowIOSubmit[LOWIO_OP_EXCLUSIVELOCK];

NTSTATUS MRxLowIOSubmit[LOWIO_OP_EXCLUSIVELOCK](
    _Inout_ PRX_CONTEXT RxContext
)
{ ... }
```

パラメーター

RxContext [in, out]

RX_コンテキスト構造体へのポインター。このパラメーターには、操作を要求している IRP が含まれています。

戻り値

MRxLowIOSubmit[LOWIO_OP_EXCLUSIVELOCK] は正常に完了したか、次のいずれかの適切な NTSTATUS 値を返します。

NTSTATUS	説明
STATUS_CONNECTION_DISCONNECTED	接続が切断されました。
STATUS_INSUFFICIENT_RESOURCES	要求を完了するためのリソースが不足しています。
STATUS_INVALID_NETWORK_RESPONSE	リモートサーバーから無効な応答を受信しました。
STATUS_INVALID_PARAMETER	<i>RxContext</i> で無効なパラメーターが指定されました。
STATUS_LINK_FAILED	リモートサーバーに再接続して要求を完了できませんでした。
STATUS_NOT_IMPLEMENTED	このルーチンは実装されていません。
STATUS_SHARING_VIOLATION	共有違反が発生しました。
STATUS_UNSUCCESSFUL	呼び出しに失敗しました。

注釈

RDBSS は、*MRxLowIOSubmit[LOWIO_OP_EXCLUSIVELOCK]* を呼び出します。これに応じて、IRP_ [MJ_ロック_制御](#) 要求が発生した場合に **応答** します。IrpSp->フラグには、SL_排他_ロックビットが設定されています。

MRxLowIOSubmit[LOWIO_OP_EXCLUSIVELOCK] を呼び出す前に、RDBSS は、*RxContext* パラメーターが指す RX_CONTEXT 構造体の次のメンバーを変更します。

Lowiocontext. Operationメンバーは lowio_OP_EXCLUSIVELOCK に設定されています。

Lowiocontext threadidメンバーは、RDBSS で操作を開始したプロセスのスレッドに設定されます。

QuadPart の値には、Lowiocontext ... byteoffset メンバーは、irpsp-> **パラメーター** の値に設定されます。

Lowiocontext. Paramsfor Locks. キーメンバーは、irpsp->Parameters. lockcontrol. keyの値に設定されます。

Lowiocontext. Locks. flagsメンバーは、irpsp-> **フラグ** の値に設定されます。

QuadPart**メンバー**には、irpsp->パラメーターの値が設定されています。Lockcontrol. length. 。

RX_コンテキスト構造のLowiocontext **操作** メンバーは、実行する低 i/o 操作を指定します。低 i/o ルーチンのいくつかは、ネットワークミニリダイレクターで同じルーチンを指すことができます。これは、このLowiocontext **操作** のメンバーが、要求された低 i/o 操作を区別するために使用できるためです。たとえば、ファイルロックに関連するすべての i/o 呼び出しで、ネットワークミニリダイレクターで同じ低 i/o ルーチンを呼び出すことができ、そのルーチンはLowiocontext. operationメンバーを使用して、要求されたロック操作とロック解除操作を区別できます。

MRxLowIOSubmit[LOWIO_OP_EXCLUSIVELOCK] ルーチンが完了するまでに時間がかかる場合は、ネットワーク通信を開始する前に、ネットワークミニリダイレクタードライバーが FCB 構造体を解放する必要があります。FCB 構造体は、[RxReleaseFcbResourceForThreadInMRx](#) を呼び出すことによって解放できます。
MRxLowIOSubmit[LOWIO_OP_EXCLUSIVELOCK] ルーチンが処理されている間、RX_コンテキストのlowiocontextスレッドメンバーは、プロセスのスレッドを示すことが保証されます。RDBSS で操作を開始しました。

RX_コンテキストのLowiocontext threadidメンバーを使用すると、別のスレッドに代わって FCB 構造体を解放できます。非同期ルーチンが完了すると、初期スレッドから取得された FCB 構造体が解放されます。

要件

対象プラットフォーム	Desktop
Header	Mrx.h (Mrx を含む)

関連項目

- [MRxLowIOSubmit\[LOWIO_OP_FSCTL\]](#)
- [MRxLowIOSubmit\[LOWIO_OP_IOCTL\]](#)
- [MRxLowIOSubmit\[LOWIO_OP___ディレクトリの変更を通知\]](#)
- [MRxLowIOSubmit\[LOWIO_OP_読み取り\]](#)
- [MRxLowIOSubmit\[LOWIO_OP_SHAREDLOCK\]](#)
- [MRxLowIOSubmit\[LOWIO_OP_ロック解除\]](#)
- [MRxLowIOSubmit\[LOWIO_OP_ロック解除_複数\]](#)
- [MRxLowIOSubmit\[LOWIO_OP_書き込み\]](#)

MRxLowIOSubmit[LOWIO_OP_FSCTL]ルーチン

2020/04/24 • [Edit Online](#)

MRxLowIOSubmit[LOWIO_OP_FSCTL] ルーチン呼び出してRDBSSネットワーク ミニリダイレクターの問題がファイル システムであることを要求するにはリモート ファイルで要求を制御します。

構文

```
PMRX_CALLDOWN MRxLowIOSubmit[LOWIO_OP_FSCTL];

NTSTATUS MRxLowIOSubmit[LOWIO_OP_FSCTL](
    _Inout_ PRX_CONTEXT RxContext
)
{ ... }
```

パラメーター

RxContext [入力、出力]

RX へのポインター_CONTEXT 構造体。このパラメーターには、操作を要求している IRP が含まれています。

戻り値

MRxLowIOSubmit[LOWIO_OP_FSCTL] ステータスを返します_次のいずれかなど、成功した場合に成功した場合、または、適切な NTSTATUS の値します。

NTSTATUS	説明
STATUS_CONNECTION_DISCONNECTED	接続が切断されました。
STATUS_INSUFFICIENT_RESOURCES	要求を完了するリソースの不足が発生しました。
STATUS_INVALID_DEVICE_REQUEST	無効なデバイス、要求が指定されました。
STATUS_INVALID_NETWORK_RESPONSE	無効な応答は、リモート サーバーから受信しました。
STATUS_INVALID_PARAMETER	無効なパラメーターがで指定された <i>RxContext</i> します。
STATUS_LINK_FAILED	要求を完了するリモート サーバーへの再接続に失敗しました。
STATUS_NOT_IMPLEMENTED	このルーチンが実装されていません。
STATUS_NOT_SUPPORTED	指定された FSCTL はネットワーク ミニリダイレクターによってサポートされていません。

STATUS_UNSUCCESSFUL	呼び出しに失敗しました。

注釈

RDBSS 呼び出し *MRxLowIOSubmit[LOWIO_OP_FSCTL]* 受信に応答する [IRP_MJ_ファイル_システム_コントロール](#) 要求。

呼び出しの前に *MRxLowIOSubmit[LOWIO_OP_FSCTL]*、RDBSS、RX では、次のメンバーを変更します `_CONTEXT` 構造体が、が指す *RxContext* パラメーター。

`LowIoContext.Operation` `LOWIO` にメンバーが設定されている `_OP_FSCTL` します。

`LowIoContext.ResourceThreadId` メンバー RDBSS で操作を開始したプロセスのスレッドに設定されます。

`LowIoContext.ParamsFor.FsCtl.FsControlCode` メンバー `FSCTL` の主要なコントロール コードに設定されます。

`LowIoContext.ParamsFor.FsCtl.MinorFunction` メンバー `FSCTL` マイナー制御コードに設定されます。

`LowIoContext.ParamsFor.FsCtl.pInputBuffer` メンバーは、入力バッファーに設定します。

`LowIoContext.ParamsFor.FsCtl.InputBufferLength` メンバーは、入力バッファーの長さに設定します。

`LowIoContext.ParamsFor.FsCtl.pOutputBuffer` メンバーは、出力バッファーに設定されます。

`LowIoContext.ParamsFor.FsCtl.OutputBufferLength` メンバーは、出力バッファーの長さに設定します。

ネットワークのミニ リダイレクターによって処理されるファイル システム コントロールのコード (`FSCTL`) 要求は、いくつかのカテゴリのいずれかに分類できます。

- `FSCTLs` 実装 RDBSS とネットワークのミニ リダイレクターで使します。
- `FSCTLs` 実装され、ネットワークのミニ リダイレクターでのみ使します。
- ネットワークのミニ リダイレクターことはありませんが表示されないようにする `FSCTLs` します。これらの `FSCTLs` デバッグのためいます。

中に、*MRxLowIOSubmit[LOWIO_OP_FSCTL]* ルーチンは、処理中、`LowIoContext.ResourceThreadId` RXのメンバー `_コンテキスト` を RDBSS で操作を開始したプロセスのスレッドを示すことが保証されます。

`LowIoContext.ResourceThreadId` RX のメンバー `_コンテキスト` は、別のスレッドの代わりに入力のリソースを解放するために使用できます。非同期のルーチンが完了したら、最初のスレッドから取得された入力のリソースを解放できます。

必要条件

対象プラットフォーム	Desktop
Header	Mrx.h (Mrx.h を含む)

関連項目

[MRxLowIOSubmit\[LOWIO_OP_EXCLUSIVELOCK\]](#)

[MRxLowIOSubmit\[LOWIO_OP_IOCTL\]](#)

[MRxLowIOSubmit\[LOWIO_OP_NOTIFY_CHANGE_DIRECTORY\]](#)

MRxLowIOSubmit[LOWIO_OP_READ]

MRxLowIOSubmit[LOWIO_OP_SHAREDLOCK]

MRxLowIOSubmit[LOWIO_OP_UNLOCK]

MRxLowIOSubmit[LOWIO_OP_UNLOCK_MULTIPLE]

MRxLowIOSubmit[LOWIO_OP_WRITE]

MRxLowIOSubmit[LOWIO_OP_IOCTL]ルーチン

2020/04/24 • [Edit Online](#)

MRxLowIOSubmit[LOWIO_OP_IOCTL] ルーチン呼び出してRDBSSネットワークに、I/O システムのコントロール要求を発行するにはミニ リダイレクター。

構文

```
PMRX_CALLDOWN MRxLowIOSubmit[LOWIO_OP_IOCTL];

NTSTATUS MRxLowIOSubmit[LOWIO_OP_IOCTL](
    _Inout_ PRX_CONTEXT RxContext
)
{ ... }
```

パラメーター

RxContext [入力、出力]

RX へのポインター_CONTEXT 構造体。このパラメーターには、操作を要求している IRP が含まれています。

戻り値

MRxLowIOSubmit[LOWIO_OP_IOCTL] ステータスを返します_次のいずれかなど、成功した場合に成功した場合、または、適切な NTSTATUS の値します。

NTSTATUS	説明
STATUS_INSUFFICIENT_RESOURCES	要求を完了するリソースの不足が発生しました。
STATUS_INVALID_DEVICE_REQUEST	無効なデバイス、要求が指定されました。
STATUS_INVALID_PARAMETER	無効なパラメーターが指定された <i>RxContext</i> します。
STATUS_NOT_IMPLEMENTED	このルーチンが実装されていません。
STATUS_NOT_SUPPORTED	指定された IOCTL はネットワーク ミニリダイレクターではサポートされていません。

注釈

RDBSS 呼び出し*MRxLowIOSubmit[LOWIO_OP_IOCTL]* 受信に応答する [IRP_MJ_デバイス_コントロール](#)または [IRP_MJ_内部_デバイス_コントロール](#) 要求します。

呼び出しの前に*MRxLowIOSubmit[LOWIO_OP_IOCTL]*、RDBSS、RX では、次のメンバーを変更します_CONTEXT 構造体が、が指す *RxContext*パラメーター。

LowIoContext.Operation LOWIO にメンバーが設定されている_OP_IOCTL します。

`LowIoContext.ResourceThreadId`メンバー RDBSS で操作を開始したプロセスのスレッドに設定されます。

`LowIoContext.ParamsFor.IoCtl.IoControlCode`メンバー IOCTL 制御コードに設定されます。

`LowIoContext.ParamsFor.IoCtl.pInputBuffer`メンバーは、入力バッファに設定します。

`LowIoContext.ParamsFor.IoCtl.InputBufferLength`メンバーは、入力バッファの長さに設定します。

`LowIoContext.ParamsFor.IoCtl.pOutputBuffer`メンバーは、出力バッファに設定されます。

`LowIoContext.ParamsFor.IoCtl.OutputBufferLength`メンバーは、出力バッファの長さに設定します。

中に、`MRxLowIOSubmit[LOWIO_OP_IOCTL]` ルーチンは、処理中、`LowIoContext.ResourceThreadId` RXのメンバー_コンテキストを RDBSS で操作を開始したプロセスのスレッドを示すことが保証されます。

`LowIoContext.ResourceThreadId` RX のメンバー_コンテキストは、別のスレッドの代わりの入力のリソースを解放するために使用できます。非同期のルーチンが完了したら、最初のスレッドから取得された入力のリソースを解放できます。

必要条件

対象プラットフォーム	Desktop
Header	Mrx.h (Mrx.h を含む)

関連項目

[MRxLowIOSubmit\[LOWIO_OP_EXCLUSIVELOCK\]](#)

[MRxLowIOSubmit\[LOWIO_OP_FSCTL\]](#)

[MRxLowIOSubmit\[LOWIO_OP_NOTIFY_CHANGE_DIRECTORY\]](#)

[MRxLowIOSubmit\[LOWIO_OP_READ\]](#)

[MRxLowIOSubmit\[LOWIO_OP_SHAREDLOCK\]](#)

[MRxLowIOSubmit\[LOWIO_OP_UNLOCK\]](#)

[MRxLowIOSubmit\[LOWIO_OP_UNLOCK_MULTIPLE\]](#)

[MRxLowIOSubmit\[LOWIO_OP_WRITE\]](#)

MRxLowIOSubmit[LOWIO_OP__ディレクトリ]ルーチンに変更を通知_

2020/04/24 • [Edit Online](#)

MRxLowIOSubmit[LOWIO_OP_通知_change_directory] ルーチンは、ディレクトリ変更通知操作のためにネットワークミニリダイレクターに要求を発行するために、[RDBSS](#)によって呼び出されます。

構文

```
PMRX_CALLEDOWN MRxLowIOSubmit[LOWIO_OP_NOTIFY_CHANGE_DIRECTORY];

NTSTATUS MRxLowIOSubmit[LOWIO_OP_NOTIFY_CHANGE_DIRECTORY](
    _Inout_ PRX_CONTEXT RxContext
)
{ ... }
```

パラメーター

RxContext [in, out]

RX_コンテキスト構造体へのポインター。このパラメーターには、操作を要求している IRP が含まれています。

戻り値

MRxLowIOSubmit[LOWIO_OP__変更の通知_ディレクトリ] は成功時に成功したか、適切な NTSTATUS 値 (次のいずれか) を返します。

戻り値	説明
STATUS_FILE_CLOSED	FCB 構造体が取得されましたが、関連付けられている SRV_OPEN 構造体が閉じられています。
STATUS_INSUFFICIENT_RESOURCES	要求を完了するためのリソースが不足しています。
STATUS_INVALID_DEVICE_REQUEST	無効なデバイス要求が指定されました。
STATUS_INVALID_PARAMETER	<i>RxContext</i> で無効なパラメーターが指定されました。
STATUS_NOT_IMPLEMENTED	このルーチンは実装されていません。
STATUS_NOT_SUPPORTED	指定された要求は、ネットワークミニリダイレクターでサポートされていません。

注釈

RDBSS は、[IRP_MJ\]directory_CONTROL](#) 要求の受信に応答して、*MRxLowIOSubmit[lowio_OP__DIRECTORY_* の

変更を_に通知します。

*MRxLowIOSubmit[LOWIO_OP_DIRECTORY]の変更を_通知*する前に、*RxContext*lパラメーターによって示されるRX_CONTEXT 構造体の次のメンバーが変更されます。

Lowiocontext 操作のメンバーは *lowio_OP_通知__ディレクトリ*の変更を通知するように設定されています。

Lowiocontext threadidメンバーは、RDBSS で操作を開始したプロセスのスレッドに設定されます。

*Irpsp->フラグ*に *SL_WATCH_*ツリービットが設定されている場合は、**WatchTree**メンバーがTRUEに設定されます。

Lowiocontext. Paramchangedirectoryのメンバーは、*irpsp->Parameters. notifydirectory.* の値に設定されています。

Lowiocontext. NotifyChangeDirectory. NotificationBufferLengthメンバーは、*irpsp->Parameters. Notifydirectory. Length*の値に設定されます。

*MmGetSystemAddressForMdlSafe MdlAddress*および *NormalPagePriority* で>、を呼び出すことによって返される値に設定されている値に、**lowiocontext**。ユーザーバッファもプローブされ、書き込みアクセス用にロックされます。

ディレクトリ変更通知操作は、通常、非同期操作としてネットワークミニリダイレクターによって実装されます。これは、かなりの時間がかかる可能性があるためです。通常、この操作は、変更通知を要求しているリモートサーバーへのネットワーク要求の送信で構成されます。この応答は、必要な変更がサーバー上で影響を受ける場合に取得されます。これは、ネットワークミニリダイレクターがローカルで開始された取り消し処理のために一意のコンテキスト値を登録する必要がある操作の例です。

*MRxLowIOSubmit[LOWIO_OP_ディレクトリ]*ルーチンが処理されていることを通知_変更しますが、RX_コンテキストの**Lowiocontext** スレッドメンバーは、RDBSS で操作を開始したプロセスのスレッド。**Lowiocontext threadid**メンバーを使用すると、別のスレッドに代わって FCB 構造体を解放できます。非同期ルーチンが完了すると、初期スレッドから取得された FCB 構造体が解放されます。FCB 構造体は、[RxReleaseFcbResourceForThreadInMRx](#)を呼び出すことによって解放できます。

要件

対象プラットフォーム	Desktop
Header	Mrx .h (Mrx を含む)

関連項目

[MRxLowIOSubmit\[LOWIO_OP_EXCLUSIVELOCK\]](#)

[MRxLowIOSubmit\[LOWIO_OP_FSCTL\]](#)

[MRxLowIOSubmit\[LOWIO_OP_IOCTL\]](#)

[MRxLowIOSubmit\[LOWIO_OP_読み取り\]](#)

[MRxLowIOSubmit\[LOWIO_OP_SHAREDLOCK\]](#)

[MRxLowIOSubmit\[LOWIO_OP_ロック解除\]](#)

[MRxLowIOSubmit\[LOWIO_OP_ロック解除_複数\]](#)

[MRxLowIOSubmit\[LOWIO_OP_書き込み\]](#)

[RxReleaseFcbResourceForThreadInMRx](#)

MRxLowIOSubmit[LOWIO_OP_読み取り] ルーチン

2020/04/24 • [Edit Online](#)

MRxLowIOSubmit[LOWIO_OP_read] ルーチンは、ネットワークミニリダイレクターに対する読み取り要求を発行するために、[RDBSS](#)によって呼び出されます。

構文

```
PMRX_CALLDOWN MRxLowIOSubmit[LOWIO_OP_READ];

NTSTATUS MRxLowIOSubmit[LOWIO_OP_READ](
    _Inout_ PRX_CONTEXT RxContext
)
{ ... }
```

パラメーター

RxContext [in, out]

RX_コンテキスト構造体へのポインター。このパラメーターには、操作を要求している IRP が含まれています。

戻り値

MRxLowIOSubmit[LOWIO_OP_読み取り] は正常終了時に成功したか、または次のいずれかのような適切な NTSTATUS 値を返します。

STATUS_FILE_CLOSED	FCB 構造体が取得されましたが、関連付けられている SRV_OPEN 構造体が閉じられています。
STATUS_INSUFFICIENT_RESOURCES	要求を完了するためのリソースが不足しています。
STATUS_INVALID_DEVICE_REQUEST	無効なデバイス要求が指定されました。
STATUS_INVALID_PARAMETER	<i>RxContext</i> で無効なパラメーターが指定されました。
STATUS_NOT_IMPLEMENTED	このルーチンは実装されていません。
STATUS_NOT_SUPPORTED	指定された要求は、ネットワークミニリダイレクターでサポートされていません。

注釈

RDBSS は、[IRP_MJ_読み取り](#) 要求の受信に応答して、*MRxLowIOSubmit[lowio_OP_read]* を呼び出します。

MRxLowIOSubmit[LOWIO_OP_READ] を呼び出す前に、*RxContext* のパラメーターによって示されている RX_CONTEXT 構

造体内の次のメンバーが RDBSS によって変更されます。

Lowiocontext 操作のメンバーが LOWIO_OP_READ に設定されています。

Lowiocontext threadidメンバーは、RDBSS で操作を開始したプロセスのスレッドに設定されます。

Lowiocontext. ReadWrite. キーメンバーは、irps->パラメーターの値に設定されます。

READWRITEFLAGのパラメーターには LOWIO__PAGING_io ビットが設定されています。この場合、Irp->フラグには、IRP_ページング_i/o ビットが設定されています。

IoReadAccess のパラメーターには、ユーザー バッファのロックが設定されています。

ByteCountメンバーは、irps->のパラメーターの値に設定されています。読み取りの長さ。

通常、読み取り要求はかなりの時間がかかるため、非同期操作としてネットワークミニリダイレクターによって実装されます。通常、この操作は、リモートサーバーへのネットワーク要求の送信で構成されます。サーバーで読み取り要求が完了すると、応答が取得されます。これは、ネットワークミニリダイレクターがキャンセル処理をローカルで行うためにコンテキストを登録する必要がある操作の例です。

MRxLowIOSubmit[LOWIO_OP_READ] ルーチンが処理されている間、RX_コンテキストのlowiocontextスレッドメンバーは、を開始したプロセスのスレッドを示すことが保証されます。RDBSS での操作です。Lowiocontext threadidメンバーを使用すると、別のスレッドに代わって FCB 構造体を解放できます。非同期ルーチンが完了すると、初期スレッドから取得された FCB 構造体が解放されます。FCB 構造体は、[RxReleaseFcbResourceForThreadInMRx](#)を呼び出すことによって解放できます。

要件

対象プラットフォーム	Desktop
Header	Mrx.h (Mrx を含む)

関連項目

[MRxLowIOSubmit\[LOWIO_OP_EXCLUSIVELOCK\]](#)

[MRxLowIOSubmit\[LOWIO_OP_FSCTL\]](#)

[MRxLowIOSubmit\[LOWIO_OP_IOCTL\]](#)

[MRxLowIOSubmit\[LOWIO_OP___ディレクトリの変更を通知\]](#)

[MRxLowIOSubmit\[LOWIO_OP_SHAREDLOCK\]](#)

[MRxLowIOSubmit\[LOWIO_OP_ロック解除\]](#)

[MRxLowIOSubmit\[LOWIO_OP_ロック解除_複数\]](#)

[MRxLowIOSubmit\[LOWIO_OP_書き込み\]](#)

[RxReleaseFcbResourceForThreadInMRx](#)

MRxLowIOSubmit[LOWIO_OP_SHAREDLOCK]

ルーチン

2020/04/24 • [Edit Online](#)

MRxLowIOSubmit[LOWIO_OP_sharedlock] ルーチンは、ネットワークリダイレクターがファイルオブジェクトに対して共有ロックを開くように要求するために、[RDBSS](#)によって呼び出されます。

構文

```
PMRX_CALLDOWN MRxLowIOSubmit[LOWIO_OP_SHAREDLOCK];

NTSTATUS MRxLowIOSubmit[LOWIO_OP_SHAREDLOCK](
    _Inout_ PRX_CONTEXT RxContext
)
{ ... }
```

パラメーター

RxContext [in, out]

RX_コンテキスト構造体へのポインター。このパラメーターには、操作を要求している IRP が含まれています。

戻り値

MRxLowIOSubmit[LOWIO_OP_SHAREDLOCK] は、正常に完了したか、次のいずれかに該当する NTSTATUS 値を_返します。

NTSTATUS	説明
STATUS_CONNECTION_DISCONNECTED	接続が切断されました。
STATUS_INSUFFICIENT_RESOURCES	要求を完了するためのリソースが不足しています。
STATUS_INVALID_BUFFER_SIZE	要求されたバッファサイズが大きすぎます。
STATUS_INVALID_NETWORK_RESPONSE	リモートサーバーから無効な応答を受信しました。
STATUS_INVALID_PARAMETER	<i>RxContext</i> で無効なパラメーターが指定されました。
STATUS_LINK_FAILED	リモートサーバーに再接続して要求を完了できませんでした。
STATUS_NOT_IMPLEMENTED	このルーチンは実装されていません。
STATUS_SHARING_VIOLATION	共有違反が発生しました。

MRxLowIOSubmit[LOWIO_OP_読み取り]

MRxLowIOSubmit[LOWIO_OP_ロック解除]

MRxLowIOSubmit[LOWIO_OP_ロック解除_複数]

MRxLowIOSubmit[LOWIO_OP_書き込み]

RxReleaseFcbResourceForThreadInMRx

MRxLowIOSubmit[LOWIO_OP_] ルーチンのロック解除

2020/04/24 • [Edit Online](#)

MRxLowIOSubmit[LOWIO_OP_UNLOCK] ルーチンは、ネットワークミニリダイレクターがファイルオブジェクトの1つのロックを解除するように要求するために、[RDBSS](#)によって呼び出されます。

構文

```
PMRX_CALLEDOWN MRxLowIOSubmit[LOWIO_OP_UNLOCK];

NTSTATUS MRxLowIOSubmit[LOWIO_OP_UNLOCK](
    _Inout_ PRX_CONTEXT RxContext
)
{ ... }
```

パラメーター

RxContext [in, out]

RX_コンテキスト構造体へのポインター。このパラメーターには、操作を要求している IRP が含まれています。

戻り値

MRxLowIOSubmit[LOWIO_OP_ロック解除] は、正常に完了したか、次のいずれかの適切な NTSTATUS 値を返します。

NTSTATUS	説明
STATUS_CONNECTION_DISCONNECTED	接続が切断されました。
STATUS_INSUFFICIENT_RESOURCES	要求を完了するためのリソースが不足しています。
STATUS_INVALID_NETWORK_RESPONSE	リモートサーバーから無効な応答を受信しました。
STATUS_INVALID_PARAMETER	<i>RxContext</i> で無効なパラメーターが指定されました。
STATUS_LINK_FAILED	リモートサーバーに再接続して要求を完了できませんでした。
STATUS_NOT_IMPLEMENTED	このルーチンは実装されていません。
STATUS_SHARING_VIOLATION	共有違反が発生しました。
STATUS_UNSUCCESSFUL	呼び出しに失敗しました。

注釈

RDBSS は、*MRxLowIOSubmit[LOWIO_OP_ロック解除]*を呼び出します。IRP [_MJ_LOCK_CONTROL](#) 要求の受信に応答して]ロック解除_の小さなコードで_ロック解除1.

*MRxLowIOSubmit[LOWIO_OP_ロックを]解除*する前に、*RxContext*パラメーターによって示されている RX_CONTEXT 構造体内の次のメンバーが RDBSS によって変更されます。

Lowiocontext. Operationメンバーは LOWIO_OP_UNLOCK に設定されています。

Lowiocontext threadidメンバーは、RDBSS で操作を開始したプロセスのスレッドに設定されます。

QuadPart の値には、Lowiocontext ... byteoffset メンバーは、irpsp->パラメーターの値に設定されます。

Lowiocontext. Paramsfor Locks. キーメンバーは、irpsp->Parameters. lockcontrol. keyの値に設定されます。

QuadPartメンバーには、irpsp->パラメーターの値が設定されています.Lockcontrol. length. 。

RX_コンテキスト構造のLowiocontext 操作メンバーは、実行する低 i/o 操作を指定します。低 i/o ルーチンのいくつかは、ネットワークミニリダイレクターで同じルーチンを指すことができます。これは、このLowiocontext 操作のメンバーが、要求された低 i/o 操作を区別するために使用できるためです。たとえば、ファイルロックに関連するすべての i/o 呼び出しで、ネットワークミニリダイレクターで同じ低 i/o ルーチンを呼び出すことができます。また、このルーチンは、要求されたロック操作とロック解除操作を区別するためにLowiocontext. operationメンバーを使用できます。

MRxLowIOSubmit[LOWIO_OP_ロック解除] ルーチンの完了に時間がかかる場合、ネットワークミニリダイレクタードライバーは、ネットワーク通信を開始する前に、FCB 構造体を解放する必要があります。FCB 構造体は、[RxReleaseFcbResourceForThreadInMRx](#)を呼び出すことによって解放できます。*MRxLowIOSubmit[LOWIO_OP_ロック解除]* ルーチンが処理されている間、RX_コンテキストのlowiocontextスレッドメンバーは、開始したプロセスのスレッドを示すことが保証されます。RDBSS の操作。

RX_コンテキストのLowiocontext threadidメンバーを使用すると、別のスレッドに代わって FCB 構造体を解放できます。非同期ルーチンが完了すると、初期スレッドから取得された FCB 構造体が解放されます。

要件

対象プラットフォーム	Desktop
Header	Mrx.h (Mrx を含む)

関連項目

- [MRxLowIOSubmit\[LOWIO_OP_EXCLUSIVELOCK\]](#)
- [MRxLowIOSubmit\[LOWIO_OP_FSCTL\]](#)
- [MRxLowIOSubmit\[LOWIO_OP_IOCTL\]](#)
- [MRxLowIOSubmit\[LOWIO_OP___ディレクトリの変更を通知\]](#)
- [MRxLowIOSubmit\[LOWIO_OP_読み取り\]](#)
- [MRxLowIOSubmit\[LOWIO_OP_SHAREDLOCK\]](#)
- [MRxLowIOSubmit\[LOWIO_OP_ロック解除_複数\]](#)
- [MRxLowIOSubmit\[LOWIO_OP_書き込み\]](#)

MRxLowIOSubmit[LOWIO_OP_ロック解除_複数]ルーチン

2020/04/24 • [Edit Online](#)

MRxLowIOSubmit[LOWIO_OP_UNLOCK_複数] ルーチンは、ネットワークミニリダイレクターがファイルオブジェクトに保持されている複数のロックを削除するように要求するために、[RDBSS](#)によって呼び出されます。

構文

```
PMRX_CALLDOWN MRxLowIOSubmit[LOWIO_OP_UNLOCK_MULTIPLE];

NTSTATUS MRxLowIOSubmit[LOWIO_OP_UNLOCK_MULTIPLE](
    _Inout_ PRX_CONTEXT RxContext
)
{ ... }
```

パラメーター

RxContext [in, out]

RX_コンテキスト構造体へのポインター。このパラメーターには、操作を要求している IRP が含まれています。

戻り値

MRxLowIOSubmit[LOWIO_OP_ロック解除_複数の] が成功したときに成功したか、適切な NTSTATUS 値 (次のいずれかを返します)。

NTSTATUS	説明
STATUS_CONNECTION_DISCONNECTED	接続が切断されました。
STATUS_INSUFFICIENT_RESOURCES	要求を完了するためのリソースが不足しています。
STATUS_INVALID_NETWORK_RESPONSE	リモートサーバーから無効な応答を受信しました。
STATUS_INVALID_PARAMETER	<i>RxContext</i> で無効なパラメーターが指定されました。
STATUS_LINK_FAILED	リモートサーバーに再接続して要求を完了できませんでした。
STATUS_NOT_IMPLEMENTED	このルーチンは実装されていません。
STATUS_SHARING_VIOLATION	共有違反が発生しました。
STATUS_UNSUCCESSFUL	呼び出しに失敗しました。

注釈

RDBSS は、*MRxLowIOSubmit[LOWIO_OP_ロック解除]* を呼び出します。IRP [JMJ_LOCK_CONTROL](#) 要求の受信に
応答して、複数の *_* の *_* ロックが解除されます。すべてまたは IRP *_* ロック解除すると *_* キーによってすべての *_* ロック解除 *_* ます。

MRxLowIOSubmit[LOWIO_OP_ 複数の] の *_* ロックを解除する前に、*RxContext* *d* パラメーターによって示される RX_コンテキスト構造の次のメンバーを変更します。

Lowiocontext. Operation メンバーは `lowio_OP_UNLOCK_MULTIPLE` に設定されています。

Lowiocontext threadid メンバーは、RDBSS で操作を開始したプロセスのスレッドに設定されます。

Lowiocontext. Locks. LockList メンバーは、`LOWIO_LOCK_list` 要素のリストに設定されます。各要素は、ロックを解除する範囲を指定します。

ロックを解除するバイト範囲は、RX の `_CONTEXT` 構造体の、**Lowiocontext ... locklist** メンバーに指定されています。
`LOWIO_ロック_` の一覧の構造は次のとおりです。

```
typedef struct _LOWIO_LOCK_LIST {
    struct _LOWIO_LOCK_LIST *Next;
    ULONG LockNumber;
    RXVBO ByteOffset;
    LONGLONG Length;
    ULONG Key;
    BOOLEAN ExclusiveLock;
} LOWIO_LOCK_LIST, *PLOWIO_LOCK_LIST;
```

RX_CONTEXT の **Lowiocontext 操作** のメンバーは、実行する低 i/o 操作を指定します。低 i/o ルーチンのいくつかは、ネットワークミニリダイレクターで同じルーチンを指している可能性があります。これは、要求された低 i/o 操作を区別するために、**Lowiocontext 操作** メンバーを使用できるためです。たとえば、ファイルロックに関連するすべての i/o 呼び出しで、ネットワークミニリダイレクターで同じ低 i/o ルーチンを呼び出すことができます。また、このルーチンは、要求されたロック操作とロック解除操作を区別するために **Lowiocontext. operation** メンバーを使用できます。

MRxLowIOSubmit[LOWIO_OP_ロック解除_ 複数の] ルーチンの完了に長い時間がかかる場合、ネットワークミニリダイレクタードライバはネットワークを開始する前に FCB 構造を解放する必要があります。communication. FCB 構造体は、[RxReleaseFcbResourceForThreadInMRx](#) を呼び出すことによって解放できます。*MRxLowIOSubmit[lowio_OP_ロック解除_ 複数の]* ルーチンが処理されている間、RX_コンテキストの **lowiocontext** スレッドメンバーは、RDBSS で操作を開始したプロセス。

RX_コンテキストの **Lowiocontext threadid** メンバーを使用すると、別のスレッドに代わって FCB 構造体を解放できます。非同期ルーチンが完了すると、初期スレッドから取得された FCB 構造体が解放されます。

要件

対象プラットフォーム	Desktop
Header	Mrx.h (Mrx を含む)

関連項目

[MRxLowIOSubmit\[LOWIO_OP_EXCLUSIVELOCK\]](#)

[MRxLowIOSubmit\[LOWIO_OP_FSCTL\]](#)

[MRxLowIOSubmit\[LOWIO_OP_IOCTL\]](#)

MRxLowIOSubmit[LOWIO_OP___ディレクトリの変更を通知]

MRxLowIOSubmit[LOWIO_OP_読み取り]

MRxLowIOSubmit[LOWIO_OP_SHAREDLOCK]

MRxLowIOSubmit[LOWIO_OP_ロック解除]

MRxLowIOSubmit[LOWIO_OP_書き込み]

RxReleaseFcbResourceForThreadInMRx

MRxLowIOSubmit[LOWIO_OP_書き込み] ルーチン

2020/04/24 • [Edit Online](#)

MRxLowIOSubmit[LOWIO_OP_write] ルーチンは、ネットワークミニリダイレクターに書き込み要求を発行するために、[RDBSS](#)によって呼び出されます。

構文

```
PMRX_CALLDOWN MRxLowIOSubmit[LOWIO_OP_WRITE];

NTSTATUS MRxLowIOSubmit[LOWIO_OP_WRITE](
    _Inout_ PRX_CONTEXT RxContext
)
{ ... }
```

パラメーター

RxContext [in, out]

RX_コンテキスト構造体へのポインター。このパラメーターには、操作を要求している IRP が含まれています。

戻り値

MRxLowIOSubmit[LOWIO_OP_書き込み] は正常に完了したか、適切な NTSTATUS 値 (次のいずれかの例)_を返します。

STATUS_FILE_CLOSED	FCB 構造体が取得されましたが、関連付けられている SRV_OPEN 構造体が閉じられています。
STATUS_INSUFFICIENT_RESOURCES	要求を完了するためのリソースが不足しています。
STATUS_INVALID_DEVICE_REQUEST	無効なデバイス要求が指定されました。
STATUS_INVALID_PARAMETER	<i>RxContext</i> で無効なパラメーターが指定されました。
STATUS_NOT_IMPLEMENTED	このルーチンは実装されていません。
STATUS_NOT_SUPPORTED	指定された要求は、ネットワークミニリダイレクターでサポートされていません。

注釈

RDBSS は、[IRP_MJ_書き込み](#) 要求の受信に応答して、*MRxLowIOSubmit[lowio_OP_書き込み]* を呼び出します。

MRxLowIOSubmit[LOWIO_OP_WRITE] を呼び出す前に、*RxContextI* パラメーターが指す `RX_CONTEXT` 構造体の次のメンバーを変更します。

`Lowiocontext. Operation` メンバーは `LOWIO_OP_WRITE` に設定されています。

`Lowiocontext threadid` メンバーは、RDBSS で操作を開始したプロセスのスレッドに設定されます。

`Lowiocontext. ReadWrite. キー` メンバーは、`irpsp->` パラメーターの値に設定されます。

`READWRITEFLAG` のパラメーターには `LOWIO__PAGING_io` ビットが設定されています。この場合、`Irp->フラグ` には、`IRP_ページング_i/o` ビットが設定されています。

パラメーターを使用して、`IoWriteAccess` 用にロックされているユーザーバッファーに設定します。

`ByteCount` メンバーは、`irpsp->` のパラメーターの値に設定されています。書き込み。長さ。

書き込み要求は、通常、非同期操作としてネットワークミニダイレクターによって実装されます。これは、かなりの時間がかかる可能性があるためです。通常、この操作は、リモートサーバーへのネットワーク要求の送信で構成されます。応答は、サーバーで書き込み要求が完了したときに取得されます。これは、ネットワークミニダイレクターがキャンセル処理をローカルで行うためにコンテキストを登録する必要がある操作の例です。

MRxLowIOSubmit[lowio_OP_書き込み] ルーチンが処理されている間、`RX_コンテキスト` の `lowiocontext` スレッドメンバーは、を開始したプロセスのスレッドを示すことが保証されます。RDBSS での操作です。`Lowiocontext threadid` メンバーを使用すると、別のスレッドに代わって FCB 構造体を解放できます。非同期ルーチンが完了すると、初期スレッドから取得された FCB 構造体が解放されます。FCB 構造体は、[RxReleaseFcbResourceForThreadInMRx](#) を呼び出すことによって解放できます。

要件

対象プラットフォーム	Desktop
Header	Mrx.h (Mrx を含む)

関連項目

[MRxLowIOSubmit\[LOWIO_OP_EXCLUSIVELOCK\]](#)

[MRxLowIOSubmit\[LOWIO_OP_FSCTL\]](#)

[MRxLowIOSubmit\[LOWIO_OP_IOCTL\]](#)

[MRxLowIOSubmit\[LOWIO_OP___ディレクトリの変更を通知\]](#)

[MRxLowIOSubmit\[LOWIO_OP_読み取り\]](#)

[MRxLowIOSubmit\[LOWIO_OP_SHAREDLOCK\]](#)

[MRxLowIOSubmit\[LOWIO_OP_ロック解除\]](#)

[MRxLowIOSubmit\[LOWIO_OP_ロック解除_複数\]](#)

[RxReleaseFcbResourceForThreadInMRx](#)

MRxQueryDirectory ルーチン

2020/04/24 • [Edit Online](#)

MRxQueryDirectory ルーチンは [RDBSS](#) によって呼び出され、ネットワークミニリダイレクターがファイルディレクトリに関する情報を照会するように要求します。

構文

```
PMRX_CALLEDOWN MRxQueryDirectory;  
  
NTSTATUS MRxQueryDirectory(  
    _Inout_ PRX_CONTEXT RxContext  
)  
{ ... }
```

パラメーター

RxContext [in, out]

RX_コンテキスト構造体へのポインター。このパラメーターには、操作を要求している IRP が含まれています。

戻り値

MRxQueryDirectory は正常に完了した状態_成功したか、または次のいずれかのような NTSTATUS 値を返します。

名前	説明
STATUS_ACCESS_DENIED	呼び出し元には、この操作に対する適切なセキュリティが不足しています。
STATUS_INSUFFICIENT_RESOURCES	クエリを完了するためのリソースが不足しています。
STATUS_INVALID_NETWORK_RESPONSE	リモートサーバーから無効なファイル情報バッファーを受信したか、返されたファイル名の長さが許容最大長を超えました。
STATUS_INVALID_PARAMETER	RxContext パラメーターによって示されている RX_CONTEXT 構造体の FileInformationClass メンバーに無効な fileinformationclass が指定されました。
STATUS_LINK_FAILED	リモートサーバーに再接続してクエリを完了できませんでした。
STATUS_NO_SUCH_FILE	クエリでエントリが見つかりませんでした。
STATUS_SHARING_VIOLATION	共有違反が発生しました。

注釈

*MRxQueryDirectory*を呼び出す前に、RDBSS は、*RxContext*パラメーターによって示される RX_コンテキスト構造内の次のメンバーを変更します。

Info. fileinformationclassメンバーはirpsp->Parameters. Querydirectory. fileinformationclassに設定されます。

情報バッファのメンバーは、i/o 要求パケットからのユーザーバッファに設定されます。このバッファは、必要に応じて、RDBSS によって既にロックされています。

LengthRemainingメンバーはirpsp->Parameters. Querydirectory. Lengthに設定されています。

Querydirectory. fileindexメンバーはirpsp->Parameters. Querydirectory. fileindexに設定されます。

Irpsp->フラグに SL_再起動_スキャンビットがある場合、RestartScanメンバーは0以外に設定されます。

Irpsp->フラグに SL__単一_エントリビットが返される場合、このメンバーは0以外の値に設定されます。

Irpsp->フラグに SL_インデックス_指定されたビットがオンになっている場合、指定されたメンバーは0以外に設定されます。

関連する FOBX のUnicodeQueryTemplateメンバーが NULL で、Fobx のFlagsメンバーが fobx_フラグ_一致しない場合、querydirectory. initialqueryメンバーは0以外の値に設定され_ALL bit on。

ワイルドカードクエリ ("*. *"など) の場合、RDBSS は、関連付けられている FOBX のUnicodeQueryTemplateメンバーを、渡されたワイルドカードクエリに設定します。

*MRxQueryDirectory*から返されたときに RX_コンテキスト構造のPostrequestメンバーがTRUEの場合、RDBSS は[RxFsdPostRequest](#)を呼び出して、によって処理するために rx_コンテキスト構造をワーカーキューに渡します。ファイルシステムプロセス (FSP)。

要件

対象プラットフォーム	Desktop
Header	Mrx.h (Mrx を含む)

関連項目

[MRxIsValidDirectory](#)

[MRxQueryEaInfo](#)

[MRxQueryFileInfo](#)

[MRxQueryQuotaInfo](#)

[MRxQuerySdInfo](#)

[MRxQueryVolumeInfo](#)

[MRxSetEaInfo](#)

[MRxSetFileInfo](#)

[MRxSetFileInfoAtCleanup](#)

[MRxSetQuotaInfo](#)

[MRxSetSdInfo](#)

MRxSetVolumeInfo

RxFsdPostRequest

MRxQueryEaInfo ルーチン

2020/04/24 • [Edit Online](#)

MRxQueryEaInfo ルーチンは、ネットワークミニリダイレクターがファイルシステムオブジェクトの拡張属性情報を照会するように要求するために、[RDBSS](#)によって呼び出されます。

構文

```
PMRX_CALLEDOWN MRxQueryEaInfo;  
  
NTSTATUS MRxQueryEaInfo(  
    _Inout_ PRX_CONTEXT RxContext  
)  
{ ... }
```

パラメーター

RxContext [in, out]

RX_コンテキスト構造体へのポインター。このパラメーターには、操作を要求している IRP が含まれています。

戻り値

MRxQueryEaInfo は正常に完了した状態_成功したか、または次のいずれかのような NTSTATUS 値を返します。

■ ■ ■ ■ ■	■ ■
STATUS_ACCESS_DENIED	呼び出し元には、この操作に対する適切なセキュリティが不足しています。
STATUS_BUFFER_OVERFLOW	拡張属性情報を受信するバッファーが小さすぎます。 この戻り値は成功と見なす必要があり、可能な限り多くの有効なデータが、 <i>RxContext</i> パラメーターによって示される RX_CONTEXT 構造体の ■ ■ ■ ■ ■ のメンバーに返される必要があります。
STATUS_BUFFER_TOO_SMALL	バッファーが小さすぎて要求されたデータを受け取ることができません。 この値が返される場合、 <i>RxContext</i> パラメーターが指す RX_CONTEXT 構造体の InformationToReturn メンバーは、呼び出しが成功するために必要なバッファーの最小サイズに設定されている必要があります。
STATUS_CONNECTION_DISCONNECTED	接続が切断されました。
STATUS_EA_CORRUPT_ERROR	リモートサーバーから無効な拡張属性情報を受信しました。

名前	説明
STATUS_INSUFFICIENT_RESOURCES	クエリを完了するためのリソースが不足しています。
STATUS_INVALID_PARAMETER	無効なパラメーターが指定されました。
STATUS_NONEXISTENT_EA_ENTRY	ファイルオブジェクトに拡張属性がなく、ユーザーが拡張属性インデックスを指定しました。
STATUS_NOT_SUPPORTED	拡張属性はサポートされていません。
STATUS_ONLY_IF_CONNECTED	SRV_OPEN 構造体は接続されていません。
STATUS_REQUEST_ABORTED	ネットワーク要求が中止されました。

注釈

RDBSS は、[IRP_MJ_クエリ_EA](#) 要求の受信に응答して、*MRxQueryEaInfo*への呼び出しを発行します。

*MRxQueryEaInfo*を呼び出す前に、RDBSS は、*RxContext*パラメーターによって示される RX_コンテキスト構造内の次のメンバーを変更します。

情報バッファのメンバーは、i/o 要求パケットからのユーザーバッファに設定されます。このバッファは、必要に応じて、RDBSS によって既にロックされています。

LengthRemainingメンバーはirrsp->パラメーターに設定されています。quer.

Querは、UserEaListメンバーがirrsp->パラメーターに設定されています。quer. ealist.

Querのメンバーは、irrsp->のパラメーターに設定されています。これは、ユーザーが設定します。

このメンバーは、irrsp->パラメーターに設定されています。これは、querです。

Irrsp->フラグに SL_再起動_スキャンビットがある場合、RestartScanメンバーは0以外に設定されます。

Irrsp->フラグに SL__単一_エントリビットが返される場合、このメンバーは0 以外の値に設定されます。

Irrsp->フラグに SL_インデックス_指定されたビットがオンになっている場合、指定されたメンバーは0以外に設定されます。

成功した場合、*MRxQueryEaInfo*は RX_コンテキスト構造の LengthRemaininging メンバーを、返された拡張属性情報の長さに設定し、さらにfobx->OffsetOfNextEaToReturnを更新する必要があります。レプリカ. *MRxQueryEaInfo*への呼び出しが成功した場合、RDBSS は、IRP のIoStatus. **情報**メンバーをirrsp->パラメーターに設定します。また、RX_コンテキストのLengthRemainingメンバーを差し引いてください。

要件

対象プラットフォーム	Desktop
Header	Mrx.h (Mrx を含む)

関連項目

[MRxIsValidDirectory](#)

[MRxQueryDirectory](#)

[MRxQueryFileInfo](#)

[MRxQueryQuotaInfo](#)

[MRxQuerySdInfo](#)

[MRxQueryVolumeInfo](#)

[MRxSetEaInfo](#)

[MRxSetFileInfo](#)

[MRxSetFileInfoAtCleanup](#)

[MRxSetQuotaInfo](#)

[MRxSetSdInfo](#)

[MRxSetVolumeInfo](#)

MRxQueryFileInfo ルーチン

2020/04/24 • [Edit Online](#)

MRxQueryFileInfo ルーチンは [RDBSS](#) によって呼び出され、ネットワークミニリダイレクターがファイルシステムオブジェクトのファイル情報を照会するよう要求します。

構文

```
PMRX_CALLEDOWN MRxQueryFileInfo;  
  
NTSTATUS MRxQueryFileInfo(  
    _Inout_ PRX_CONTEXT RxContext  
)  
{ ... }
```

パラメーター

RxContext [in, out]

RX_コンテキスト構造体へのポインター。このパラメーターには、操作を要求している IRP が含まれています。

戻り値

MRxQueryFileInfo は正常に完了した状態_成功したか、または次のいずれかのような NTSTATUS 値を返します。

■ ■ ■ ■ ■	■ ■
STATUS_ACCESS_DENIED	呼び出し元には、この操作に対する適切なセキュリティが不足しています。
STATUS_BUFFER_OVERFLOW	ファイル情報を受信するバッファーが小さすぎます。 この戻り値は成功と見なす必要があり、可能な限り多くの有効なデータが、 <i>RxContext</i> パラメーターによって示される RX_CONTEXT 構造体の ■■■ のメンバーに返される必要があります。
STATUS_BUFFER_TOO_SMALL	バッファーが小さすぎて要求されたデータを受け取ることができません。 この値が返される場合、 <i>RxContext</i> パラメーターが指す RX_CONTEXT 構造体の InformationToReturn メンバーは、呼び出しが成功するために必要なバッファーの最小サイズに設定されている必要があります。
STATUS_INSUFFICIENT_RESOURCES	クエリを完了するためのリソースが不足しています。
STATUS_INVALID_NETWORK_RESPONSE	リモートサーバーから無効なファイル情報バッファーを受け取りました。

エラーコード	説明
STATUS_INVALID_PARAMETER	無効なパラメーターが指定されました。 この値は、RX_CONTEXT のFileinformationclassメンバーに無効な値が渡された場合に返されます。この値は、指定されたFileinformationclassメンバーがFileStreamInformation用であり、リモートファイルシステムでストリームがサポートされていない場合にも返されます。
STATUS_OBJECT_NAME_NOT_FOUND	オブジェクト名が見つかりませんでした。これはエラーコードです。

注釈

RDBSS は、[IRP_MJ_クエリ_情報](#) 要求の受信に応答して、*MRxQueryFileInfo*への呼び出しを発行します。

*MRxQueryFileInfo*を呼び出す前に、RDBSS は、*RxContext*パラメーターによって示される RX_コンテキスト構造内の次のメンバーを変更します。

Info. fileinformationclassメンバーは、irpsp->Parameters. Queryfile. fileinformationclassに設定され、要求されたファイル_情報_クラス値に設定されます。

情報バッファのメンバーは、i/o 要求パケットからのユーザーバッファに設定されます。

LengthRemainingメンバーは、irpsp->Parameters. Queryfile. の値に設定されます。

Querydirectory. fileindexメンバーはirpsp->Parameters. Querydirectory. fileindexに設定されます。

Irp->フラグに SL_再起動_スキャンビットが設定されている場合は、RestartScanメンバーが設定されます。

Irp->フラグが SL を持つ場合は、Querydirectory. returnsingleentryメンバーが設定され_単一_エントリビットセットが返されます。

UnicodeQueryTemplate がNULLで fobx >フラグに FOBX_フラグが設定されていない > 場合は、querydirectory. initialqueryメンバーが設定されます。これは、すべてのビットセット_一致しません。

成功した場合、ネットワークミニリダイレクターは、RX_コンテキスト構造のLengthRemainingメンバーを、返されたファイル情報の長さを差し引いた値に設定する必要があります。 *MRxQueryFileInfo*への呼び出しが成功した場合、RDBSS は IRP のlostatus. 情報メンバーをirpsp->パラメーターに設定します。LengthRemaining メンバーから RX_コンテキストの値を差し引いた値。

RDBSS は、SL_インデックス_指定された、Irp->フラグが設定されている要求をサポートしていません。ネットワークミニリダイレクターは、SL_インデックス_指定されているIrp->フラグが設定された状態で、*MRxQueryFileInfo*への呼び出しを受信しません。

要件

対象プラットフォーム	Desktop
Header	Mrx.h (Mrx を含む)

関連項目

[MRxIsValidDirectory](#)

MRxQueryDirectory

MRxQueryEaInfo

MRxQueryQuotaInfo

MRxQuerySdInfo

MRxQueryVolumeInfo

MRxSetEaInfo

MRxSetFileInfo

MRxSetFileInfoAtCleanup

MRxSetQuotaInfo

MRxSetSdInfo

MRxSetVolumeInfo

MRxQueryQuotaInfo ルーチン

2020/04/24 • [Edit Online](#)

MRxQueryQuotaInfo ルーチンは、ネットワークミニリダイレクターがファイルシステムオブジェクトのクォータ情報を照会するように要求するために、[RDBSS](#)によって呼び出されます。

構文

```
PMRX_CALLEDOWN MRxQueryQuotaInfo;  
  
NTSTATUS MRxQueryQuotaInfo(  
    _Inout_ PRX_CONTEXT RxContext  
)  
{ ... }
```

パラメーター

RxContext [in, out]

RX_コンテキスト構造体へのポインター。このパラメーターには、操作を要求している IRP が含まれています。

戻り値

MRxQueryQuotaInfo は正常に完了した状態_成功したか、または次のいずれかのような NTSTATUS 値を返します。

NTSTATUS	説明
STATUS_ACCESS_DENIED	呼び出し元には、この操作に対する適切なセキュリティが不足しています。
STATUS_BUFFER_OVERFLOW	クォータ情報を受信するバッファーが小さすぎます。 この戻り値は成功と見なす必要があり、可能な限り多くの有効なデータが、 <i>RxContext</i> パラメーターによって示される RX_CONTEXT 構造体の <code>InformationToReturn</code> のメンバーに返される必要があります。
STATUS_BUFFER_TOO_SMALL	バッファーが小さすぎて要求されたデータを受け取ることができません。 この値が返される場合、 <i>RxContext</i> パラメーターが指す RX_CONTEXT 構造体の <code>InformationToReturn</code> メンバーは、呼び出しが成功するために必要なバッファーの最小サイズに設定されている必要があります。
STATUS_CONNECTION_DISCONNECTED	接続が切断されました。これはエラーコードです。
STATUS_INSUFFICIENT_RESOURCES	クエリを完了するためのリソースが不足しています。これはエラーコードです。

名前	説明
STATUS_INVALID_PARAMETER	無効なパラメーターが指定されました。これはエラーコードです。
STATUS_NOT_SUPPORTED	クォータはサポートされていません。

注釈

RDBSS は、[IRP_MJ_クエリ_クォータ](#)要求の受信に応答して、*MRxQueryQuotaInfo*への呼び出しを発行します。

*MRxQueryQuotaInfo*を呼び出す前に、RDBSS は、*RxContext*パラメーターによって示される RX_コンテキスト構造内の次のメンバーを変更します。

情報バッファのメンバーは、i/o 要求パケットからのユーザーバッファに設定されます。このバッファは、必要に応じて、RDBSS によって既にロックされています。

LengthRemainingメンバーはirps->Parameters. Queryquota. Lengthに設定されています。

Queryquota. sidlistメンバーはirps->Parameters. Queryquota. sidlistに設定されています。

Queryquota. sidlistlengthメンバーは、irps->パラメーターに設定されています。Queryquota. sidlistlength。

Queryquota. startsidメンバーはirps->Parameters. Queryquota. startsidに設定されています。

Queryquota. lengthメンバーは、irps->Parameters. queryquota. lengthに設定されています。

Irps->フラグに SL_再起動_スキャンビットが設定されている場合、RestartScanメンバーは0以外に設定されます。

Irps->フラグに SL__単一_エントリビットセットが返される場合、このメンバーは0 以外の値に設定されます。

Irps->フラグに SL_インデックス_指定されたビットセットがある場合、指定されたメンバーは0以外に設定されます。

成功した場合、ネットワークミニリダイレクターは、RX_コンテキスト構造のLengthRemainingメンバーに、返されるクォータ情報の長さを設定する必要があります。*MRxQueryQuotaInfo*への呼び出しが成功した場合、RDBSS は IRP のlostatus. 情報メンバーを RX_コンテキストのLengthRemainingメンバーに設定します。

*MRxQueryQuotaInfo*の呼び出しが成功した場合、RX_コンテキスト構造のInformationToReturnメンバーは、返されるクォータ情報の長さに設定する必要があります。呼び出しが失敗した場合は、RX_コンテキストのInformationToReturnメンバーを0に設定する必要があります。

要件

対象プラットフォーム	Desktop
Header	Mrx.h (Mrx を含む)

関連項目

[MRxIsValidDirectory](#)

[MRxQueryDirectory](#)

[MRxQueryEaInfo](#)

[MRxQueryFileInfo](#)

MRxQuerySdInfo

MRxQueryVolumeInfo

MRxSetEaInfo

MRxSetFileInfo

MRxSetFileInfoAtCleanup

MRxSetQuotaInfo

MRxSetSdInfo

MRxSetVolumeInfo

MRxQuerySdInfo ルーチン

2020/04/24 • [Edit Online](#)

MRxQuerySdInfo ルーチンは、ネットワークミニリダイレクターがファイルシステムオブジェクトのセキュリティ記述子情報を照会するように要求するために、[RDBSS](#)によって呼び出されます。

構文

```
PMRX_CALLDOWN MRxQuerySdInfo;  
  
NTSTATUS MRxQuerySdInfo(  
    _Inout_ PRX_CONTEXT RxContext  
)  
{ ... }
```

パラメーター

RxContext [in, out]

RX_コンテキスト構造体へのポインター。このパラメーターには、操作を要求している IRP が含まれています。

戻り値

MRxQuerySdInfo は正常に完了した状態_成功したか、または次のいずれかのような NTSTATUS 値を返します。

NTSTATUS	説明
STATUS_ACCESS_DENIED	呼び出し元には、この操作に対する適切なセキュリティが不足しています。
STATUS_BUFFER_OVERFLOW	セキュリティ記述子の情報を受信するバッファーが小さすぎます。 この戻り値は成功と見なす必要があり、可能な限り多くの有効なデータが、 <i>RxContext</i> パラメーターによって示される RX_CONTEXT 構造体の <code>InformationToReturn</code> メンバーに返される必要があります。
STATUS_BUFFER_TOO_SMALL	バッファーが小さすぎて要求されたデータを受け取ることができません。 この値が返される場合、 <i>RxContext</i> パラメーターが指す RX_CONTEXT 構造体の <code>InformationToReturn</code> メンバーは、呼び出しが成功するために必要なバッファーの最小サイズに設定されている必要があります。
STATUS_CONNECTION_DISCONNECTED	接続が切断されました。
STATUS_INSUFFICIENT_RESOURCES	クエリを完了するためのリソースが不足しています。

エラーコード	説明
STATUS_INVALID_PARAMETER	無効なパラメーターが指定されました。
STATUS_NETWORK_ACCESS_DENIED	ネットワークアクセスが拒否されました。
STATUS_NOT_IMPLEMENTED	リモートページファイルの情報など、要求された機能は実装されていません。
STATUS_NOT_SUPPORTED	セキュリティ記述子の情報は、リモート共有ではサポートされていません。
STATUS_OBJECT_PATH_NOT_FOUND	オブジェクトのパスが見つかりませんでした。このエラーは、NTFS ストリームオブジェクトの情報が要求され、リモートファイルシステムでストリームがサポートされていない場合に返されます。
STATUS_REPARSE	シンボリックリンクを処理するには、再解析が必要です。

注釈

RDBSS は、[IRP_MJ_クエリ_セキュリティ](#) 要求の受信に応答して、*MRxQuerySdInfo* への呼び出しを発行します。

MRxQuerySdInfo を呼び出す前に、RDBSS は、*RxContext* パラメーターによって示される RX_コンテキスト構造内の次のメンバーを変更します。

Querysecurity. securityinformation メンバーは、irpsp->Parameters. Querysecurity. securityinformation に設定されています。

情報バッファのメンバーは、i/o 要求パケットからのユーザーバッファに設定されます。このバッファは、必要に応じて、RDBSS によって既にロックされています。

LengthRemaining メンバーは irpsp->Parameters. Querysecurity. Length に設定されています。

成功した場合、ネットワークミニリダイレクターは、RX_コンテキスト構造の InformationToReturn メンバーを、返されたセキュリティ情報の長さに設定する必要があります。*MRxQuerySdInfo* への呼び出しが成功した場合、RDBSS は IRP の IoStatus. 情報 メンバーを RX_コンテキストの InformationToReturn メンバーに設定します。

要件

対象プラットフォーム	Desktop
Header	Mrx.h (Mrx を含む)

関連項目

[MRxIsValidDirectory](#)

[MRxQueryDirectory](#)

[MRxQueryEaInfo](#)

MRxQueryFileInfo

MRxQueryQuotaInfo

MRxQueryVolumeInfo

MRxSetEaInfo

MRxSetFileInfo

MRxSetFileInfoAtCleanup

MRxSetQuotaInfo

MRxSetSdInfo

MRxSetVolumeInfo

MRxQueryVolumeInfo ルーチン

2020/04/24 • [Edit Online](#)

MRxQueryVolumeInfo ルーチンは、ネットワークミニリダイレクターのクエリボリューム情報を要求するために [RDBSS](#) によって呼び出されます。

構文

```
PMRX_CALLEDOWN MRxQueryVolumeInfo;  
  
NTSTATUS MRxQueryVolumeInfo(  
    _Inout_ PRX_CONTEXT RxContext  
)  
{ ... }
```

parameters

RxContext [in, out]

RX_コンテキスト構造体へのポインター。このパラメーターには、操作を要求している IRP が含まれています。

戻り値

MRxQueryVolumeInfo は正常に完了した状態_成功したか、または次のいずれかのような NTSTATUS 値を返します。

NTSTATUS	説明
STATUS_ACCESS_DENIED	呼び出し元には、この操作に対する適切なセキュリティが不足しています。
STATUS_BUFFER_OVERFLOW	ボリューム情報を受信するバッファーが小さすぎます。 この戻り値は成功と見なす必要があり、可能な限り多くの有効なデータが、 <i>RxContext</i> パラメーターによって示される RX_CONTEXT 構造体の <code>InformationToReturn</code> メンバーに返される必要があります。
STATUS_BUFFER_TOO_SMALL	バッファーが小さすぎて要求されたデータを受け取ることができません。 この値が返される場合、 <i>RxContext</i> パラメーターが指す RX_CONTEXT 構造体の <code>InformationToReturn</code> メンバーは、呼び出しが成功するために必要なバッファーの最小サイズに設定されている必要があります。
STATUS_CONNECTION_DISCONNECTED	接続が切断されました。
STATUS_INSUFFICIENT_RESOURCES	クエリを完了するためのリソースが不足しています。

エラーコード	説明
STATUS_INVALID_PARAMETER	無効なパラメーターが指定されました。
STATUS_NETWORK_NAME_DELETED	ネットワーク名が削除されました。
STATUS_NOT_IMPLEMENTED	要求された機能は実装されていません。

解説

RDBSS は、次のいずれかの場合に *MRxQueryVolumeInfo* の呼び出しを発行します。

- RDBSS は、**IRP_MJ_クエリ_ボリューム_情報** 要求を受信します。
- RDBSS は、**IRP_MJ_ファイル_システム_制御** 要求の FSCTL_LMR_GET_LINK_情報制御コードの追跡_を受信します。

IRP_MJ_クエリ_ボリューム_情報要求の場合に *MRxQueryVolumeInfo* を呼び出す前に、RDBSS は、*RXCONTEXT* が指す *RX_コンテキスト構造* 内の次のメンバーを変更します。引き

Info. *fsinformationclass* メンバーは *irpsp->Parameters. Queryvolume. fsinformationclass* に設定されています。

情報バッファのメンバーは、*Irp->AssociatedIrp* に設定されます。

LengthRemaining メンバーは *irpsp->Parameters. Queryvolume. Length* に設定されています。

IRP_MJ_クエリ_ボリューム_情報要求では、*RX_コンテキスト構造* の *postrequest* メンバーが *MRxQueryVolumeInfo* から返されたときに **TRUE** である場合、RDBSS は呼び出します。要求をポストする *RxFsdPostRequest*。この場合、*IRP_MJ_QUERY_VOLUME_INFORMATION* 要求は、*RX_コンテキスト構造* を、ファイルシステムプロセス (FSP) による処理のためにワーカーキューに渡します。_

MRxQueryVolumeInfo から返されたときに *RX_コンテキスト構造* の *Postrequest* メンバーが **FALSE** の場合、ネットワークミニリダイレクターは *rx_コンテキスト構造* の *LengthRemaining* メンバーをに設定する必要があります。返されるボリューム情報の長さ。RDBSS は、*IRP* の *loststatus. 情報* メンバーを *irpsp->パラメーター* に設定します。*QUERYVOLUME LENGTH* は *RX_CONTEXT* 構造体の *LengthRemaining* メンバーを差し引いたものです。

MRxQueryVolumeInfo への呼び出しが成功した場合、ネットワークミニリダイレクターは、*RX_コンテキスト構造* の *LengthRemaining* メンバーを、ボリューム情報の長さを差し引いた値に設定する必要があります。結果。*MRxQueryVolumeInfo* への呼び出しが成功した場合、RDBSS は *IRP* の *loststatus. 情報* メンバーを *irpsp->パラメーター* に設定します。Queryvolume Length は *RX* の *LengthRemaining* メンバーを差し引いたものです。コンテキスト構造。

IRP_MJ_QUERY_VOLUME_INFORMATION 要求の場合、Info. *FsInformationClass* メンバーが *FileFsDeviceInformation* に設定されていると、ネットワークミニリダイレクターは次の情報を *RX* に返し *@no_*RxContext** パラメーターによってポイントされる *t_6_CONTEXT* 構造体。_

情報バッファのメンバーには、*ファイル_FS_デバイス_情報* 構造体が含まれています

情報の *Buffer. 特性* メンバーは、ボリュームの特性に設定されます。これには、オプションの1つとして、*ファイル_リモート_デバイス* が含まれている必要があります。

(*Devicetype* メンバーは、関連付けられている *NET* のルート構造の (*devicetype* メンバーに設定されます。関連付けられている *NET_ルート* の *Type* メンバーが *NET_ROOT_PIPE* の場合、(*devicetype* メンバーは *_パイプ* という名前の *ファイル_デバイス_* に設定されます。

IRP_MJ_QUERY_VOLUME_INFORMATION 要求の場合、Info. *FsInformationClass* メンバー

がFileFsVolumeInformationに設定されていると、ネットワークミニリダイレクターは次の情報を RX に返し @no_*RxContext*パラメーターによってポイントされる t_6_CONTEXT 構造体。_

情報バッファのメンバーには、ファイル_FS_ボリューム_情報構造体が含まれています。

情報バッファのメンバーは、関連付けられている NET_のルート構造のvolumeinfoメンバーに設定されます。

LengthRemainingメンバーは、関連付けられている NET_のルート構造のVolumeInfoLengthメンバーに設定されます。

[IRP_MJ_ファイル_システム_コントロール](#)からのMRxQueryVolumeInfo呼び出しは、リンク追跡情報の要求です。IRP_MJ_ファイル_システム_コントロールに対してMRxQueryVolumeInfoを呼び出す前に、RDBSS は、RxContextIパラメーターが指す RX_CONTEXT 構造体の次のメンバーを変更します。

FileFsObjectIdInformationに設定されたInfo. FsInformationClassメンバー。

情報バッファのメンバーは、ファイル_FS_OBJECTID_情報構造体に設定されます。

LengthRemainingメンバーはsizeof(FILE_FS_OBJECTID_INFORMATION) に設定されています。

この場合、IRP_MJ_FILE_SYSTEM_CONTROL 要求では、IRP のAssociatedIrpメンバーがリンクをポイントして_情報構造を追跡します。_

要求が IRP_MJ_ファイルとして開始された場合_システム_コントロールに対して、戻り値が STATUS_SUCCESS または STATUS_BUFFER_OVERFLOW であることを示します。RDBSS は、ファイルのobjectidメンバー_FS_OBJECTID_情報構造体にコピーします。これは、RX_コンテキスト>構造の情報バッファのメンバーに渡されます。DiskParameters。FCB 構造体のメンバーであり、IRP のAssociatedIrpメンバーになります。MRxQueryVolumeInfoへの呼び出しが成功した場合、RDBSS はリンクの型メンバーを_追跡_情報構造体に設定します。FCB 構造体のnetroot>フラグのメンバーが netroot_フラグ_DFS_認識_netroot ビットセットを持つ場合、型のメンバーは RDBSS によってdfslinktrackinginformationに設定されます。FCB 構造体のnetroot>フラグのメンバーに netroot_フラグが設定されていない場合_DFS_認識_netroot ビットセットの場合、型のメンバーは RDBSS によってNtfsLinkTrackingInformationに設定されます。成功した場合、RDBSS は、IRP のlostatus. 情報メンバーを、追跡_情報構造_リンクのサイズに設定します。

前提条件

ターゲットプラットフォーム	デスクトップ
ヘッダー	Mrx.h (Mrx を含む)

関連項目

[MRxIsValidDirectory](#)

[MRxQueryDirectory](#)

[MRxQueryEaInfo](#)

[MRxQueryFileInfo](#)

[MRxQueryQuotaInfo](#)

[MRxQuerySdInfo](#)

[MRxSetEaInfo](#)

[MRxSetFileInfo](#)

[MRxSetFileInfoAtCleanup](#)

MRxSetQuotaInfo

MRxSetSdInfo

MRxSetVolumeInfo

RxFsdPostRequest

MRxSetEaInfo ルーチン

2020/04/24 • [Edit Online](#)

MRxSetEaInfo ルーチンは、ネットワークミニリダイレクターがファイルシステムオブジェクトに拡張属性情報を設定するように要求するために、[RDBSS](#)によって呼び出されます。

構文

```
PMRX_CALLEDOWN MRxSetEaInfo;  
  
NTSTATUS MRxSetEaInfo(  
    _Inout_ PRX_CONTEXT RxContext  
)  
{ ... }
```

parameters

RxContext [in, out]

RX_コンテキスト構造体へのポインター。このパラメーターには、操作を要求している IPR が含まれています。

戻り値

MRxSetEaInfo は正常に完了した状態_成功したか、または次のいずれかのような NTSTATUS 値を返します。

NTSTATUS	説明
STATUS_ACCESS_DENIED	呼び出し元には、この操作に対する適切なセキュリティが不足しています。
STATUS_EA_TOO_LARGE	渡された拡張属性情報が、リモート共有でサポートされているサイズを超えています。
STATUS_FILE_CLOSED	SRV_OPEN 構造体が閉じられました。
STATUS_INSUFFICIENT_RESOURCES	クエリを完了するためのリソースが不足しています。
STATUS_INVALID_PARAMETER	無効なパラメーターが指定されました。
STATUS_NETWORK_ACCESS_DENIED	ネットワークアクセスが拒否されました。このエラーは、ネットワークミニリダイレクターが読み取り専用共有に拡張属性を設定するように要求された場合に返されます。
STATUS_NOT_IMPLEMENTED	リモートページファイルの拡張情報の設定など、要求された機能は実装されていません。

エラーコード	説明
STATUS_NOT_SUPPORTED	拡張属性はサポートされていません。
STATUS_OBJECT_NAME_NOT_FOUND	オブジェクト名が見つかりませんでした。このエラーは、ネットワークミニリダイレクターがファイルに拡張属性を設定するように要求したが、ファイルが存在しない場合に返されることがあります。
STATUS_OBJECT_PATH_NOT_FOUND	オブジェクトのパスが見つかりませんでした。このエラーは、NTFSストリームオブジェクトが渡され、リモートファイルシステムでストリームがサポートされていない場合に返されます。
STATUS_ONLY_IF_CONNECTED	SRV_OPEN 構造体は接続されていません。
STATUS_REPARSE	シンボリックリンクを処理するには、再解析が必要です。

解説

RDBSS は、*MRxSetEaInfo*への呼び出しを発行し、**[_EA 要求の IRP_MJ_セット](#)**を受信します。

*MRxSetEaInfo*を呼び出す前に、RDBSS は、*RxContextI*パラメーターによって示される RX_コンテキスト構造内の次のメンバーを変更します。

情報バッファのメンバーは、i/o 要求パケットからのユーザーバッファに設定されます。このバッファは、必要に応じて、RDBSS によって既にロックされています。

LengthRemainingメンバーはirpsp->パラメーターに設定されています。quer.

前提条件

ターゲットプラットフォーム	デスクトップ
ヘッダー	Mrx .h (Mrx を含む)

関連項目

[MRxIsValidDirectory](#)

[MRxQueryDirectory](#)

[MRxQueryEaInfo](#)

[MRxQueryFileInfo](#)

[MRxQueryQuotaInfo](#)

[MRxQuerySdInfo](#)

[MRxQueryVolumeInfo](#)

[MRxSetFileInfo](#)

MRxSetFileInfoAtCleanup

MRxSetQuotaInfo

MRxSetSdInfo

MRxSetVolumeInfo

MRxSetFileInfo ルーチン

2020/04/24 • [Edit Online](#)

MRxSetFileInfo ルーチンは、ネットワークミニリダイレクターがファイルシステムオブジェクトのファイル情報を設定するように要求するために、[RDBSS](#)によって呼び出されます。

構文

```
PMRX_CALLEDOWN MRxSetFileInfo;  
  
NTSTATUS MRxSetFileInfo(  
    _Inout_ PRX_CONTEXT RxContext  
)  
{ ... }
```

parameters

RxContext [in, out]

RX_コンテキスト構造体へのポインター。このパラメーターには、操作を要求している IRP が含まれています。

戻り値

MRxSetFileInfo は正常に完了した状態_成功したか、または次のいずれかのような NTSTATUS 値を返します。

NTSTATUS	説明
STATUS_ACCESS_DENIED	呼び出し元には、この操作に対する適切なセキュリティが不足しています。
STATUS_INSUFFICIENT_RESOURCES	クエリを完了するためのリソースが不足しています。
STATUS_INVALID_PARAMETER	無効なパラメーターが指定されました。
STATUS_NETWORK_ACCESS_DENIED	ネットワークアクセスが拒否されました。このエラーは、ネットワークミニリダイレクターが読み取り専用共有のファイル情報を設定するように要求された場合に返されます。
STATUS_NOT_IMPLEMENTED	リモートページファイルのファイル情報の設定など、要求された機能は実装されていません。
STATUS_OBJECT_NAME_NOT_FOUND	オブジェクト名が見つかりませんでした。このエラーは、ネットワークミニリダイレクターがファイルのファイル情報を設定するように要求したが、ファイルが存在しない場合に返されることがあります。

NTSTATUS	説明
STATUS_OBJECT_PATH_NOT_FOUND	オブジェクトのパスが見つかりませんでした。このエラーは、NTFS ストリームオブジェクトが渡され、リモートファイルシステムでストリームがサポートされていない場合に返されます。
STATUS_ONLY_IF_CONNECTED	SRV_OPEN 構造体は接続されていません。
STATUS_REPARSE	シンボリックリンクを処理するには、再解析が必要です。

解説

RDBSS は、[IRP_MJ_SET_情報](#) 要求の受信に応答して、*MRxSetFileInfo* への呼び出しを発行します。

MRxSetFileInfo を呼び出す前に、RDBSS は、*RxContext* パラメーターによって示される RX_コンテキスト構造内の次のメンバーを変更します。

Info. fileinformationclass メンバーは、`irpsp->Parameters. Setfile. fileinformationclass` に設定され、指定された **ファイル_情報_クラス** 値に設定されます。

情報バッファ のメンバーは、`Irp->AssociatedIrp` に設定されます。

情報の長さ のメンバーは `irpsp->Parameters. setfile. length` に設定されます。

前提条件

ターゲットプラットフォーム	デスクトップ
ヘッダー	Mrx.h (Mrx を含む)

関連項目

[MRxIsValidDirectory](#)

[MRxQueryDirectory](#)

[MRxQueryEaInfo](#)

[MRxQueryFileInfo](#)

[MRxQueryQuotaInfo](#)

[MRxQuerySdInfo](#)

[MRxQueryVolumeInfo](#)

[MRxSetEaInfo](#)

[MRxSetFileInfoAtCleanup](#)

[MRxSetQuotaInfo](#)

[MRxSetSdInfo](#)

[MRxSetVolumeInfo](#)

MRxSetFileInfoAtCleanup ルーチン

2020/04/24 • [Edit Online](#)

MRxSetFileInfoAtCleanup ルーチンは、クリーンアップ時にネットワークミニリダイレクターがファイルシステムオブジェクトにファイル情報を設定するよう要求するために、[RDBSS](#)によって呼び出されます。

構文

```
PMRX_CALLDOWN MRxSetFileInfoAtCleanup;  
  
NTSTATUS MRxSetFileInfoAtCleanup(  
    _Inout_ PRX_CONTEXT RxContext  
)  
{ ... }
```

parameters

RxContext [in, out]

RX_コンテキスト構造体へのポインター。このパラメーターには、操作を要求している IRP が含まれています。

戻り値

MRxSetFileInfoAtCleanup は正常に完了した場合、または適切な NTSTATUS 値を_状態を返します。

解説

RDBSS は、ファイルオブジェクトへの最後のハンドルが閉じられたときに、クリーンアップ中に *MRxSetFileInfoAtCleanup* への呼び出しを発行します。これは、ファイルオブジェクトへの最後の参照が削除されたときに呼び出される close 操作とは異なります。

MRxSetFileInfoAtCleanup は、ファイルのタイムスタンプまたはファイルのサイズが変更された場合に、RDBSS によって呼び出されます。*MRxSetFileInfoAtCleanup* by RDBSS の呼び出しは、これらの変更ごとに個別に行われます。ファイルサイズとタイムスタンプの両方が変更された場合、RDBSS は *MRxSetFileInfoAtCleanup* を2回呼び出します。

MRxSetFileInfoAtCleanup を呼び出す前に、RDBSS は、ファイルのタイムスタンプが変更された場合に *RxContext* パラメーターが指す RX_コンテキスト構造内の次のメンバーを変更します。

Info. FileInformationClass メンバーは、FILE_INFORMATION_CLASS のファイル_情報_クラス値に設定されます。

情報バッファー のメンバーは、スタックに割り当てられた基本_情報構造_ファイルに設定されます。

情報の長さ のメンバーは、基本_情報の構造_、ファイルの sizeof に設定されます。

MRxSetFileInfoAtCleanup を呼び出す前に、RDBSS は、ファイルのサイズが変更された場合に *RxContext* パラメーターが指す RX_コンテキスト構造内の次のメンバーを変更します。

FileEndOfFileInformation の_クラス値のファイル_情報に設定されます。

情報バッファー のメンバーは、スタックに割り当てられている_ファイル_情報構造の末尾__ファイルに設定されます。

Info メンバーは sizeof (file_END__ファイル_情報) に設定されます。

RDBSS は、*MRxSetFileInfoAtCleanup* からの戻り値を無視します。

ネットワークミニリダイレクターは、このルーチンで何も実行しないことを選択し、状態_SUCCESS を返します。ファイルサイズまた

はタイムスタンプに対する変更は、クリーンアップ操作中に処理されます。

前提条件

ターゲットプラットフォーム	デスクトップ
ヘッダー	Mrx.h (Mrx を含む)

関連項目

[MRxIsValidDirectory](#)

[MRxQueryDirectory](#)

[MRxQueryEaInfo](#)

[MRxQueryFileInfo](#)

[MRxQueryQuotaInfo](#)

[MRxQuerySdInfo](#)

[MRxQueryVolumeInfo](#)

[MRxSetEaInfo](#)

[MRxSetFileInfo](#)

[MRxSetQuotaInfo](#)

[MRxSetSdInfo](#)

[MRxSetVolumeInfo](#)

MRxSetQuotaInfo ルーチン

2020/04/24 • [Edit Online](#)

MRxSetQuotaInfo ルーチンは、ネットワークミニリダイレクターがファイルシステムオブジェクトのクォータ情報を設定するように要求するために、[RDBSS](#)によって呼び出されます。

構文

```
PMRX_CALLDOWN MRxSetQuotaInfo;  
  
NTSTATUS MRxSetQuotaInfo(  
    _Inout_ PRX_CONTEXT RxContext  
)  
{ ... }
```

パラメーター

RxContext [in, out]

RX_コンテキスト構造体へのポインター。このパラメーターには、操作を要求している IRP が含まれています。

戻り値

MRxSetQuotaInfo は正常に完了した状態_成功したか、または次のいずれかのような NTSTATUS 値を返します。

NTSTATUS	説明
STATUS_ACCESS_DENIED	呼び出し元には、この操作に対する適切なセキュリティが不足しています。
STATUS_CONNECTION_DISCONNECTED	接続が切断されました。
STATUS_INSUFFICIENT_RESOURCES	クエリを完了するためのリソースが不足しています。
STATUS_INVALID_PARAMETER	無効なパラメーターが指定されました。
STATUS_NOT_SUPPORTED	クォータはサポートされていません。

注釈

RDBSS は、[IRP_MJ_SET_クォータ](#) 要求の受信にตอบสนองして、*MRxSetQuotaInfo* への呼び出しを発行します。

MRxSetQuotaInfo を呼び出す前に、RDBSS は、*RxContext* パラメーターによって示される RX_コンテキスト構造内の次のメンバーを変更します。

情報バッファのメンバーは、i/o 要求パケットからのユーザーバッファに設定されます。このバッファは、必要に応じて、RDBSS によって既にロックされています。

LengthRemainingメンバーはirps->Parameters. Setquota. Lengthに設定されています。

要件

対象プラットフォーム	Desktop
Header	Mrx.h (Mrx を含む)

関連項目

[MRxIsValidDirectory](#)

[MRxQueryDirectory](#)

[MRxQueryEaInfo](#)

[MRxQueryFileInfo](#)

[MRxQueryQuotaInfo](#)

[MRxQuerySdInfo](#)

[MRxQueryVolumeInfo](#)

[MRxSetEaInfo](#)

[MRxSetFileInfo](#)

[MRxSetFileInfoAtCleanup](#)

[MRxSetSdInfo](#)

[MRxSetVolumeInfo](#)

MRxSetSdInfo ルーチン

2020/04/24 • [Edit Online](#)

MRxSetSdInfo ルーチンは、ネットワークミニリダイレクターがファイルシステムオブジェクトのセキュリティ記述子情報を設定するように要求するために、[RDBSS](#)によって呼び出されます。

構文

```
PMRX_CALLEDOWN MRxSetSdInfo;  
  
NTSTATUS MRxSetSdInfo(  
    _Inout_ PRX_CONTEXT RxContext  
)  
{ ... }
```

パラメーター

RxContext [in、out]

RX_コンテキスト構造体へのポインター。このパラメーターには、操作を要求している IRP が含まれています。

戻り値

MRxSetSdInfo は正常に完了した状態_成功したか、または次のいずれかのような NTSTATUS 値を返します。

NTSTATUS	説明
STATUS_ACCESS_DENIED	呼び出し元には、この操作に対する適切なセキュリティが不足しています。
STATUS_CONNECTION_DISCONNECTED	接続が切断されました。
STATUS_INSUFFICIENT_RESOURCES	クエリを完了するためのリソースが不足しています。
STATUS_INVALID_PARAMETER	無効なパラメーターが指定されました。
STATUS_NETWORK_ACCESS_DENIED	ネットワークアクセスが拒否されました。
STATUS_NOT_IMPLEMENTED	リモートページファイルのセキュリティ情報の設定など、要求された機能は実装されていません。
STATUS_NOT_SUPPORTED	セキュリティ記述子の情報は、リモート共有ではサポートされていません。

エラーコード	説明
STATUS_OBJECT_PATH_NOT_FOUND	オブジェクトのパスが見つかりませんでした。このエラーは、NTFS ストリームオブジェクトのセキュリティ情報を設定するように要求され、リモートファイルシステムでストリームがサポートされていない場合に返されることがあります。
STATUS_REPARSE	シンボリックリンクを処理するには、再解析が必要です。

注釈

RDBSS は、[IRP_MJ_SET_SECURITY](#) 要求の受信に応答して、*MRxSetSdInfo* への呼び出しを発行します。

MRxSetSdInfo を呼び出す前に、RDBSS は、*RxContext* パラメーターによって示される RX_コンテキスト構造内の次のメンバーを変更します。

Setsecurity. securityinformation メンバーは、irps->Parameters. Setsecurity. securityinformation に設定されています。

Setsecurity. securitydescriptor メンバーは、irps->Parameters. setsecurity. securitydescriptor に設定されています。

要件

対象プラットフォーム	Desktop
Header	Mrx.h (Mrx を含む)

関連項目

[MRxIsValidDirectory](#)

[MRxQueryDirectory](#)

[MRxQueryEaInfo](#)

[MRxQueryFileInfo](#)

[MRxQueryQuotaInfo](#)

[MRxQuerySdInfo](#)

[MRxQueryVolumeInfo](#)

[MRxSetEaInfo](#)

[MRxSetFileInfo](#)

[MRxSetFileInfoAtCleanup](#)

[MRxSetQuotaInfo](#)

[MRxSetVolumeInfo](#)

MRxSetVolumeInfo ルーチン

2020/04/24 • [Edit Online](#)

MRxSetVolumeInfo ルーチンは、ネットワークミニリダイレクターがボリューム情報を設定するように要求するために、[RDBSS](#)によって呼び出されます。

構文

```
PMRX_CALLDOWN MRxSetVolumeInfo;  
  
NTSTATUS MRxSetVolumeInfo(  
    _Inout_ PRX_CONTEXT RxContext  
)  
{ ... }
```

パラメーター

RxContext [in、out]

RX_コンテキスト構造体へのポインター。このパラメーターには、操作を要求している IRP が含まれています。

戻り値

MRxSetVolumeInfo は正常に完了した状態_成功したか、または次のいずれかのような NTSTATUS 値を返します。

■ ■ ■ ■ ■	■ ■
STATUS_ACCESS_DENIED	呼び出し元には、この操作に対する適切なセキュリティが不足しています。
STATUS_CONNECTION_DISCONNECTED	接続が切断されました。
STATUS_INSUFFICIENT_RESOURCES	クエリを完了するためのリソースが不足しています。
STATUS_INVALID_PARAMETER	無効なパラメーターが指定されました。
STATUS_NETWORK_NAME_DELETED	ネットワーク名が削除されました。
STATUS_NOT_IMPLEMENTED	要求された機能は実装されていません。
STATUS_NOT_SUPPORTED	この要求は、リモート共有ではサポートされていません。

注釈

RDBSS は、[IRP_MJ_SET_VOLUME_INFORMATION](#) 要求の受信に応答して、*MRxSetVolumeInfo* への呼び出しを発行します。

*MRxSetVolumeInfo*を呼び出す前に、RDBSS は、*RxContextI*パラメーターによって示される RX_コンテキスト構造内の次のメンバーを変更します。

Info. fsinformationclassメンバーはirps->Parameters. setsinformationclassに設定されています。

情報バッファのメンバーは、Irp->AssociatedIrpに設定されます。

LengthRemainingメンバーはirps->Parameters. Setvolume. Lengthに設定されています。

要件

対象プラットフォーム	Desktop
Header	Mrx.h (Mrx を含む)

関連項目

[MRxIsValidDirectory](#)

[MRxQueryDirectory](#)

[MRxQueryEaInfo](#)

[MRxQueryFileInfo](#)

[MRxQueryQuotaInfo](#)

[MRxQuerySdInfo](#)

[MRxQueryVolumeInfo](#)

[MRxSetEaInfo](#)

[MRxSetFileInfo](#)

[MRxSetFileInfoAtCleanup](#)

[MRxSetQuotaInfo](#)

[MRxSetSdInfo](#)

MRxShouldTryToCollapseThisOpen ルーチン

2020/04/24 • [Edit Online](#)

MRxShouldTryToCollapseThisOpen ルーチンは [RDBSS](#) によって呼び出され、ネットワークミニリダイレクターが、開いている要求を既存のファイルシステムオブジェクトに対して試行するかどうかを示すことを要求します。

構文

```
PMRX_CALLDOWN MRxShouldTryToCollapseThisOpen;  
  
NTSTATUS MRxShouldTryToCollapseThisOpen(  
    _Inout_ PRX_CONTEXT RxContext  
)  
{ ... }
```

パラメーター

RxContext [in, out]

RX_コンテキスト構造体へのポインター。このパラメーターには、操作を要求している IRP が含まれています。

戻り値

MRxShouldTryToCollapseThisOpen は正常に完了した状態_成功したか、次のような適切な NTSTATUS 値を返します。

NTSTATUS	説明
STATUS_MORE_PROCESSING_REQUIRED	ネットワークミニリダイレクターはこの値を返して、開いている要求の折りたたみを無効にします。

注釈

MRxShouldTryToCollapseThisOpen は、開いている要求を折りたたむ必要があるかどうかを判断するために呼び出されます。

MRxShouldTryToCollapseThisOpen を呼び出す前に、RDBSS は、*RxContext* パラメーターによって示される RX_コンテキスト構造内の次のメンバーを変更します。

PRelevantSrvOpen メンバーは、開いている SRV_ に設定されます。

MRxShouldTryToCollapseThisOpen の呼び出しは、ディレクトリの変更通知要求である可能性があります。そのため、ネットワークミニリダイレクターは、変更通知が正しく機能するように、開いている要求を折りたたむことができない可能性があります。

RX_コンテキスト構造の RDBSS メンバーに、_BACKUP_ インテントオプションまたはファイル_削除用の_を開く_が含まれている場合は、[折りたたみを許可しない] が開き__閉じるオプションセットです。

要件

対象プラットフォーム	Desktop

Header	Mrx.h (Mrx を含む)

関連項目

[MRxAreFilesAliased](#)

[MRxCleanupFobx](#)

[MRxCloseSrvOpen](#)

[MRxCollapseOpen](#)

[MRxCreate](#)

[MRxDeallocateForFcb](#)

[MRxDeallocateForFobx](#)

[MRxExtendForCache](#)

[MRxExtendForNonCache](#)

[MRxFlush](#)

[MRxForceClosed](#)

[MRxIsLockRealizable](#)

[MRxTruncate](#)

[MRxZeroExtend](#)

MRxStop ルーチン

2020/04/24 • [Edit Online](#)

MRxStop ルーチンは、ネットワークミニリダイレクターを停止するために [RDBSS](#) によって呼び出されます。

構文

```
PMRX_CALLDOWN_CTX MRxStop;  
  
NTSTATUS MRxStop(  
    _Inout_ PRX_CONTEXT          RxContext,  
    _Inout_ PRDBSS_DEVICE_OBJECT RxDeviceObject  
)  
{ ... }
```

パラメーター

RxContext [in, out]

RX_コンテキスト構造体へのポインター。このパラメーターには、ネットワークミニリダイレクターの停止を要求した IRP が含まれます。

RxDeviceObject [in, out]

このネットワークミニリダイレクターの RDBSS_デバイス_オブジェクト構造へのポインター。

戻り値

MRxStop は正常に完了した状態_成功したか、または次のいずれかのような NTSTATUS 値を返します。

STATUS_REDIRECTOR_HAS_OPEN_HANDLES	ネットワークミニリダイレクターに開いているハンドルがあるため、この時点では停止できません。
STATUS_REDIRECTOR_NOT_STARTED	ネットワークミニリダイレクターが開始されませんでした。

注釈

MRxStop は、RDBSS の観点からネットワークミニリダイレクターを停止して初期化前します。ネットワークミニリダイレクターを停止すると、メモリ割り当てやその他のシステムリソースが解放される可能性があります。

MRxStop を呼び出す前に、RDBSS によって次の値が変更されます。

RxContext によってポイントされる RX_コンテキスト構造の **MajorFunction** メンバーは、IRP のメジャー関数に設定されます。

RxContext によってポイントされている RX_コンテキスト構造の **LowIocontext** のコードメンバーは、ネットワークミニリダイレクターを停止するために使用された FSTCL 要求の場合、IRP の FsCtl コードに設定されます。

RxDeviceObject によってポイントされている RDBSS_デバイス_オブジェクト構造の **startstopcontext. State** メンバーは、_の進行状況で RDBSS_STOP_ に設定されます。

RxDeviceObject によってポイントされるオブジェクト構造の RDBSS_ デバイスの **Startstopcontext** の **pstopcontext** メンバーは、*RxContext* パラメーターに設定されます。

*MRxStop*は、[RXSTOPMINIRDR](#)ルーチンから RDBSS によって呼び出されます。

*MRxStop*が STATUS_SUCCESS を返した場合、ルーチンは成功しました。その他の戻り値は、ネットワークミニリダイレクターの停止中にエラーが発生したことを示します。

*MRxStop*によって STATUS_SUCCESS が返された場合、RDBSS は RDBSS の状態を_RDBSS に設定します。この状態は、*RxDeviceObject*によってポイントされるオブジェクト構造_RDBSS_デバイスのstartstopcontext. stateメンバーに格納されます。

ネットワークミニリダイレクターは、通常、ネットワークミニリダイレクターが開始されているかどうかを示す内部変数を保持します。たとえば、ネットワークミニリダイレクターは、停止、開始、および開始操作または停止操作が進行中であることを追跡できます。

要件

対象プラットフォーム	Desktop
Header	Mrx.h (Mrx を含む)

関連項目

[MRxDevFcbXXXControlFile](#)

[MrxStart](#)

[RxStopMinirdr](#)

MRxTruncate ルーチン

2020/04/24 • [Edit Online](#)

MRxTruncate ルーチンは、ネットワークミニリダイレクターがファイルシステムオブジェクトの内容を切り捨てることを要求するために、RDBSS によって呼び出されます。

構文

```
PMRX_CALLDOWN MRxTruncate;  
  
NTSTATUS MRxTruncate(  
    _Inout_ PRX_CONTEXT RxContext  
)  
{ ... }
```

パラメーター

RxContext [in, out]

RX_コンテキスト構造体へのポインター。このパラメーターには、操作を要求している IRP が含まれています。

戻り値

MRxTruncate は正常に完了した状態_成功したか、または次のいずれかのような NTSTATUS 値を返します。

■ ■ ■ ■	■
STATUS_NOT_IMPLEMENTED	このルーチンは実装されていません。

注釈

MRxTruncate は、次の両方の条件に該当する場合にクリーンアップ操作の一部として呼び出されます。

- ファイルオブジェクトは、ディスクファイルまたはディレクトリに対応します。
- これは最後のクリーンアップ呼び出しであり、ファイルオブジェクトに切り捨て用にマークが付けられています。

ファイルオブジェクトは、FCB 構造体の `fcbstate` メンバーに `FCB_STATE` がある場合は切り捨て対象としてマークされ、`_CLOSE` ビットセットでは切り捨て__ます。RDBSS は、後でキャッシュマップを初期化解除します。

MRxTruncate の呼び出しの後に、クリーンアップ操作の一部として [MRxCleanupFobx](#) が呼び出されます。

RDBSS は、*MRxTruncate* からの戻り値を無視します。

要件

対象プラットフォーム	Desktop
Header	Mrx.h (Mrx を含む)

関連項目

[MRxAreFilesAliased](#)

[MRxCleanupFobx](#)

[MRxCloseSrvOpen](#)

[MRxCollapseOpen](#)

[MRxCreate](#)

[MRxDeallocateForFcb](#)

[MRxDeallocateForFobx](#)

[MRxExtendForCache](#)

[MRxExtendForNonCache](#)

[MRxFlush](#)

[MRxForceClosed](#)

[MRxIsLockRealizable](#)

[MRxShouldTryToCollapseThisOpen](#)

[MRxZeroExtend](#)

MRxZeroExtend ルーチン

2020/04/24 • [Edit Online](#)

MRxZeroExtend ルーチンは、ネットワークミニリダイレクターがファイルシステムオブジェクトの内容を切り捨てることを要求するために、[RDBSS](#)によって呼び出されます。

構文

```
PMRX_CALLDOWN MRxZeroExtend;  
  
NTSTATUS MRxZeroExtend(  
    _Inout_ PRX_CONTEXT RxContext  
)  
{ ... }
```

パラメーター

RxContext [in, out]

RX_コンテキスト構造体へのポインター。このパラメーターには、操作を要求している IRP が含まれています。

戻り値

MRxZeroExtend は正常に完了した状態_成功したか、次のような適切な NTSTATUS 値を返します。

NTSTATUS	説明
STATUS_NOT_IMPLEMENTED	このルーチンは実装されていません。

注釈

ファイルオブジェクトが削除対象としてマークされておらず、ファイルオブジェクトがページングファイルでない場合は、クリーンアップ操作の一部として *MRxZeroExtend* が呼び出されます。 *MRxZeroExtend* は、有効なデータ長とファイルサイズの間部分がゼロ拡張であることを確認するために呼び出されます。 *MRxZeroExtend* を呼び出した後、RDBSS は fcb 構造体の構造体の QuadPart メンバーを、Fcb 構造体の QuadPart メンバーと等しい値に設定します。

MRxZeroExtend の呼び出しの後に、クリーンアップ操作の一部として [MRxCleanupFobx](#) が呼び出されます。

RDBSS は、*MRxZeroExtend* からの戻り値を無視します。

要件

対象プラットフォーム	Desktop
Header	Mrx.h (Mrx を含む)

関連項目

MRxAreFilesAliased

MRxCleanupFobx

MRxCloseSrvOpen

MRxCollapseOpen

MRxCreate

MRxDeallocateForFcb

MRxDeallocateForFobx

MRxExtendForCache

MRxExtendForNonCache

MRxFlush

MRxForceClosed

MRxIsLockRealizable

MRxShouldTryToCollapseThisOpen

MRxTruncate

RxNewMapUserBuffer 関数

2020/04/24 • [Edit Online](#)

RxNewMapUserBufferは、低 i/o に使用されるユーザーバッファアドレスを返します。

構文

```
PVOID RxNewMapUserBuffer(  
    _In_ PRX_CONTEXT RxContext  
);
```

パラメーター

] の *RxContext* [

この要求の RX_コンテキスト構造体へのポインター。

戻り値

RxNewMapUserBufferは、成功した場合はマップされたアドレスポインターを返し、失敗した場合はNULLを返します。

注釈

MDL が存在する場合は、MDL によってユーザーバッファが記述され、MDL のシステムアドレスがRxNewMapUserBufferによって返されることが前提となります。それ以外の場合は、ユーザーバッファはRxNewMapUserBufferによって直接返されます。

RxNewMapUserBuffer ルーチンは、*RxContext*変数の>MdladdressメンバーがNULLである-かどうかを確認し、UserBuffer を>-返します。この場合の*RxContext*変数のメンバー。->MdladdressメンバーがNULLでない場合、RxNewMapUserBufferは IRP から MDL を返すために[MmGetSystemAddressForMdlSafe](#)を呼び出します。

RxNewMapUserBufferルーチンは、windows XP と windows 2000 でのみ使用できます。

要件

対象プラットフォーム	Desktop
バージョン	RxNewMapUserBuffer ルーチンは、Windows XP および Windows 2000 でのみ使用できます。
Header	Rxprocs (Rxcontx または Rxprocs を含む)
IRQL	<= APC_LEVEL

関連項目

[MmGetSystemAddressForMdlSafe](#)

RxLowIoCompletion

RxLowIoGetBufferAddress

RxMapSystemBuffer

RX_コンテキスト

_RxAllocatePoolWithTag 関数

2020/04/24 • [Edit Online](#)

`_RxAllocatePoolWithTag`は、メモリ trashing のインスタンスをキャッチするために使用できる、ブロックの先頭に4バイトのタグがあるプールからメモリを割り当てます。

構文

```
VOID* _RxAllocatePoolWithTag(  
    ULONG Type,  
    ULONG Size,  
    ULONG Tag,  
    PSZ   FileName,  
    ULONG LineNumber  
);
```

パラメーター

型

割り当てられるプールの種類。このパラメーターには、プール_型の次の列挙値のいずれかを指定できます。

NonPagedPool

非ページングは、任意の IRQL からアクセスできるシステムメモリを使用します。NonPagedPoolメモリは不足しているリソースであり、ドライバーは必要なときにのみ割り当てます。システムで割り当てることができるのは、ページ_サイズの倍数でNonPagedPoolからページ_サイズより大きいバッファーのみです。ページ_サイズより大きいバッファーの要求に対して、複数の非ページングメモリを_サイズではありません。

PagedPool

IRQL < ディスパッチ_レベルでのみ割り当ておよびアクセスできるページング可能なシステムメモリ。

サイズ

割り当てるメモリブロックのサイズ (バイト単位)。

タグ

割り当てられたバッファーをマークするために使用される4バイトのタグ。タグの使用の詳細については、「[ExAllocatePoolWithTag](#)」を参照してください。タグ内の各文字の ASCII 値は、0 ~ 127 の範囲で指定する必要があります。

ファイル名

メモリ割り当てが発生したソースファイル名へのポインター。このパラメーターは現在使用されていません。

LineNumber

メモリ割り当てが発生したソースファイル内の行番号。このパラメーターは現在使用されていません。

戻り値

`RxAllocatePoolWithTag`は、要求を満たすために空きプール内のメモリが不足している場合にNULLを返します。それ以外の場合、ルーチンは割り当てられたメモリへのポインターを返します。

注釈

このルーチンを直接使用するのではなく、`RxAllocatePoolWithTag`マクロを呼び出すことをお勧めします。リテールビルドでは、このマクロは[ExAllocatePoolWithTag](#)を呼び出すように定義されています。チェックを行ったビルドでは、このマクロは

`_RxAllocatePoolWithTag`を呼び出すように定義されています。

`_RxAllocatePoolWithTag`ルーチンは、`lowpoolpriority` に設定された優先度 (要求の重要度) を使用して `ExAllocatePoolWithTagPriority` を呼び出します。リソースが不足している場合、システムは `LowPoolPriority` の要求に失敗する可能性があります。このルーチンを使用する場合は、ドライバーを割り当てエラーから回復するように準備する必要があります。

システムがページの `_サイズ` 以上のプールメモリからバッファを割り当てた場合、バッファはページの境界に配置されます。ページ `_サイズ` より小さいメモリ要求は、必ずしもページの境界に沿って配置されるわけではありませんが、常に1つのページに収まるように、8バイトの境界に合わせて配置されます。ページサイズの倍数ではないブロックを要求する正常に割り当てられた場合は、最後に割り当てられたページの未使用のバイトがすべて無駄になります。これは、ページ `_サイズ` の倍数ではない `_ブロック` を要求します。

システムは、割り当てられたメモリに `pool` タグを関連付けます。WinDbg などのプログラミングツールでは、割り当てられた各バッファに関連付けられたプールタグを表示できます。タグの値は通常、逆順で表示されます。たとえば、呼び出し元が "Fred" をタグとして渡すと、メモリがダンプされた場合、またはデバッガーのメモリ使用量を追跡している場合は、' derf 'として表示されます。

`_RxAllocatePoolWithTag`で割り当てられたメモリは `_RxFreePool`を呼び出すことによって解放する必要があります。

`_RxAllocatePoolWithTag`の呼び出し元は、`IRQL <= ディスパッチ_レベル`で実行されている必要があります。ディスパッチ_レベルで実行している呼び出し元は、*型*パラメーターに `NonPagedPool` 値を指定する必要があります。 `IRQL <= APC_LEVEL` で実行される呼び出し元は、*型*パラメーターに任意のプール_型の値を指定できます。

要件

対象プラットフォーム	Desktop
Header	Ntrxdef.h (Ntrxdef.h を含む)
IRQL	「解説」を参照してください。

関連項目

[ExAllocatePoolWithTag](#) に

[_RxCheckMemoryBlock](#)

[_RxFreePool](#)

_RxCheckMemoryBlock ルーチン

2020/04/24 • [Edit Online](#)

_RxCheckMemoryBlockメモリブロックを特別な RX チェック_プール_ヘッダー ヘッダーの署名。ルーチンを使用するには、ネットワークのミニ リダイレクター ドライバーはメモリにこの特別な署名ブロックを追加する必要がありますので注意が割り当てられます。この特殊なヘッダー ブロックが実装されていないために、このルーチンを使用しない必要があります。

構文

```
BOOLEAN _RxCheckMemoryBlock(  
    PVOID Buffer,  
    PSZ   FileName,  
    ULONG LineNumber  
);
```

パラメーター

バッファー

プールのメモリが解放されるバッファーへのポインター。

ファイル名

メモリの割り当てが発生したソース ファイル名へのポインター。

LineNumber

メモリの割り当てが発生したソース ファイル内の行番号。

戻り値

RxCheckMemoryBlock返しますTRUEメモリブロックは、チェックをパスした場合またはFALSE失敗した場合。

注釈

推奨されます、RxCheckMemoryBlockこのルーチンを直接使用する代わりにマクロが呼び出されます。製品版ビルドでは、このマクロに何も定義されます。呼び出すチェック済みのビルドでこのマクロが定義されている _RxCheckMemoryBlockします。

このルーチンは、特殊なメモリのヘッダー ブロックからは使用できません (RX_プール_ヘッダー) を呼び出すときに、このルーチンをチェックしない追加、_RxAllocatePoolWithTagルーチン。ネットワークのミニ リダイレクター ドライバーは、このルーチンを使用するために割り当てられたメモリへのこの特殊な署名ブロックを追加する必要があります。

必要条件

対象プラットフォーム	Desktop
Header	Ntrxdef.h (Ntrxdef.h を含む)
IRQL	<= APC_LEVEL

関連項目

[_RxAllocatePoolWithTag](#)

[_RxFreePool](#)

_RxFreePool 関数

2020/04/24 • [Edit Online](#)

_RxFreePoolを使用して割り当てられていたメモリを解放 _RxAllocatePoolWithTagします。

構文

```
VOID _RxFreePool(  
    PVOID Buffer,  
    PSZ   FileName,  
    ULONG LineNumber  
);
```

パラメーター

バッファー

プールのメモリが解放されるバッファーへのポインター。

ファイル名

メモリの割り当てが発生したソース ファイル名へのポインター。このパラメーターは現在は使用されません。

LineNumber

メモリの割り当てが発生したソース ファイル内の行番号。このパラメーターは現在は使用されません。

戻り値

なし

注釈

推奨されます、RxFreePoolこのルーチンを直接使用する代わりにマクロが呼び出されます。呼び出す製品版ビルドでこのマクロが定義されているExFreePoolします。

割り当てられたメモリ [_RxAllocatePoolWithTag](#) を呼び出して解放する必要が _RxFreePoolします。

_RxFreePool ルーチンの呼び出し ExFreePoolします。

要件

対象プラットフォーム	Desktop
Header	Ntrxdef.h (Ntrxdef.h を含む)
IRQL	<= APC_LEVEL

関連項目

[_RxAllocatePoolWithTag](#)

`_RxCheckMemoryBlock`

__RxSynchronizeBlockingOperationsMaybeDroppingFcbLock 関数

2020/04/24 • [Edit Online](#)

__RxSynchronizeBlockingOperationsMaybeDroppingFcbLockは、ブロックしている i/o 要求を同じ作業キューに同期します。

構文

```
NTSTATUS __RxSynchronizeBlockingOperationsMaybeDroppingFcbLock(  
    _Inout_ PRX_CONTEXT RxContext,  
    _Inout_ PLIST_ENTRY BlockingIoQ,  
    _In_     BOOLEAN     DropFcbLock  
);
```

パラメーター

RxContext [in, out]

同期されている操作の RX_コンテキストへのポインター。

BlockingIoQ [in, out]

キューのリスト_エントリへのポインター。

] の *Dropfcblock* [

FCB リソースを解放する必要があるかどうかを示すブール値。このパラメーターが TRUE の場合、FCB リソースが解放されます。

戻り値

__RxSynchronizeBlockingOperationsMaybeDroppingFcbLockは成功時に成功したか、次のいずれかのような適切な NTSTATUS 値_を返します。

NTSTATUS	説明
STATUS_CANCELLED	I/o 要求と関連付けられた RX_CONTEXT が取り消されました。
■	<i>RxContext</i> は非同期操作のためのもので、 <i>RxContext</i> がキューに追加されました。

注釈

__RxSynchronizeBlockingOperationsMaybeDroppingFcbLockルーチンは、ブロッキング i/o 要求を同じ作業キューに同期します。RDBSS は、内部的に_を使用して名前付きパイプ操作を同期するために、_を使用します。作業キューは、*RxContext*によってポイントされる RX_コンテキスト構造のpfcbメンバーに関連付けられているファイルオブジェクト拡張機能 (fobx) によって参照されるキューです。

ネットワークミニリダイレクターは __RxSynchronizeBlockingOperationsMaybeDroppingFcbLockを使用して、ネットワークミニリダイレクターによって管理されている別のキューで操作を同期できます。

*RxContext*が非同期操作にマークされている場合は、__RxSynchronizeBlockingOperationsMaybeDroppingFcbLockによって、*RxContext*がキューに追加され、STATUS_PENDING が返されます。*RxContext*が同期操作にマークされている場合、__RxSynchronizeBlockingOperationsMaybeDroppingFcbLockはブロックされ、RxResumeBlockedOperations への呼び出

しが行われると*RxContext*が再開され [_順次](#)。

ブロッキング i/o 要求が取り消された場合、`__RxSynchronizeBlockingOperationsMaybeDroppingFcbLock`は、エラーを示す状態_キャンセルを返します。

*RxContext*が指す `RX_`コンテキスト構造の*syncevent*メンバーは、
`__RxSynchronizeBlockingOperationsMaybeDroppingFcbLock`を呼び出す前にリセットされている必要があります。
*Dropfcblock*パラメーターがTRUEに設定されている場合は、
`__RxSynchronizeBlockingOperationsMaybeDroppingFcbLock`を呼び出す前に、FCB リソースをロックする必要があります。

次の2つのマクロは、Windows XP と Windows 2000 で `__RxSynchronizeBlockingOperationsMaybeDroppingFcbLock`を呼び出すために定義されています。

`RxSynchronizeBlockingOperations` - *dropfcblock*パラメーターをFALSEに設定してを呼び出します。

`RxSynchronizeBlockingOperationsAndDropFcbLock` - *dropfcblock*パラメーターをTRUEに設定してを呼び出します。

要件

対象プラットフォーム	Desktop
バージョン	<code>__RxSynchronizeBlockingOperationsMaybeDroppingFcbLock</code> ルーチンは、Windows XP および Windows 2000 でのみ使用できます。
Header	Rxcontx (Rxcontx を含む)

関連項目

[RxCompleteRequest_Real](#)

[RxCreateRxContext](#)

[RxDereference](#)

[RxDereferenceAndDeleteRxContext_Real](#)

[RxInitializeContext](#)

[RxPrepareContextForReuse](#)

[RxResumeBlockedOperations_順次](#)

[__RxSynchronizeBlockingOperations](#)

SeStopImpersonatingClient ルーチン

2020/04/24 • [Edit Online](#)

SeStopImpersonatingClient ルーチンは、ユーザーの呼び出し元スレッドの偽装を終了します。

構文

```
VOID SeStopImpersonatingClient(void);
```

パラメーター

このルーチンにはパラメーターがありません。

戻り値

なし

注釈

サーブスレッドは、[SeImpersonateClientEx](#) ルーチンを呼び出すことによってユーザーの権限を借用できます。スレッドがユーザーの偽装を完了すると、SeStopImpersonatingClient ルーチンが呼び出され、偽装が終了します。

要件

対象プラットフォーム	Desktop
バージョン	Windows XP 以降のバージョンの Windows オペレーティングシステムで使用できます。
Header	Ntifs (Ntifs を含む)
IRQL	PASSIVE_LEVEL

関連項目

[SeImpersonateClientEx](#)

PCOMplete_ロック_IRP_ルーチンルーチン

2020/04/24 • [Edit Online](#)

ファイルシステムフィルタードライバー (レガシフィルタ) は、フィルタの *Completelockirproutine* コールバックとして、IRP_ルーチン型のルーチン、pcomplete_ロックを登録できます。

構文

```
PCOMplete_LOCK_IRP_ROUTINE CompleteLockIrpRoutine;  
  
NTSTATUS CompleteLockIrpRoutine(  
    _In_ PVOID Context,  
    _In_ PIRP Irp  
)  
{ ... }
```

パラメーター

] のコンテキスト

[Fsrtlprocessfilelock](#) に渡されたコンテキストポインター。

] のIrp

ファイルロックの irp [_MJ_ロック_制御](#) 要求を完了しています。ロック要求の種類は、次のいずれかになります。

IRP_ロック

IRP_ロック解除_すべて

IRP_、_キーによってすべての__ロック解除_

IRP__のロック解除_SINGLE

戻り値

このルーチンは、STATUS_SUCCESS または適切な NTSTATUS 値を返します。成功コードではない NTSTATUS 値が返された場合、ファイルのロックがファイルから削除されます。

注釈

ファイルシステムフィルタードライバー (レガシフィルタ) では、必要に応じて、PCOMplete_ロック_IRP_ルーチン型のルーチンを、バイト範囲のファイルロックのための従来のフィルタの *Completelockirproutine* ルーチンとして指定できます。

このルーチンを指定するために、レガシフィルタは [Fsrtlallocatefilelock](#) または [FsRtlInitializeFileLock](#) の *Completelockirproutine* パラメーターとしてルーチンへのポインターを渡します。

レガシフィルタでファイルロックに対して *Completelockirproutine* ルーチンが指定されている場合、[IRP_MJ_ロック_制御](#) 操作がファイルロックに対して実行されると、システムはこのルーチンを呼び出します。

要件

対象プラットフォーム	Desktop

Header	Ntifs (Ntifs を含む)
IRQL	<= APC_LEVEL

関連項目

[FsRtlAllocateFileLock](#)

[FsRtlCheckLockForReadAccess](#)

[FsRtlCheckLockForWriteAccess](#)

[FsRtlFreeFileLock](#)

[FsRtlInitializeFileLock](#)

[FsRtlProcessFileLock](#)

[FsRtlUninitializeFileLock](#)

[IRP_MJ_ロック_コントロール](#)

[PUNLOCK_ルーチン](#)

PFREE_FUNCTION 関数ポインター

2020/08/04 • [Edit Online](#)

PFREE_FUNCTION型指定されたルーチンは、ファイルシステムのレガシフィルタードライバーによって、フィルター
のFreeCallbackコールバックルーチンとして登録できます。ファイルシステムは、ファイルシステム
がFsRtlTeardownPerFileContextsを使用してファイルコンテキストオブジェクトを削除するか、
FsRtlTeardownPerStreamContextsを使用してストリームコンテキストオブジェクトを削除すると、FreeCallbackを呼び出しま
す。

FREE_FUNCTION型を使用して、コールバックルーチンを宣言する必要があります。詳細については、「解説」の例を参照してく
ださい。

構文

```
typedef VOID ( *FreeCallback)(  
    _In_ PVOID Buffer  
);
```

パラメーター

バッファー [から]

解放するFSRTL_PER_FILE_CONTEXTまたはFSRTL_PER_STREAM_CONTEXT構造体へのポインター。

戻り値

なし

解説

ファイルシステムがファイルのファイルコンテキストオブジェクトを破棄すると、FsRtlTeardownPerFileContextsを呼び出す必要が
あります。このルーチンは、ファイルに関連付けられているすべてのファイルごとのコンテキスト構造のFreeCallbackルーチンを呼び出
します。このコールバックルーチンは、FSRTL_PER_FILE_CONTEXTオブジェクトと関連付けられているコンテキスト情報に使用さ
れるメモリを解放する必要があります。これは、FsRtlTeardownPerStreamContextsが呼び出されたときのストリームコンテキ
ストごとのケースでもあり、FreeCallbackはFSRTL_PER_STREAM_CONTEXTオブジェクトに使用されるメモリを解放します。

FreeCallbackの呼び出し中に、ファイルコンテキストオブジェクトごと、またはストリームコンテキストオブジェクトごとにアクセスを同期す
る方法については、FsRtlTeardownPerFileContextsとFsRtlTeardownPerStreamContextsを参照してください。

NOTE

FreeCallbackルーチンは、ファイルシステムに再帰的に呼び出したり、ファイルシステムリソースを取得したりすることはできません。

Myfreefunctionという名前のFreeCallback callback 関数を定義するには、まず、Static Driver Verifier (sdv) およびその他の
検証ツールが必要とする関数宣言を次のように指定する必要があります。

```
FREE_FUNCTION MyFreeFunction;
```

次のようにコールバック関数を実装します。


```
VOID
MyFreeFunction (
    __in PVOID Buffer
)
{...}
```

要件

ターゲットプラットフォーム: デスクトップ

バージョン: windows Vista 以降で使用できます。

ヘッダー: Wdm (wdm.h または Ntddk を含む)

IRQL: <= APC_LEVEL

関連項目

[FsRtlTeardownPerFileContexts](#)

[FsRtlTeardownPerStreamContexts](#)

[FSRTL_PER_FILE_CONTEXT](#)

[FSRTL_PER_STREAM_CONTEXT](#)

[レガシ ファイル システム フィルター ドライバー内のファイル別コンテキストの追跡](#)

[レガシ ファイル システム フィルター ドライバー内のストリーム別コンテキストの追跡](#)

PUNLOCK_ルーチン関数ポインター

2020/04/24 • [Edit Online](#)

フィルター (レガシフィルターまたはミニフィルター) では、PUNLOCK_ルーチンで型指定されたルーチンを、ファイル_ロック構造のフィルターの *UnlockRoutine* コールバックルーチンとして登録できます。

構文

```
typedef VOID ( *UnlockRoutine)(
    _In_ PVOID          Context,
    _In_ PFILE_LOCK_INFO FileLockInfo
);
```

パラメーター

] のコンテキスト

[Fltprocessfilelock](#) または [fsrtlprocessfilelock](#) に渡されたコンテキストポインター。

] の *Filelockinfo* [

バイト範囲ロックのロック_情報構造体を_ファイルへの非透過的ポインター。

戻り値

なし

注釈

フィルター (レガシフィルターまたはミニフィルター) では、必要に応じて PUNLOCK_ルーチン型のルーチンを、バイト範囲のファイルロックのフィルターの *UnlockRoutine* コールバックとして指定できます。

ファイル_ロック構造の *UnlockRoutine* ルーチンをフィルターが指定した場合、このルーチンは、ロックがファイル内のロックされたバイト範囲から削除されるときに呼び出されます。

ミニパスでは、[FltAllocateFileLock](#) の *UnlockRoutine* パラメーターとしてルーチンへのポインターを渡すことによって、このルーチンを指定します。

レガシフィルターでは、[Fsrtlallocatefilelock](#) または [FsRtlInitializeFileLock](#) の *UnlockRoutine* パラメーターとしてルーチンへのポインターを渡すことによって、このルーチンを指定します。

要件

対象プラットフォーム	Desktop
Header	Ntifs (Ntifs または Fltkernel.h を含む)
IRQL	<= APC_LEVEL

関連項目

[FltAllocateFileLock](#)

[FltCheckLockForReadAccess](#)

[FltCheckLockForWriteAccess](#)

[FltFreeFileLock](#)

[FltInitializeFileLock](#)

[FltProcessFileLock](#)

[FltUninitializeFileLock](#)

[FsRtlAllocateFileLock](#)

[FsRtlCheckLockForReadAccess](#)

[FsRtlCheckLockForWriteAccess](#)

[FsRtlFreeFileLock](#)

[FsRtlInitializeFileLock](#)

[FsRtlProcessFileLock](#)

[FsRtlUninitializeFileLock](#)

[IRP_MJ_ロック_コントロール](#)

[PCOMPLETE_ロック_IRP_ルーチン](#)

[PFLT__ロック_コールバック_データ_ルーチンの完了](#)

デュアル_OPLOCK_キー_ECP_コンテキスト構造

2020/04/24 • [Edit Online](#)

デュアル_oplock_キー_ECP_コンテキスト構造には、デュアル OPLOCK キーの余分な create パラメーターコンテキストが含まれています。ターゲットファイルオブジェクトと親ファイルオブジェクトの両方の oplock キーは、この構造体で設定できます。

構文

```
typedef struct _DUAL_OPLOCK_KEY_ECP_CONTEXT {
    GUID      ParentOplockKey;
    GUID      TargetOplockKey;
    BOOLEAN   ParentOplockKeySet;
    BOOLEAN   TargetOplockKeySet;
} DUAL_OPLOCK_KEY_ECP_CONTEXT, *PDUAL_OPLOCK_KEY_ECP_CONTEXT;
```

Members

ParentOplockKey

親 oplock キー値を表す GUID。

TargetOplockKey

ターゲット oplock キー値を表す GUID。

ParentOplockKeySet

ParentOplockKeyに親の oplock キーの有効な GUID が含まれている場合は TRUE に設定します。

TargetOplockKeySet

TargetOplockKeyにターゲットの oplock キーの有効な GUID が含まれている場合は TRUE に設定します。

コメント

デュアル_OPLOCK_キー_ECP_コンテキスト構造は、ファイルとディレクトリに対する oplock 要求を許可するためのデュアル OPLOCK キーを提供します。[OPLOCK_キー_ecp_コンテキスト](#)構造、デュアル_OPLOCK_キー_ECP_コンテキストは、追加のパラメーターリスト ([ecp_リスト](#)) で設定され、後で[IRP_MJ](#)の処理中にファイルオブジェクトに関連付けられます。これは、ファイルシステムまたはファイルシステムフィルタードライバーによって作成されます。

[FsRtlAllocateExtraCreateParameter](#)、[FsRtlInitializeExtraCreateParameter](#)、[FltRemoveExtraCreateParameter](#)などのサポートルーチンを呼び出すときには、値GUID_ECP_2_OPLOCK_キーが使用されます。

要件

バージョン	この構造体は、Windows 8 以降で使用できます。
ヘッダー	Ntifs (Ntifs を含む)

参照

[ECP_一覧](#)

[_コンテキストを作成_IO_ドライバー](#)

[IoCreateFileEx](#)

[IRP_MJ_作成](#)

[_キー_ECP_コンテキストの OPLOCK](#)

OPLOCK_KEY_ECP_CONTEXT 構造体

2020/08/04 • [Edit Online](#)

OPLOCK_KEY_ECP_CONTEXT 構造体は、OPLOCK キーをファイルにアタッチするために使用されます。この構造は、Windows 8 以降のバージョンでは廃止されています。フィルターでは、代わりに[DUAL_OP_LOCK_KEY_ECP_CONTEXT](#)を使用する必要があります。

構文

```
typedef struct _OPLOCK_KEY_ECP_CONTEXT {
    GUID OplockKey;
    ULONG Reserved;
} OPLOCK_KEY_ECP_CONTEXT, *POPLOCK_KEY_ECP_CONTEXT;
```

メンバー

OplockKey

Oplock キーの GUID。この GUID は、異なるハンドル間で共有され、同じクライアントキャッシュに属するものとして識別されます。2つのハンドルが同じ oplock キーを共有する場合、1つのハンドルに対して実行される要求は、もう一方のハンドルの未解決の oplock を解除しません。

予約されています。

予約済み。を0に設定する必要があります。

注釈

ファイルの作成時に ECPs を使用して追加情報をファイルに関連付ける方法については、「[IRP_MJ_CREATE 操作での追加の作成パラメーターの使用](#)」を参照してください。

ミニフィルターでは、前から ECP が出てきたときに、OPLOCK_KEY_ECP_CONTEXT 構造の内容を変更することはできません。このキーを使用して、oplock キー ECP に関する情報のみを取得する必要があります。この問題の詳細については、「[システム定義の ECPs](#)」を参照してください。

Oplock キーを使用すると、アプリケーション独自の oplock を中断することなく、同じストリームに対して複数のハンドルを開くことができます。Oplock が解除されるのは、アプリケーションが共有違反 (STATUS_SHARING_VIOLATION) を受信した後です。

Oplock は、ストリームが開かれるときにストリームハンドルで許可されます。このようなストリームハンドルは、oplock キーに関連付けることができます。呼び出し元は、[IoCreateFileEx](#) ルーチンに oplock キーを明示的に指定して、ストリームハンドルを作成できます。呼び出し元がハンドルを作成するときに、呼び出し元が oplock キーを明示的に指定していない場合、オペレーティングシステムはハンドルのキーが他のハンドルの他のキーと異なるようにハンドルに関連付けられた一意の oplock キーとしてハンドルを扱います。Oplock が許可されている以外のハンドルでファイル操作を受信し、oplock のハンドルに関連付けられている oplock キーが操作のハンドルに関連付けられているキーと異なっていて、その操作が現在許可されている oplock と互換性がない場合、その oplock は解除されます。Oplock は、互換性のない操作を実行しているプロセスまたはスレッドの場合でも中断されます。たとえば、排他的 oplock が付与されているストリームをプロセスが開き、同じプロセスが別の (またはない) oplock キーを使用して同じストリームを再び開く場合、排他的な oplock は直ちに解除されます。

Oplock キーは、ハンドルが作成されるときにハンドルに関連付けられます。Oplock が付与されていない場合でも、ハンドルを oplock キーに関連付けることができます。

Oplock キーと oplock キーの詳細については、「[Oplock セマンティクスの概要](#)」を参照してください。

要件

バージョン: この構造は windows 7 以降で使用でき、windows 8 以降のバージョンでは廃止されています。

ヘッダー: *Ntifs* (Ntifs または Ntddk を含む)

関連項目

[DUAL_OP_LOCK_KEY_ECP_CONTEXT](#)

[IoCreateFileEx](#)

WCIFS_リダイレクト_ECP_CONTEXT 構造体

2020/04/24 • [Edit Online](#)

特定の操作を作成するファイルのリダイレクト状態について説明します。

構文

```
typedef struct _WCIFS_REDIRECTION_ECP_CONTEXT {
    USHORT      Size;
    USHORT      Flags;
    FILE_ID_128 FileId;
    GUID         VolumeGuid;
} WCIFS_REDIRECTION_ECP_CONTEXT, *PWCIFS_REDIRECTION_ECP_CONTEXT;
```

メンバー

サイズ

構造体のサイズ `sizeof(WCIFS_REDIRECTION_ECP_CONTEXT)` します。

フラグ

ファイルのリダイレクトの状態を示します。

I	II
WCIFS_REDIRECTION_FLAGS_CREATE_SERVICED_FROM_LAYER 0x0001	これは LayerRootLocations のレジストリ キーに登録されていない層からリダイレクトされたファイルです。
WCIFS_REDIRECTION_FLAGS_CREATE_SERVICED_FROM_SCRATCH 0x0002	これは、新しいまたは変更されたファイルではリダイレクトされません。
WCIFS_REDIRECTION_FLAGS_CREATE_SERVICED_FROM_REGISTERED_LAYER 0x0004	これは、LayerRootLocations レジストリ キーに記載されているレイヤーからリダイレクトされたファイルです。
WCIFS_REDIRECTION_FLAGS_CREATE_SERVICED_FROM_REMOTE_LAYER 0x0008	これは、コンテナを基準としてリモート ファイル システムからリダイレクトされたファイルです。これは、そのサーバー上のレイヤーとして登録されていない可能性があります。HYPER-V コンテナでは、リモート サーバーは、HYPER-V コンテナ ユーティリティ VM のホストです。

FileId

バックアップ ファイルの識別子。

VolumeGuid

バックアップ ファイルが存在するディスク ボリュームの GUID ベースの識別子です。

要件

サポートされている最小のクライアント	Windows 10 バージョン 1607

サポートされている最小のサーバー	Windows Server 2016
Header	Ntifs.h

FLT_ファイル_名の_オプション

2020/08/04 • [Edit Online](#)

FLT_FILE_NAME_OPTIONS 型は、ファイル名情報クエリの名前の形式、クエリメソッド、およびフラグを指定するフラグのビットマスクです。

```
typedef ULONG FLT_FILE_NAME_OPTIONS;
#define FLT_VALID_FILE_NAME_FORMATS 0x000000ff
    #define FLT_FILE_NAME_NORMALIZED 0x00000001
    #define FLT_FILE_NAME_OPENED 0x00000002
    #define FLT_FILE_NAME_SHORT 0x00000003
#define FLT_VALID_FILE_NAME_QUERY_METHODS 0x0000ff00
    #define FLT_FILE_NAME_QUERY_DEFAULT 0x00000100
    #define FLT_FILE_NAME_QUERY_CACHE_ONLY 0x00000200
    #define FLT_FILE_NAME_QUERY_FILESYSTEM_ONLY 0x00000300
    #define FLT_FILE_NAME_QUERY_ALWAYS_ALLOW_CACHE_LOOKUP 0x00000400
#define FLT_VALID_FILE_NAME_FLAGS 0xff000000
    #define FLT_FILE_NAME_REQUEST_FROM_CURRENT_PROVIDER 0x01000000
    #define FLT_FILE_NAME_DO_NOT_CACHE 0x02000000
    #define FLT_FILE_NAME_ALLOW_QUERY_ON_REPARSE 0x04000000
```

ビット 0 ~ 7 はファイル形式を示し、[FltGetFileNameFormat](#) マクロを使用してクエリを実行できます。これらの形式の詳細については、「[FLT_FILE_NAME_INFORMATION](#)」を参照してください。現在、次の値が定義されています。

「	”
FLT_FILE_NAME_NORMALIZED	ファイルの正規化された名前。
FLT_FILE_NAME_OPENED	ハンドルがこのファイルに開かれたときに使用された名前。この名前は正規化されていません。
FLT_FILE_NAME_SHORT	ファイルの短い (8.3) 名前。ファイルの短い名前には、ボリューム名、ディレクトリパス、またはストリーム名は含まれません。この名前は正規化されていません。

Bits 8 ~ 15 では、フィルターマネージャーによって使用されるファイル名のクエリ方法を指定します。このメソッドには、[FltGetFileNameQueryMethod](#) マクロを使用してクエリを実行できます。これらの値の詳細については、「[FltGetFileNameInformation](#)」を参照してください。現在、次の値が定義されています。

「	”
FLT_FILE_NAME_QUERY_DEFAULT	ファイル名のファイルシステムに対してクエリを実行するのが現在安全でない場合は、何も行いません。それ以外の場合は、フィルターマネージャーの名前キャッシュでファイル名情報を照会します。名前がキャッシュに見つからない場合は、ファイルシステムに対してクエリを実行し、結果をキャッシュします。
FLT_FILE_NAME_QUERY_CACHE_ONLY	フィルターマネージャーの名前キャッシュでファイル名情報を照会します。ファイルシステムに対してクエリを実行しないでください。
FLT_FILE_NAME_QUERY_FILESYSTEM_ONLY	ファイルシステムでファイル名情報を照会します。フィルターマネージャーの名前キャッシュに対してクエリを実行しないでください。また、ファイルシステムクエリの結果をキャッシュしません。

I	II
FLT_FILE_NAME_QUERY_ALWAYS_ALLOW_CACHE_LOOKUP	フィルターマネージャーの名前キャッシュでファイル名情報を照会します。名前がキャッシュ内に見つからず、その名前が現在安全である場合は、ファイルシステムに対してファイル名情報を照会し、結果をキャッシュします。

ビット 16 ~ 23 は現在使用されていません。

名前プロバイダーミニフィルターでは、ファイル名フラグを指定するためにビット 24 ~ 31 が使用されます。現在、次の値が定義されています。

I	II
FLT_FILE_NAME_REQUEST_FROM_CURRENT_PROVIDER	名前プロバイダーのミニフィルターでは、このフラグを使用して、名前のクエリ要求をスタック内の下位の名前プロバイダーによって満たされるのではなく、自身 (名前プロバイダーミニフィルター) にリダイレクトするように指定できます。
FLT_FILE_NAME_DO_NOT_CACHE	このフラグは、このクエリから取得した名前がキャッシュされないことを示します。Name provider ミニフィルターは、中間クエリを実行して名前を生成するときに、このフラグを使用します。
FLT_FILE_NAME_ALLOW_QUERY_ON_REPARSE	名前プロバイダーのミニパスでは、このフラグを使用して、STATUS_REPARSE が返された場合でも、作成後のパスで名前をクエリするのが安全であることを指定できます。 FileObject > FileName フィールドが変更されていないことを確認するのは、呼び出し元の責任です。マウントポイントやシンボリックリンクの再解析ポイントでは、このフラグを使用しないでください。

要件

ヘッダー * * * *: fltkernel .h (fltkernel. h を含む)

関連トピック

[FLT_FILE_NAME_INFORMATION](#)

[FltGetDestinationFileNameInformation](#)

[FltGetFileNameFormat](#)

[FltGetFileNameInformation](#)

[FltGetFileNameInformationUnsafe](#)

[FltGetFileNameQueryMethod](#)

ファイル_ロック 構造

2020/04/24 • [Edit Online](#)

オペレーティングシステムは、ファイルのロックをサポートするために、不透明なファイル_ロック構造を使用します。

構文

```
typedef struct _FILE_LOCK {
    PCOMPLETE_LOCK_IRP_ROUTINE CompleteLockIrpRoutine;
    PUNLOCK_ROUTINE             UnlockRoutine;
    BOOLEAN                     FastIoIsQuestionable;
    BOOLEAN                     SpareC[3];
    PVOID                       LockInformation;
    FILE_LOCK_INFO              LastReturnedLockInfo;
    PVOID                       LastReturnedLock;
    volatile LONG               LockRequestsInProgress;
} FILE_LOCK, *PFILE_LOCK;
```

Members

CompleteLockIrpRoutine

システム用に予約されています。

UnlockRoutine

システム用に予約されています。

問題あり

システム用に予約されています。

SpareC

システム用に予約されています。

LockInformation

システム用に予約されています。

LastReturnedLockInfo

システム用に予約されています。

LastReturnedLock

システム用に予約されています。

LockRequestsInProgress

システム用に予約されています。

注釈

ファイルシステムドライバー、レガシファイルシステムフィルタードライバーおよびミニフィルターでは、さまざまなルーチンを使用して、ファイル_ロックオブジェクトを作成および使用したり、ファイルへの読み取り/書き込みアクセスをテストしたりすることができます。

- ファイル_ロックオブジェクトに割り当てるには、[FsRtlAllocateFileLock](#)または[FltAllocateFileLock](#)を呼び出します。
- 初期化されていないファイル_ロックオブジェクトを初期化するには、[FsRtlInitializeFileLock](#)または[FltInitializeFileLock](#)を呼び出します。[FsRtlAllocateFileLock](#)または[FltAllocateFileLock](#)から返されたファイル_ロックは既に初期化されていることに注意してください。

- オブジェクト_ロックを解除するには、[Fsrtluninitializefilelock](#)または[fltuninitializefilelock](#)を呼び出します。
- [Fsrtlallocatefilelock](#)ルーチンによって割り当てられるファイル_ロックオブジェクトを解放するには、[Fsrtlfreefilelock](#)を呼び出します。[FltAllocateFileLock](#)ルーチンによって割り当てられるファイル_ロックオブジェクトを解放するには、[Fltfreefilelock](#)を呼び出します。

ファイル_ロックが初期化された後、[Fsrtlchecklockforreadaccess](#)、[fltchecklockforwriteaccess](#)、[Fsrtlchecklockforreadaccess](#)などのルーチンを使用して、他のスレッドがファイルにアクセスできるかどうかを判断できます。

要件

バージョン	Microsoft Windows 2000、およびそれ以降のバージョンの Windows オペレーティングシステムで使用できます。
Header	Ntifs (FltKernel.h または Ntifs を含む)

関連項目

- [FltAllocateFileLock](#)
- [FltCheckLockForReadAccess](#)
- [FltCheckLockForWriteAccess](#)
- [FltInitializeFileLock](#)
- [Fltinitializefilelock](#) [Fsrtlallocatefilelock](#)
- [FsRtlCheckLockForReadAccess](#)
- [FsRtlCheckLockForWriteAccess](#)
- [FsRtlFastCheckLockForRead](#)
- [FsRtlFastCheckLockForWrite](#)
- [FsRtlInitializeFileLock](#)
- [FsRtlUninitializeFileLock](#)

ACE

2020/04/24 • [Edit Online](#)

ACE は、アクセス制御リスト (ACL) のアクセス制御エントリ (ACE) です。

次に、現在定義されている ACE 型を示します。

名前	説明
ACCESS_ALLOWED_ACE	指定された権限をユーザーまたはグループに付与します。この ACE は、任意 ACL (DACL) に格納されます。
ACCESS_DENIED_ACE	ユーザーまたはグループに対して指定された権限を拒否します。この ACE は、DACL に格納されます。
SYSTEM_AUDIT_ACE	システムレベルの監査を実行するアクセスの種類を指定します。この ACE は、システム ACL (SACL) に格納されます。

4つ目の ACE 構造、システム_アラーム_ACE は、現在サポートされていません。

ACL には、Ace の一覧が含まれています。ACE は、特定のユーザーまたはグループのオブジェクトへのアクセスを定義するか、特定のユーザーまたはグループのシステム管理メッセージまたはアラームを生成するアクセスの種類を定義します。ユーザーまたはグループは、セキュリティ識別子 (SID) によって識別されます。

各 ACE は、ACE_ヘッダー構造で開始されます。ヘッダーに続くデータの形式は、ヘッダーに指定されている ACE の種類によって異なります。

この構造体は、32ビットの境界上に配置する必要があります。

要件: ntifs (ntifs を含む)

関連トピック

[アクセス_許可_ACE](#)

[ACE_アクセス_拒否されました](#)

[ACE_ヘッダー](#)

[.ACL](#)

[RtlAddAccessAllowedAce](#)

[RtlGetAce](#)

[SID](#)

[システム_アラーム_ACE](#)

[SYSTEM_AUDIT_ACE](#)

セキュリティ_記述子_コントロール

2020/04/24 • [Edit Online](#)

セキュリティ_記述子_コントロール型は、[セキュリティ_記述子](#) 構造体またはそのコンポーネントの意味を修飾する一連のビットフラグです。各セキュリティ記述子には、セキュリティ_記述子_制御ビットを格納する制御メンバーがあります。

セキュリティ_記述子_コントロール

```
typedef USHORT SECURITY_DESCRIPTOR_CONTROL, *PSECURITY_DESCRIPTOR_CONTROL;
```

コントロールの値には、次のセキュリティ_記述子_制御ビットフラグの組み合わせを含めることができます。

VALUE	II
SE_DACL_AUTO_INHERIT_REQ	セキュリティ記述子によって保護されているオブジェクトのプロバイダーが、自動的に DACL を既存の子オブジェクトに伝達するように要求します。プロバイダーで自動継承がサポートされている場合は、既存の子オブジェクトに DACL を伝達し、オブジェクトとその子オブジェクトのセキュリティ記述子に SE_DACL_AUTO_INHERITED ビットを設定します。
SE_DACL_AUTO_INHERITED	Windows 2000 以降、では、DACL が既存の子オブジェクトへの継承可能な Ace の自動伝達をサポートするセキュリティ記述子を示しています。 自動継承をサポートする Windows 2000 Acl の場合、このビットは常に設定されます。自動継承をサポートしていない Windows NT 4.0 Acl とこれらの Acl を区別するために使用されます。 このビットは、継承可能な Ace の自動伝達をサポートしていない Windows NT 4.0 以前のセキュリティ記述子では設定されません。
SE_DACL_DEFAULTED	既定の DACL を持つセキュリティ記述子を示します。たとえば、オブジェクトの作成者が DACL を指定していない場合、オブジェクトは作成者のアクセストークンから既定の DACL を受け取ります。このフラグは、ACE の継承に関して、システムが DACL をどのように処理するかに影響する可能性があります。SE_DACL_PRESENT フラグが設定されていない場合、システムはこのフラグを無視します。 このフラグは、オブジェクトの最終的な DACL がどのように計算されるかを決定するために使用され、セキュリティ保護可能なオブジェクトのセキュリティ記述子コントロールに物理的に格納されることはありません。 このフラグを設定するには、Rtlsetdaclsecuritydescriptor を使用します。

VALUE	II
SE_DACL_PRESENT	DACL を持つセキュリティ記述子を示します。このフラグが設定されていない場合、またはこのフラグが設定され、DACL がNULLの場合、セキュリティ記述子はすべてのユーザーにフルアクセスを許可します。 このフラグは、セキュリティ記述子がセキュリティ保護可能なオブジェクトに関連付けられるまで、呼び出し元によって指定されたセキュリティ情報を保持するために使用されます。セキュリティ記述子がセキュリティ保護可能なオブジェクトに関連付けられると、SE_DACL_PRESENT フラグは常にセキュリティ記述子コントロールで設定されます。 このフラグを設定するには、Rtlsetdaclsecuritydescriptorを使用します。
SE_DACL_PROTECTED	セキュリティ記述子の DACL を、継承可能な Ace によって変更されないように保護します。
SE_DACL_UNTRUSTED	セキュリティ記述子の DACL によってポイントされた ACL が、信頼されていないソースによって提供されたことを示します。このフラグが設定されていて、複合 ACE が検出された場合、システムは Ace 内のサーバー Sid に対して既知の有効な Sid を置き換えます。
SE_GROUP_DEFAULTED	セキュリティ記述子のグループ SID を提供する、セキュリティ記述子の元のプロバイダーではなく、既定の機構。
SE_OWNER_DEFAULTED	セキュリティ記述子の所有者セキュリティ識別子 (SID) が指定されている場合、セキュリティ記述子の元のプロバイダーではなく、既定の機構。このフラグを設定するには、 RtlSetOwnerSecurityDescriptor を使用します。
SE_RM_CONTROL_VALID	セキュリティ記述子のリソースコントロールマネージャービットが有効であることを示します。リソースマネージャー制御ビットは、構造体にアクセスするリソースマネージャー固有の情報を含む、 SECURITY_DESCRIPTOR 構造体のSbz1メンバーの8ビットです。(詳細については、Microsoft Windows Software Development Kit (SDK) For Windows 7 および .NET Framework 4.0 のドキュメントを参照してください)。
SE_SACL_AUTO_INHERIT_REQ	セキュリティ記述子によって保護されているオブジェクトのプロバイダーが、自動的に SACL を既存の子オブジェクトに伝達するように要求します。プロバイダーで自動継承がサポートされている場合は、既存の子オブジェクトに SACL が伝達され、オブジェクトとその子オブジェクトのセキュリティ記述子に SE_SACL_AUTO_INHERITED ビットが設定されます。

VALUE	II
SE_SACL_AUTO_INHERITED	SACL が、既存の子オブジェクトへの継承可能な Ace の自動伝達をサポートするセキュリティ記述子を示します。このビットは、オブジェクトとその既存の子オブジェクトに対して自動継承アルゴリズムが実行されている場合にのみ設定されます。 このビットは、継承可能な Ace の自動伝達をサポートしていない Windows NT 4.0 以前のセキュリティ記述子では設定されません。
SE_SACL_DEFAULTED	SACL によって提供される、セキュリティ記述子の元のプロバイダーではなく、既定の機構。このフラグは、ACE 継承に関して、システムが SACL をどのように処理するかに影響する可能性があります。SE_SACL_PRESENT フラグが設定されていない場合、システムはこのフラグを無視します。
SE_SACL_PRESENT	SACL を持つセキュリティ記述子を示します。
SE_SACL_PROTECTED	継承可能な Ace によってセキュリティ記述子の SACL が変更されないように保護します。
SE_SELF_RELATIVE	連続したメモリブロック内のすべてのセキュリティ情報を持つ、自己相対形式のセキュリティ記述子を示します。このフラグが設定されていない場合、セキュリティ記述子は絶対形式です。詳細については、Windows SDK のドキュメントの「絶対セキュリティ記述子と自己相対セキュリティ記述子」を参照してください。
SE_SERVER_SECURITY	ソース (明示的または既定) に関係なく、ACL が入力 ACL に基づくサーバー ACL を持つオブジェクトのプロバイダーに対して、セキュリティ記述子によって保護されることを要求します。これは、すべての許可 Ace を、現在のサーバーを許可する複合 Ace に置き換えることによって行われます。このフラグは、サブジェクトが偽装している場合にのみ意味を持ちます。

要件

ntifs (ntifs を含む)

関連トピック

[ACE](#)

[.ACL](#)

[RtlSetDaclSecurityDescriptor](#)

[RtlSetOwnerSecurityDescriptor](#)

[セキュリティ_記述子](#)

セキュリティ情報

2020/04/24 • [Edit Online](#)

セキュリティ情報

```
typedef ULONG SECURITY_INFORMATION, *PSECURITY_INFORMATION;
```

セキュリティ情報型の値は、設定または照会するオブジェクトに関連するセキュリティ情報を識別するために使用されます。このセキュリティ情報には次のものが含まれます。

- オブジェクトの所有者
- オブジェクトのプライマリグループ
- オブジェクトの随意アクセス制御リスト (DACL)
- オブジェクトのシステム ACL (SACL)

セキュリティ情報の各項目は、ビットフラグによって指定されます。次の値はビットを指定します。

VALUE	〃	〃〃〃〃
DACL_SECURITY_INFORMATION	オブジェクトの DACL が設定または照会されていることを示します。 次の項目に対して、DACL に対するクエリが実行されます。 IRP_MJ_QUERY_SECURITY FLT_PARAMETERS (IRP_MJ_QUERY_SECURITY 用) FltQuerySecurityObject Sequerysecurity記述子情報 次の項目について、DACL が設定されます。 IRP_MJ_SET_SECURITY FLT_PARAMETERS (IRP_MJ_SET_SECURITY 用) FltSetSecurityObject Sesetsecurity記述子情報 Sesetsecurity記述子 Infoex	次の場合に READ_CONTROL アクセスが必要: IRP_MJ_QUERY_SECURITY FLT_PARAMETERS (IRP_MJ_QUERY_SECURITY 用) FltQuerySecurityObject Sequerysecurity記述子情報 次の場合に WRITE_DAC アクセスが必要: IRP_MJ_SET_SECURITY FLT_PARAMETERS (IRP_MJ_SET_SECURITY 用) FltSetSecurityObject Sesetsecurity記述子情報 Sesetsecurity記述子 Infoex

VALUE	II	IIII
GROUP_SECURITY_INFORMATION	オブジェクトのプライマリグループ識別子が設定または照会されていることを示します。 次の項目について、グループ識別子が照会されます。 IRP_MJ_QUERY_SECURITY FLT_PARAMETERS (IRP_MJ_QUERY_SECURITY 用) FltQuerySecurityObject Sequerysecurity記述子情報 IRP_MJ_SET_SECURITY FLT_PARAMETERS (IRP_MJ_SET_SECURITY 用) FltSetSecurityObject Sesetsecurity記述子情報 Sesetsecurity記述子 Infoex	次の場合に READ_CONTROL アクセスが必要: IRP_MJ_QUERY_SECURITY FLT_PARAMETERS (IRP_MJ_QUERY_SECURITY 用) FltQuerySecurityObject Sequerysecurity記述子情報 次の場合に WRITE_OWNER アクセスが必要: IRP_MJ_SET_SECURITY FLT_PARAMETERS (IRP_MJ_SET_SECURITY 用) FltSetSecurityObject Sesetsecurity記述子情報 Sesetsecurity記述子 Infoex
OWNER_SECURITY_INFORMATION	オブジェクトの所有者識別子が設定または照会されていることを示します。 次の項目について、所有者識別子が照会されます。 IRP_MJ_QUERY_SECURITY FLT_PARAMETERS (IRP_MJ_QUERY_SECURITY 用) FltQuerySecurityObject Sequerysecurity記述子情報 IRP_MJ_SET_SECURITY FLT_PARAMETERS (IRP_MJ_SET_SECURITY 用) FltSetSecurityObject Sesetsecurity記述子情報 Sesetsecurity記述子 Infoex	次の場合に READ_CONTROL アクセスが必要: IRP_MJ_QUERY_SECURITY FLT_PARAMETERS (IRP_MJ_QUERY_SECURITY 用) FltQuerySecurityObject Sequerysecurity記述子情報 次の場合に WRITE_OWNER アクセスが必要: IRP_MJ_SET_SECURITY FLT_PARAMETERS (IRP_MJ_SET_SECURITY 用) FltSetSecurityObject Sesetsecurity記述子情報 Sesetsecurity記述子 Infoex

VALUE	説明	備考
SACL_SECURITY_INFORMATION	オブジェクトの SACL が設定または照会されていることを示します。 次の項目に対して、SACL が照会されます。 IRP_MJ_QUERY_SECURITY FLT_PARAMETERS (IRP_MJ_QUERY_SECURITY 用) FltQuerySecurityObject Sequerysecurity記述子情報 IRP_MJ_SET_SECURITY FLT_PARAMETERS (IRP_MJ_SET_SECURITY 用) 次の項目について、SACL が設定されます。 FltSetSecurityObject Sesetsecurity記述子情報 Sesetsecurity記述子 Infoex	常に ACCESS_SYSTEM_SECURITY アクセスが必要です。
PROCESS_TRUST_LABEL_SECURITY_INFORMATION	予約済み。	

要件

Wdm (Wdm を含む)

関連トピック

[.ACL](#)

[セキュリティ記述子](#)

[Sequerysecurity記述子情報](#)

[Sesetsecurity記述子情報](#)

[Sesetsecurity記述子 Infoex](#)

[ZwQuerySecurityObject](#)

[ZwSetSecurityObject](#)

RxAssert ルーチン

2020/04/24 • [Edit Online](#)

RxAssertがインストールされている場合、RDBSS のチェックされたビルドに対してアサート文字列をカーネルデバッガーに送信します。RDBSS のリテールビルドの場合、このルーチンの呼び出しによってバグチェックが行われます。

構文

```
VOID RxAssert(  
    _In_      PVOID FailedAssertion,  
    _In_      PVOID FileName,  
    _In_      ULONG LineNumber,  
    _In_opt_  PCHAR Message  
);
```

パラメーター

] の失敗したアサーション
失敗したアサーション。

ファイル名[]
RxAssertまたはrtlassertが呼び出されたソースファイルの名前。

] のLineNumber [
RxAssertまたはrtlassertが呼び出されたソースファイル内の行番号。

でのメッセージ]、オプションの]
省略可能なメッセージ。

戻り値

なし

注釈

Rxassertヘッダーファイルを使用すると、この rxassert ルーチンも呼び出されるように Windows カーネルの RtlAssert 呼び出しが再定義されます。

リテールビルドでは、RxAssertは、BugCheckParamater1 として行番号を使用して、値0xa55a0000 を渡すKeバグ checkexを呼び出します。

要件

対象プラットフォーム	Desktop
Header	Rxassert (Rxassert を含む)
IRQL	<= APC_LEVEL

関連項目

[アサート](#)

[RtlAssert](#)

[RxDbgBreakPoint](#)

RxDbgBreakPoint 関数

2020/04/24 • [Edit Online](#)

RxDbgBreakPointがインストールされている場合は、カーネル デバッガーを中断します。

構文

```
VOID RxDbgBreakPoint(  
    ULONG LineNumber  
);
```

パラメーター

LineNumber

ソース内の行番号ファイルRxDbgBreakPointが呼び出されました。

戻り値

なし

コメント

このルーチンは、カーネルを呼び出すDbgBreakPointルーチン。

このルーチンは、インストールされている場合は、カーネル デバッガーによって処理される例外を発生させます。それ以外の場合、デバッグ システムによって処理されます。システムにデバッガーが接続されていない場合は、標準的な方法で、例外を処理することができます。

カーネル モードでは、結果としてブルー スクリーン (バグ チェック) が処理されていない中断例外になります。ただし、カーネル デバッグを有効にするターゲット コンピューターに、カーネル モードのデバッガーを接続することができます。詳細については、次を参照してください。[Windows デバッグ](#)します。

必要条件

対象プラットフォーム	Desktop
Header	Rxassert.h (Rxassert.h を含む)
IRQL	<= APC_LEVEL

関連項目

[RxAssert](#)

[Windows デバッグ](#)