

# → Filter Manager



**Rajeev Nagar**

Lead Program Manager

Core File Services

rajeevn@microsoft.com

## → Filter Manager Overview

- Legacy Filtering Mechanisms & Issues
- Filter Manager Benefits
- Filter Manager Architecture
- Features / Functionality
- Project Status & Release Plans

## → Question and Answer

→ Many products use a ***file system filter***

- Historically, caused much customer pain
- Issues include stability, performance, & interoperability

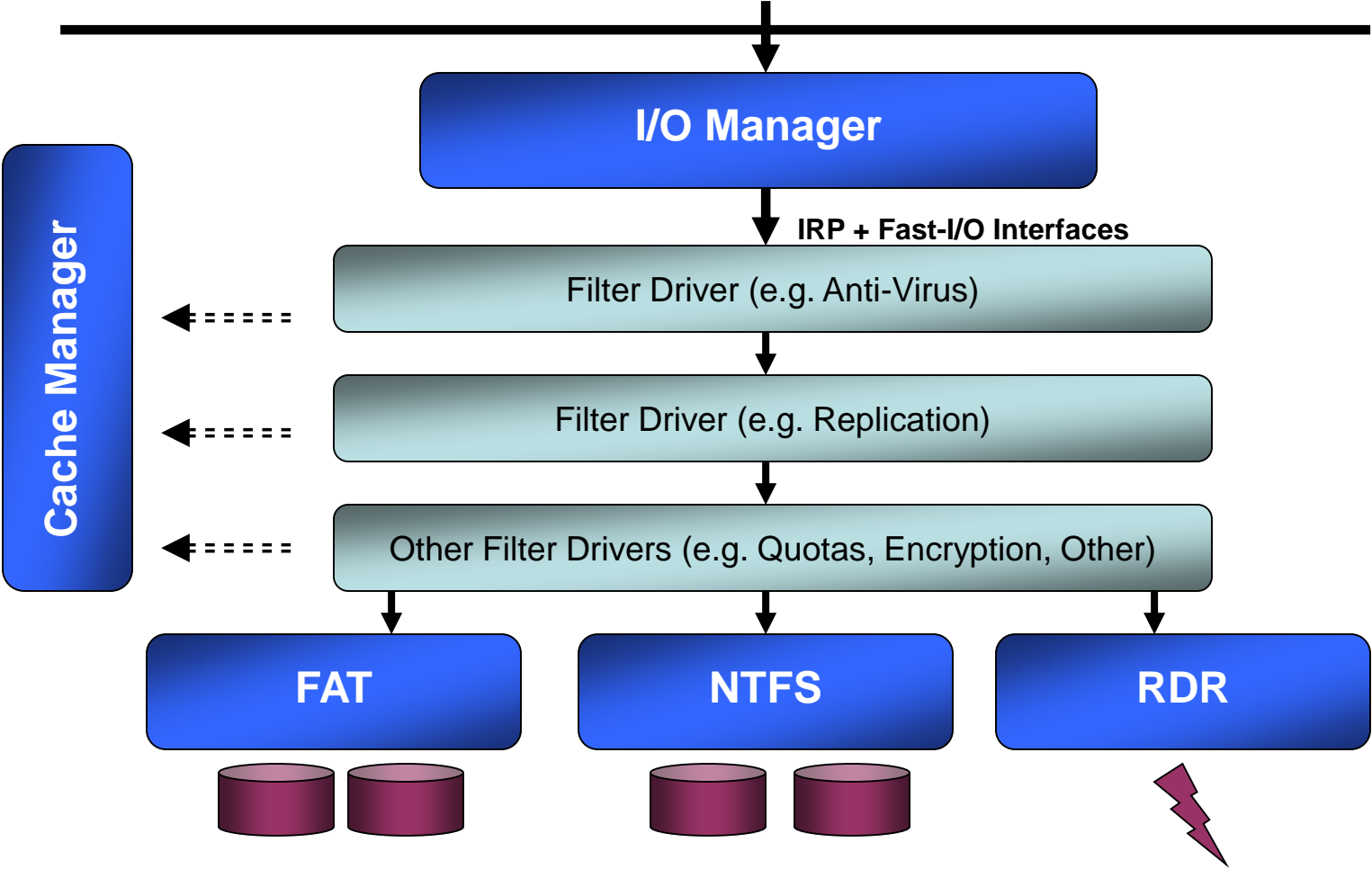
→ Examples of products with filter drivers:

- Antivirus products
  - Filter watches I/O to and from certain file types (.exe, .doc, etc.) looking for virus signatures
- File replication products
  - File-system-level mirroring
- System Restore
  - Backs up system files when changes are about to be made so that the user can return to the original state
- Many more...
  - Quota products, backup agents, undelete, encryption products, etc.

→ We've come a long way in addressing issues with filter drivers:

- Improved documentation
- Plug-fests
- AV certification program
- However, 7% of OCA crashes are still attributed directly to 3<sup>rd</sup> party filter drivers

NtReadFile() / NtWriteFile(), ...



- Kernel-mode drivers
- Attach to locally mounted volumes (e.g. C: )and/or to redirectors (e.g. RDR/WebDAV)
  - Attach to file system driver control device objects
  - “Walk” list of mounted volumes in an unsafe manner
  - Intercept mount volume requests
  - Poll for redirector load
- Intercept IRPs and fast-i/o requests issued by I/O Manager to File System Driver (FSD)
- Perform filter-specific processing prior to dispatching request to FSD and/or post-completion of request processing by FSD
  - Often impact control flow
  - Often message returned data/metadata
- May generate new I/O Request Packets (IRPs) as part of processing

→ Reliability

- A bug in your driver will cause a blue-screen or deadlock

→ Performance

- You're on the path of all I/O

→ Development and maintenance cost

- Complex code
- Hard to develop, test, debug, maintain
- Must revise with each OS version and/or service packs

→ Not your core competency

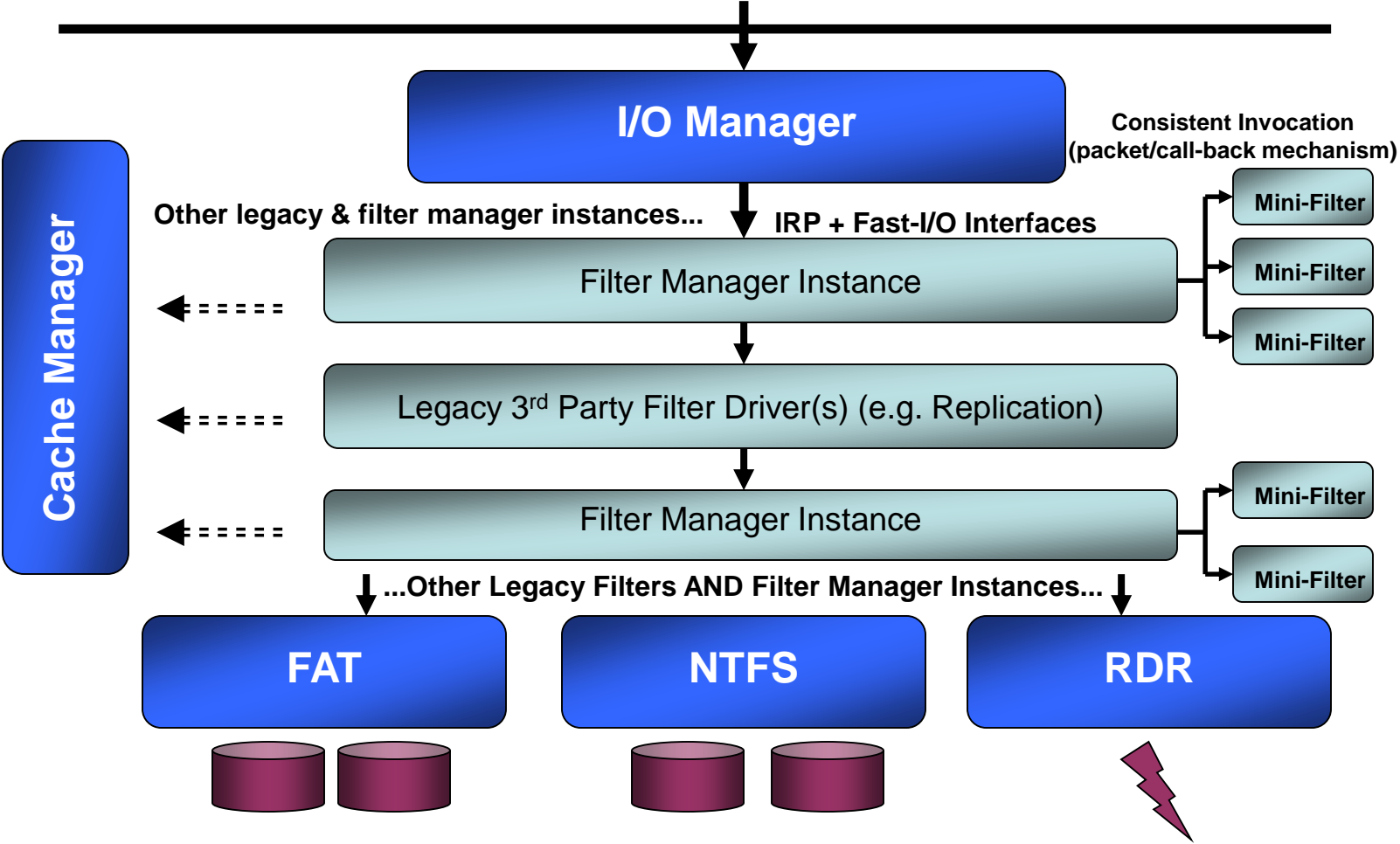
→ Not your core value add to the customer

- Many problems with current model (legacy filters)
  - Poor control over stack ordering (load order groups)
  - No unload support
  - Stack limit issues
  - Complex interfaces (“fast-io” and IRPs)
  - Reentrancy issues
  - Inefficiencies due to redundant work in filters
  - Ad-hoc (reinvented) methods for common tasks
    - Attach to mounted volumes and redirectors
    - Generate IRPs
    - Obtain file/path name
    - Maintain filter contexts per object (volume, stream, other)
    - Manage buffers
  - Substantial Performance Degradation
- Expect even more problems with new functionality e.g. TxF (Transactional NTFS) support

- Callback based rather than chained dispatch routines
  - Helps solve many stack overflow issues
  - Ability for system to add new operation types w/o breaking existing filters
- Uniform interface for all operations
  - Fast I/O, IRP, callbacks are all intercepted in the same manner
- Isolation from gnarly IRP processing rules
  - Filter Manager does this processing on behalf of the filters
- Dynamic load/unload (Ability to unload)
- Non re-entrant filter initiated I/O
- Efficient pass through
- Deterministic Load Order (ease interoperability/testing)
- Efficient context management
- A library of value-add APIs
  - File name management
  - IO cancellation and queuing
  - Buffer Management
- Efficient and secure user/kernel communication
- Support for TxF



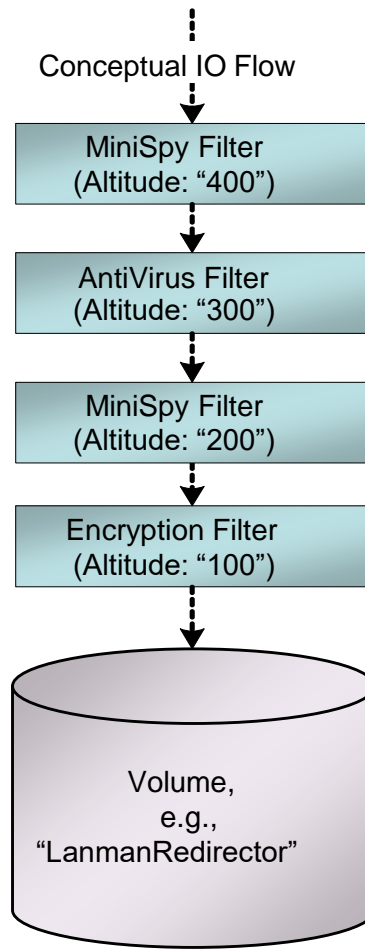
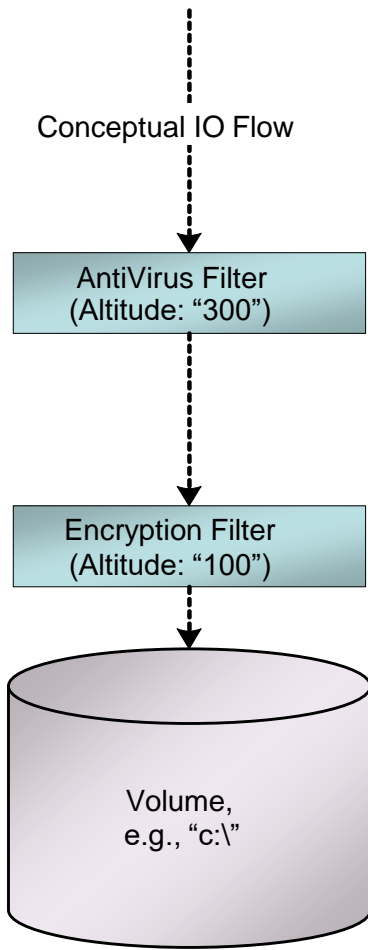
NtReadFile() / NtWriteFile(), ...



- Legacy file system filter
- Manages the complexity of I/O system through new interfaces and library routines
- Has kernel and user-mode interfaces
- Supports multiple loaded mini-filters and multiple instances per volume
- Coexists with other legacy filter drivers (until they are all phased out)

- Just another kernel mode driver
- Register with filter manager in DriverEntry()
- Leverage filter manager to attach to volumes (local and remote)
- Utilize filter manager to process only I/O operations of interest (specify appropriate callbacks)
- Determine control flow easily and efficiently
- Utilize available library functions for commonly required functionality such as:
  - obtaining file name/path
  - synchronize post-processing of I/O operations
  - queue and manage per-object context
  - other ...
- Be able to unload/upgrade driver in field w/o requiring reboot
- Leverage filter manager provided efficient user/kernel communication mechanism
- Interoperate correctly with transactional file system support

- Mini-filter registers only for operations in which it is interested through FLT\_REGISTRATION structure
  - Register pre-operation callback and/or post-operation callback
  
- FLT\_CALLBACK\_DATA replaces the IRP
  - FLT\_CALLBACK\_DATA->Iopb contains parameters for this operation, similar to IO\_STACK\_LOCATION
  - No management of FLT\_CALLBACK\_DATA needed, i.e., no more IoSkipCurrentIrpStackLocation(), IoSetCompletionRoutine()
  - Common structure for all types of operations: Irp, FastIo, and FsFilter



- *Instance*: Instantiation of a filter on a volume at a particular altitude
- Support multiple instances of a mini-filter on a volume
- *Altitude* determines relative stack position

## → FltRegisterFilter()

- Register with Filter Manager
- All callback information in FLT\_REGISTRATION structure

## → FltStartFiltering()

- Begin enumeration of existing volumes in system
- InstanceSetup() callback is called for mini-filter to see if it wants to attach

- Through `FilterUnload()` callback, mini-filter is allowed to accept or deny the unload request
  
- To unload, Filter Manager synchronizes the safe removal of all mini-filter instances through a series of notifications
  - `InstanceQueryTeardown()` – allows filter to fail the teardown request for given instance
  - `InstanceTeardownStart()` – Notifies filter that teardown process is beginning for given instance
  - `InstanceTeardownComplete()` – Notifies filter teardown process has finished for given instance

→ Mini-filter communicates control flow choice through callback return value

→ In pre-operation, filter can:

- Pass through the operation – `FLT_PREOP_SUCCESS_NO_CALLBACK`
- Ask to see operation completion – `FLT_PREOP_SUCCESS_WITH_CALLBACK`
- Pend the operation – `FLT_PREOP_PENDING`
- Ask to have completion synchronized to current thread – `FLT_PREOP_SYNCHRONIZE`
- Complete the operation – `FLT_PREOP_COMPLETE`



- In postOperation, mini-filter can:
  - Do its work and continue completion processing –  
FLT\_POSTOP\_FINISHED\_PROCESSING
  - Pend the completion processing –  
FLT\_POSTOP\_MORE\_PROCESSING\_REQUIRED
  
- For pended IOs, continue processing with  
FltCompletePendedPreOperation() or  
FltCompletePendedPostOperation()

- Queuing Support
- Buffer Manipulation (locking/swapping)
- Context Management
- File Name Management
- I/O Generation

- Provides common functionality for user-mode applications that work with filter drivers
- Application must link with filterlib.dll
- Include header files fltUser.h and fltUserStructures.h
- Load and unload mini-filters
  - FilterLoad(), FilterUnload()
- Open handles to filters or instances to get information
  - FilterCreate(), FilterInstanceCreate()
  - FilterGetInformation(), FilterInstanceGetInformation()

→ Enumerate filters, instances, and volumes

- FilterFindFirst(), FilterFindNext()
- FilterVolumeFindFirst(), FilterVolumeFindNext()
- FilterInstanceFindFirst(), FilterInstanceFindNext()
- FilterVolumeInstanceFindFirst(), FilterVolumeInstanceFindNext()

→ Open handle to communication port

- FilterConnectCommunicationPort()

→ Add and remove mini-filter instances

- FilterAttach(), FilterAttachAtAltitude()
- FilterDetach()

→ Command line utility for common filter management operations

- Load and unload mini-filters
- Attach/detach mini-filters to/from volumes
- Enumerate mini-filters, instances, volumes

→ “fltmc help”

- Displays help information for utility

## → Fltkd.dll debugger extension

- !fltkd.help will list all the available commands
- For more specific help on a single command, issue that command with no parameters
- !cbd: Filter Manager equivalent to !irp
- !volumes, !filters: List all volumes/filters in system
- !volume, !filter, !instance: Give detail on a specific object
- Ignore version warning, turn off with “.noverion” command

## → Run with debug fltmgr.sys

- Lots of ASSERT to catch common errors

- Enable through verifying mini-filter via Driver Verifier with “I/O Verification“ option
- Verification starts when a filter registers with the Filter Manager
- Validates all Filter Manager API calls by mini-filter
  - Validates parameters and calling context
- Verifies all the special return values from mini-filter’s pre/post callback routines
- Ensures mini-filter changed the parameters in the callback data in a coherent/consistent manner
- More to come in future

- All existing Microsoft filters converted to minifilter model for Longhorn
- Minifilters and Legacy filters will coexist – however, goal is to strongly encourage all filters to be converted to minifilter model
- Filter Manager to be released in
  - Longhorn
  - Windows Storage Server
  - Windows Server 2003 SP1
  - WinXP SP2
  - Support for Windows 2000 (release plans being finalized)
- IFS Kit update for Windows Server 2003 SP1, Windows XP Service Pack 2 and the Longhorn driver kit will contain filter manager libraries, headers, and samples
- For more information, contact [rajeevn@microsoft.com](mailto:rajeevn@microsoft.com)



## → **Port your legacy filter to the mini-filter model**

- Send us feedback on the filter manager including any additional support that may benefit your product/mini-filter

***Microsoft***<sup>®</sup>

**© 2003 Microsoft Corporation. All rights reserved.**

**This presentation is for informational purposes only. Microsoft makes no warranties, express or implied, in this summary.**

