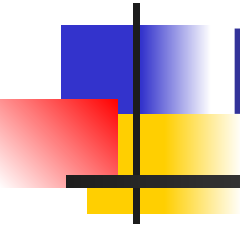


Loading and Unloading Minifilters





Features

- A minifilter may be loaded at any time
 - A minifilter's "altitude" defines its location in the attachment stack
 - Filters have control over what volumes they attach to
 - Filters may support multiple instances (more than one attachment to a given volume)
 - Filters are automatically notified about existing volumes



Features

- A minifilter may be unloaded at any time
 - Filters have control over when they unload
 - If reloaded, will be inserted back in the same frame



Triggering Minifilter Load

- Driver start type of **BOOT**, **SYSTEM** or **AUTO** when the system boots
 - Must use existing load order group definitions for minifilters
 - This is necessary to support proper interoperation with legacy filters
- Service Start request via:
 - "sc start" or "net start" commands
 - service APIs
- An explicit load request via:
 - "fltmc load" command
 - **FltLoadFilter()** API (Kernel mode)
 - **FilterLoad()** API (User mode)



Load Order Groups

- FSFilter Activity Monitor
- FSFilter Undelete
- FSFilter Anti-Virus
- FSFilter Replication
- FSFilter Continuous Backup
- FSFilter Content Screener
- FSFilter Quota Management
- FSFilter System Recovery
- FSFilter Cluster File System
- FSFilter HSM
- FSFilter Compression
- FSFilter Encryption
- FSFilter Physical Quota Management
- FSFilter Open File
- FSFilter Security Enhancer
- FSFilter Copy Protection



Minifilter Startup

- **DriverEntry ()** routine called when driver is loaded
 - Do global initialization
 - Call **FltRegisterFilter ()** API
 - Registers callbacks with Filter Manager
 - Call **FltStartFiltering ()** API
 - Volume enumeration may start before this call returns



Triggering Instance Creation

- At minifilter `FltStartFiltering()` time
 - Existing volumes enumerated
- Volume mount
- An explicit attachment request via:
 - “`fltmc attach`” command
 - `FltAttachVolume()` API (kernel mode)
 - `FilterAttach()` API (user mode)

What controls which instances are created



- Instance definitions in INF file
 - Defines: instance name, altitude, flags
 - Altitude values are defined and maintained by Microsoft
 - Flags contains OR-able bit values:
 - 0x01 = when set suppress automatic attachment
 - 0x02 = when set suppress manual attachment
 - Defines: DefaultInstance
 - Must be specified, used to order filters so mount and instance setup callbacks are sent in the correct order
 - Also used with `FltAttachVolume()` / `FilterAttach()` APIs when no instance name is specified



What controls which instances are created (cont)

- Instance definitions in INF file (cont)
 - Multiple instances may be defined
 - Definitions apply across all volumes
 - Currently uses AddRegistry section
 - A new "Instance" section type will be added to INF files
- **InstanceSetup ()** callback in **FLT_REGISTRATION** structure



Sample Instance Definitions

- From MiniSpy.inf:

```
[Minispy.AddRegistry]
```

```
HKR,"Instances","DefaultInstance",0x00000000,"Minispy - Top Instance"
```

```
HKR,"Instances\Minispy - Bottom Instance","Altitude",0x00000000,"365000"
```

```
HKR,"Instances\Minispy - Bottom Instance","Flags",0x00010001,0x1
```

```
HKR,"Instances\Minispy - Middle Instance","Altitude",0x00000000,"370000"
```

```
HKR,"Instances\Minispy - Middle Instance","Flags",0x00010001,0x1
```

```
HKR,"Instances\Minispy - Top Instance","Altitude",0x00000000,"385000"
```

```
HKR,"Instances\Minispy - Top Instance","Flags",0x00010001,0x1
```



InstanceSetup callback

- If NULL, the instance is always created
- If defined:
 - **FLT_INSTANCE_SETUP_FLAGS** parameter identifies why this instance is being created
 - Automatic
 - Manual
 - Newly mounted volume
 - **VolumeDeviceType** parameter identifies the device type for this volume
 - **FILE_DEVICE_DISK_FILE_SYSTEM**
 - **FILE_DEVICE_NETWORK_FILE_SYSTEM**
 - **FILE_DEVICE_CD_ROM_FILE_SYSTEM**



InstanceSetup callback (cont)

- **VolumeFilesystemType** parameter identifies the file system type for this volume
 - **FLT_FSTYPE_NTFS**
 - **FLT_FSTYPE_FAT**
 - **FLT_FSTYPE_LANMAN**
 - **Etc**
- Instance creation may be failed by returning an error or warning **NTSTATUS**



Triggering Minifilter Unload

- Service stop request via:
 - `"sc stop"` or `"net stop"` commands
 - Service APIs
- An explicit unload request via:
 - `"fltmc unload"` command
 - `FltUnloadFilter()` API (kernel mode)
 - `FilterUnload()` API (user mode)



Controlling Minifilter Unload

- Two mechanisms through **FLT_REGISTRATION** structure
 - **FilterUnload()** callback
 - **FLTFL_REGISTRATION_DO_NOT_SUPPORT_SERVICE_STOP** flag
- FltMgr sets **DriverUnload()** routine in filter
 - It calls, at the appropriate time, any **DriverUnload()** routine the minifilter may have set in its **DriverObject**



FilterUnload callback

- If NULL, the minifilter **cannot** be unloaded
- If defined:
 - Mandatory unloads (via service stop) cannot be failed
 - Non-mandatory unloads (via `FltUnloadFilter()` or `FilterUnload()` APIs) may be failed by returning an error or warning `NTSTATUS`
 - `FLT_FILTER_UNLOAD_FLAGS` parameter identifies reason for unload



FLTFL_REGISTRATION_DO_NOT_SUPPORT_SERVICE_STOP flag

- If set, a minifilter can not be unloaded via a service stop request
- If a `FilterUnload()` callback is defined, the minifilter may be unloaded via the `FltUnloadFilter()` or `FilterUnload()` APIs
- Use this flag if you always need to have the option of failing an unload request

Minifilter's Responsibilities in FilterUnload callback

- Call `FltUnregisterFilter()`, Filter Manager then:
 - Deletes all instances
 - Deletes volume contexts
 - Waits for outstanding Filter references
 - Entries pending in generic work queue
 - `FltObjectReference()/FltObjectDereference()`
 - When this returns all instances have been deleted
- Do global cleanup:
 - Delete global EResources
 - Free global memory and delete lookaside lists
 - Unregister global callbacks
 - Timer, Process or Thread notification callbacks
- Minifilter will be unloaded if a success NTSTATUS is returned



Triggering Instance teardown

- A minifilter being unloaded
- A volume being dismounted
- An explicit detach request via
 - `fltmc detach` command
 - `FltDetachVolume()` API (kernel mode)
 - `FilterDetach()` API (user mode)



Controlling Instance Teardown

- In **FLT_REGISTRATION** structure:
 - **InstanceQueryTeardown()** callback
 - **InstanceTeardownStart()** callback
 - **InstanceTeardownComplete()** callback

InstanceQueryTeardown Callback



- Only called for explicit detach requests via `FltDetachVolume()` or `FilterDetach()`
 - **Not** called for `FilterUnload()`
 - **Not** called for volume dismount
- If NULL, instance **cannot** be torn down via an explicit detach request
- If defined:
 - May be failed by returning an error or warning NTSTATUS
 - If a success NTSTATUS is returned, teardown starts immediately

InstanceTeardownStart Callback

- May be NULL, instance is still torndown
- If defined:
 - Must:
 - Pass on or complete pended preOperation IOs
 - Use `FltCompletePendedPreOperation()`
 - Guarantee you won't pend any new IOs (see `FltCbdoqXXX()` routines)
 - Complete pended postOperation IOs
 - Use `FltCompletePendedPostOperation()`
 - May:
 - Close opened files
 - Make worker threads start doing minimal work
 - Cancel filter initiated IOs
 - Stop queuing new work items
 - No **new** operation callbacks are being sent to the minifilter, may see operation callbacks for operations started before teardown was initiated

Outstanding Operation Callbacks



- Any currently executing preOperation callback continues normal processing
- Any currently executing postOperation callback continues normal processing
- Any IO that has completed the preOperation callback and is waiting for a postOperation callback may be “drained” or “canceled”



“Draining” Operation Callbacks

- Is a PostOperation callback that is called asynchronously from the actual operation being completed.
 - Always called at a safe IRQL
- **FLTFL_POST_OPERATION_DRAINING** in “Flags” parameter is set
- Receives “fake” CallbackData structure
 - Minimally initialized
 - Contains valid **FLT_IO_PARAMETER_BLOCK** (Iopb)
 - **IoStatus.Status** contains **STATUS_FLT_POST_OPERATION_CLEANUP**
- Receives fully populated **FLT_RELATED_OBJECTS** structure
- Must:
 - Perform minimal work
 - Cleanup context from preOperation callback
 - Return **FLT_POSTOP_FINISHED_PROCESSING**
- Must Not:
 - Restore swapped data buffers
 - Attempt to defer the operation in any way
- If drained, will not receive a normal postOperation callback

“Cancelling” Operation Callbacks



- If buffers have been swapped for a given operation, that operation is not drainable
 - Instead, Filter Manager attempts to cancel the operation
- After canceling, Filter Manager waits for the operation to complete



Minifilter Generated IOs

- These IOs will continue normal processing
- Minifilter should cancel any long lived IOs
 - Oplocks, directory change notifications, etc.
- Instance teardown will wait for all filter generated IOs to complete

InstanceTeardownComplete callback



- May be NULL, instance is still torndown
- If defined:
 - When called, all outstanding IO operations have been completed or drained
 - **WARNING:** This routine will not be called if:
 - There are any outstanding pended operations
 - There is any outstanding filter generated IO
 - The unload request will look like it has hung
 - Must
 - Close any files that are still open
 - Referencing an Instance (with FltObjectReference) does **not** prevent this routine from being called



Final Cleanup of Instance

- Waits for outstanding Instance references
 - Waits for deferred IO work items to complete
 - Waits for any other references on the instance
 - `FltObjectReference()/FltObjectDereference()`
- All remaining contexts deleted
- The instance is now gone



Debugging Aids

- In checked builds:
 - Lots of internal asserts
 - When your minifilter unloads, the following is reported on the debugger screen:
 - Contexts you have forgotten to release
 - Files opened by `FltCreateFile()` that you have forgotten to close
- Try `!fltkd.help` in debugger for Filter Manager debugger extensions