

# I/O System (II)

# Roadmap for This Lecture

- Windows Driver Foundation
  - Kernel-Mode Driver Framework
  - User-Mode Driver Framework
- Plug-and-Play Manager
- Power Manager
- FileMon: troubleshooting of I/O system
- Labs

# Windows Driver Foundation

- WDF simplifies driver development with two frameworks:
  - Kernel-Mode Driver Framework (KMDF)
    - Supports Win 2000 SP4 and later
    - Simple interface to WDM
    - Call into KMDF library to perform generic work on hardware
  - User-Mode Driver Framework (UMDF)
    - Supports Win XP and later
    - Enables USB-based and high-latency protocol buses drivers to be implemented as user-mode drivers
    - UMDF drivers can crash and not affect the system stability
    - Written in C++ COM-like classes → lower barrier

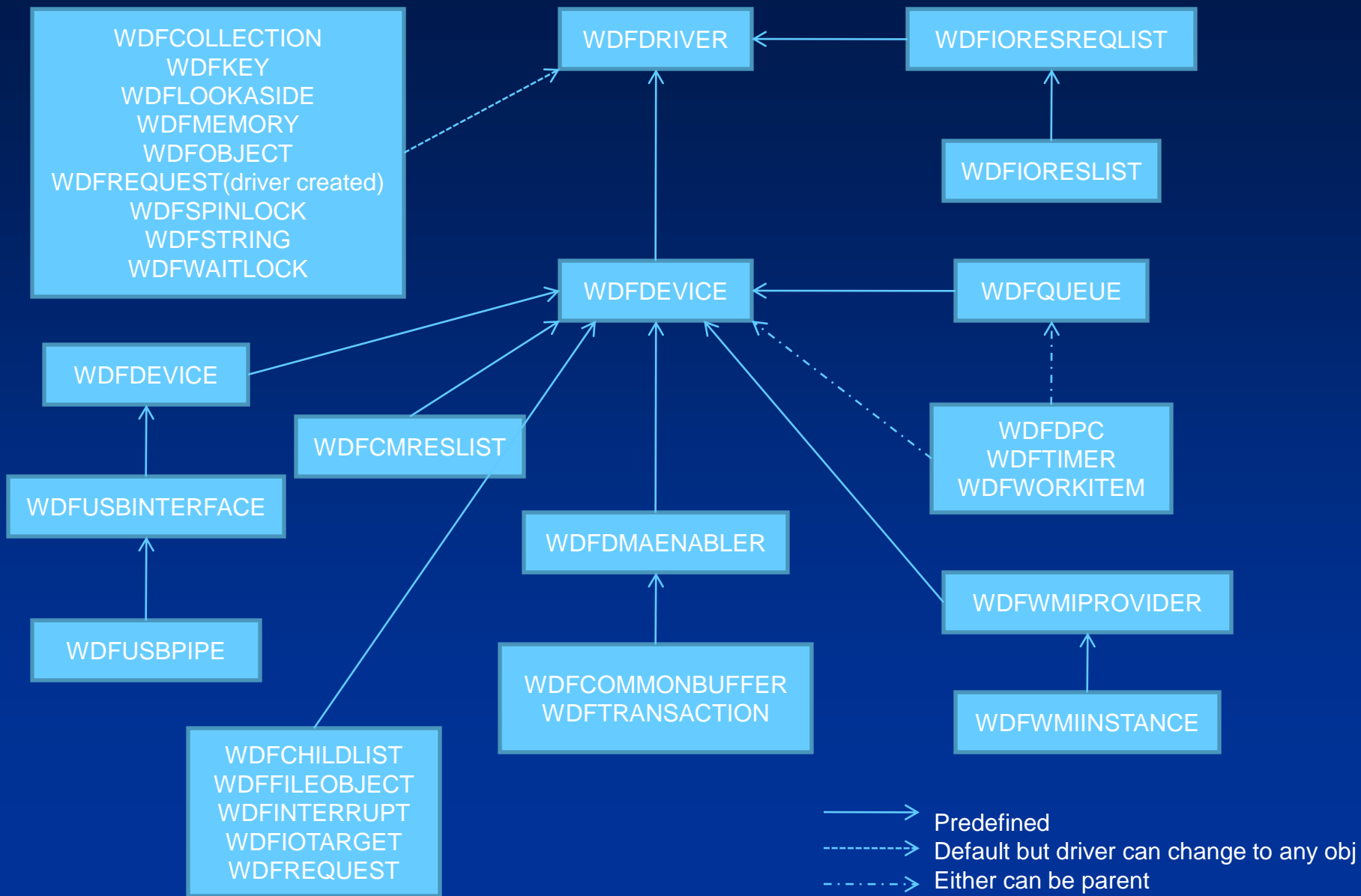
# KMDF Drivers – Structure and Operations

- KMDF supports all WDM-complying drivers if they:
  - Perform standard I/O processing and IRP manipulation
  - Use Windows kernel API directly
  - Provide their own dispatch functions instead of relying on port or class driver and other clients
- Has following functions:
  - An initialization routine
    - Non-PnP driver creates first device object here
  - An add-device routine
    - *EvtDriverDeviceAdd* callback
  - One or more *EvtIo\** routines (optional)
    - Handles requests from device queues
- KMDF drivers are event based
  - Events are not synchronization based but internal
  - Callbacks are optional → default generic action can be used

# KMDF Drivers – Data Model

- Object-based --- but doesn't use object manager
- KMDF manages objects internally
- Framework provides routines to perform operations on objects
  - *WdfDeviceCreate* function creates a device
  - *Get/Set* APIs (never fails)
  - *Assign/Retrieve* APIs (can fail)
- Part of a hierarchy
  - Most objects have a parent
  - Root is WDFDRIVER
  - Next is WDFDEVICE → created by *WdfDeviceCreate*
  - Objects are opaque
  - Hierarchy affects object's locality and life time
  - Object context areas: specific data about an object outside the framework

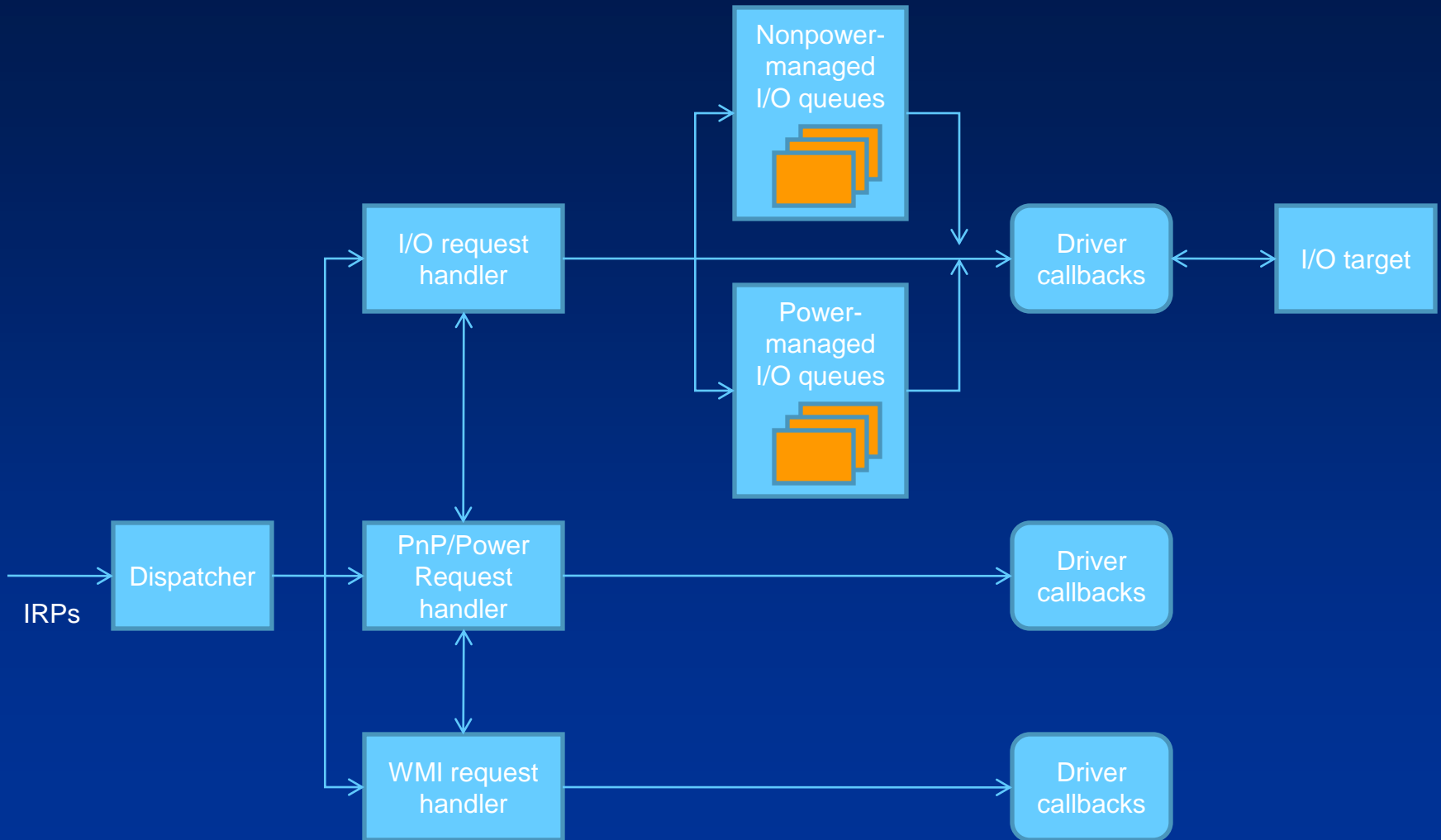
# KMDF Object Hierarchy



# KMDF Object Attributes

Attribute	Description
ContextSizeOverride	Size of the object context area
ContextTypeInfo	Type of the object context area
EvtCleanupCallback	Callback to notify the driver of the object's cleanup before deletion (references may still exist)
EvtDestroyCallback	Callback to notify the driver of the object's imminent deletion (references will be 0)
ExecutiveLevel	Describes the max IRQL at which the callbacks will be invoked by KMDF
ParentObject	Identifies the parent of this object
Size	Size of the object
SynchronizationScope	Specifies if the callbacks should be synchronized with the parent, a queue or device or nothing

# KMDF I/O Model





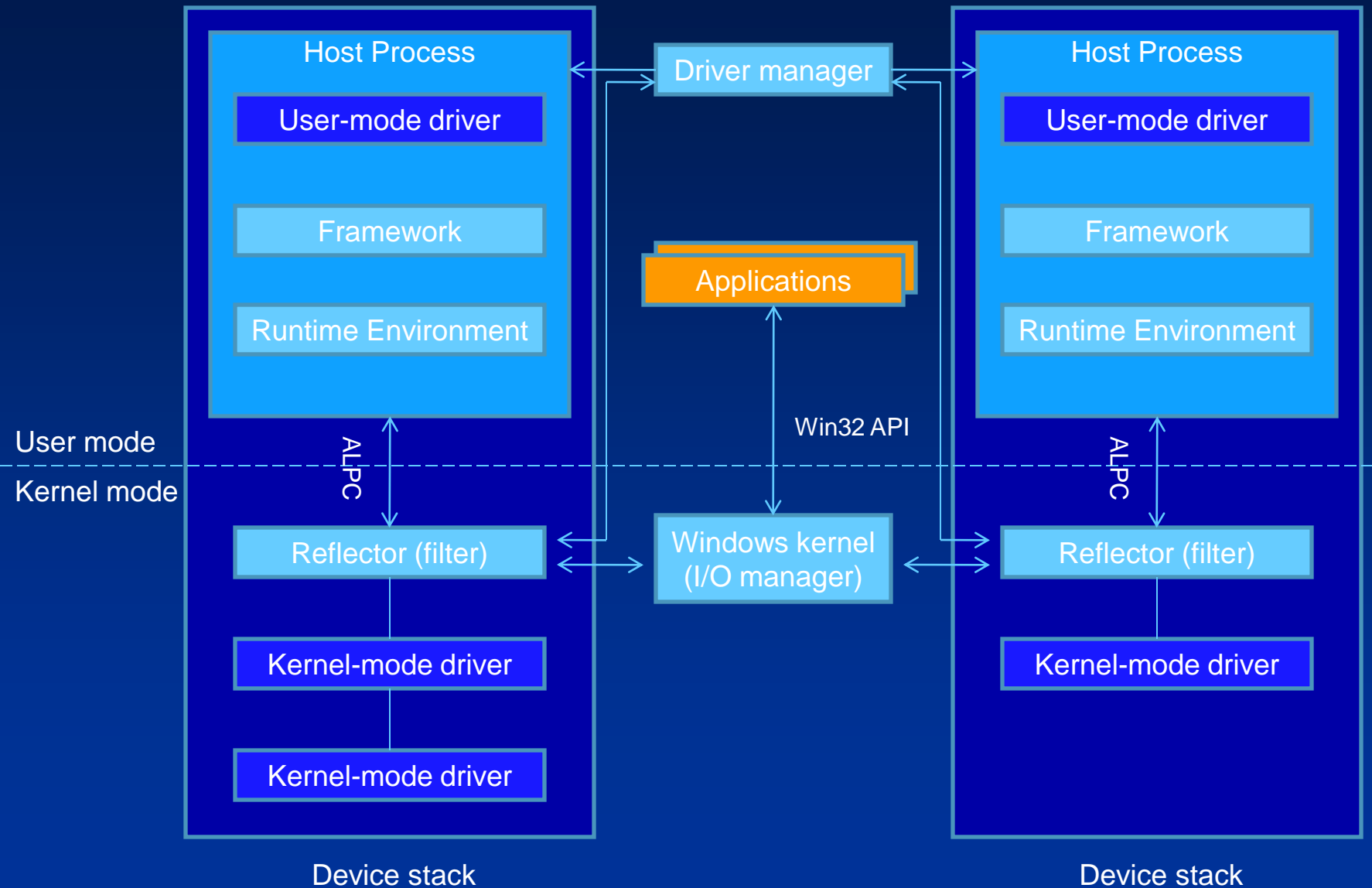
# KMDF I/O Model

- **I/O request handler:** processes standard device operations
- **PnP and power request handler:** processes PnP and power events and notifies other drivers of state changes
- **WMI handler:** handles tracing and logging
- IRP finished but request not fully processed:
  - Bus/function drivers: complete IRP with `STATUS_INVALID_DEVICE_REQUEST`
  - Filter drivers: forward to lower driver
- Queues are `WDFQUEUE` object containing `WDFREQUEST` objects, with following options:
  - Register callbacks to queue
  - Power management state
  - Dispatch method
  - Can accept zero-length buffers or not

# UMDF Overview

- Support protocol device classes
  - Same standardized, generic protocol and offer specialized functionality on it
  - E.g. IEEE 1394 (firewire), USB, bluetooth and TCP/IP
  - Portable music players, PDAs, cell phones, webcams
  - SideShow-compatible devices (auxiliary displays)
  - Windows Portable Device (WPD) framework → USB removable storage
- Same driver programming model as KMDF, but
  - User-mode objects
  - No direct handling of interrupts, DMA, nonpaged pools and strict timing
  - Can't be on kernel stack, or be the client of another driver or kernel itself
- Run in a driver host process:
  - User-mode driver
  - Driver framework (DLL containing COM components for each object)
  - Run-time environment (for I/O dispatching, driver loading, stack management, communication and thread pool)

# UMDF Architecture



# The PnP Manager

- Foundation: industry standards for *enumeration* and identification of devices attached to a bus
- Windows PnP supports:
  - Automatically recognition of installed devices, during boot, addition and removal of devices
  - Hardware *resource arbitration*
  - Loading the right drivers
  - Application and driver mechanisms for detection hardware configuration changes
  - Storage device state
  - Network devices
- Level of PnP support:

Type of Device	Type of Drivers	
	Plug & Play	Non-Plug & Play
Plug & Play	Full PnP	No PnP
Non-Plug & Play	Possible partial PnP	No PnP

# Resource Arbitration

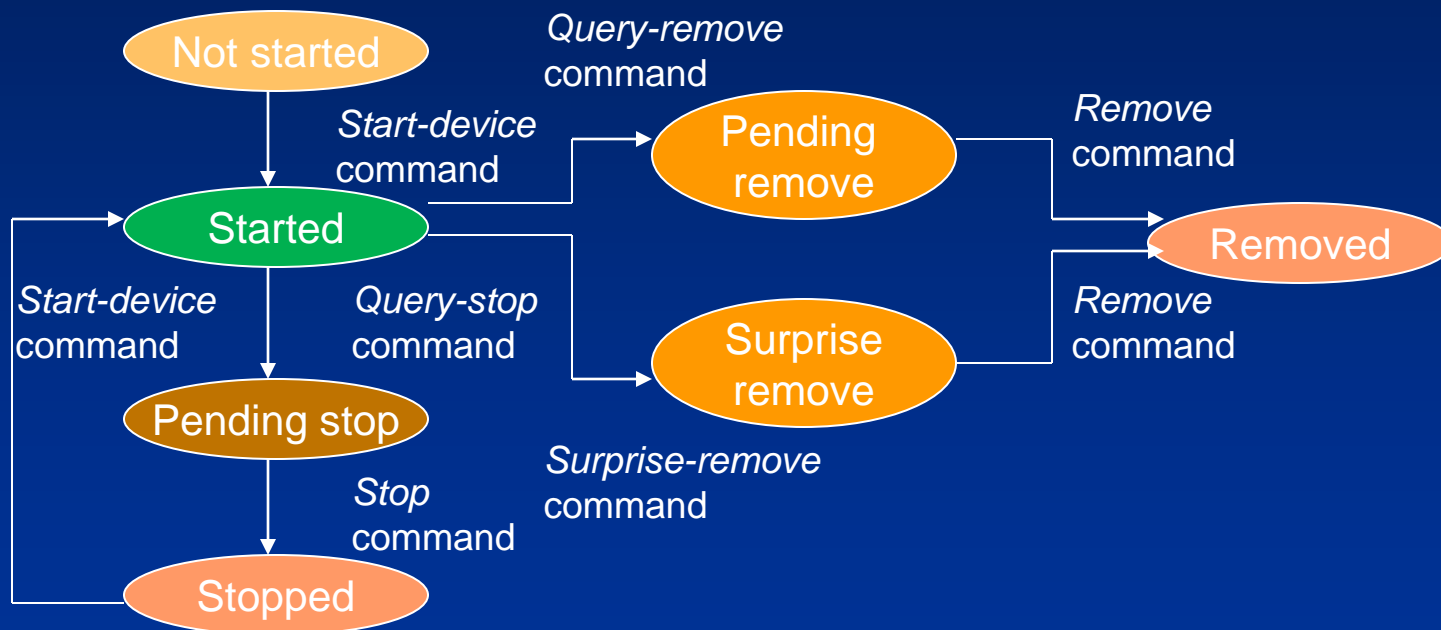
- Devices require system hardware resources to function (e.g. IRQs, I/O ports)
- The PnP Manager keeps track of hardware resource assignments
- If a device requires a resource that's already been assigned, the PnP Manager tries to reassign resources in order to accommodate
- Example:
  1. Device 1 can use IRQ 5 or IRQ 6
  2. PnP Manager assigns it IRQ 5
  3. Device 2 can only use IRQ 5
  4. PnP Manager reassigns Device 1 IRQ 6
  5. PnP Manager assigns Device 2 IRQ 5

# Driver support for PnP

- Driver must implement:
  - PnP dispatch routine
  - Power management routine
  - Add-device routine
- Bus driver supports different PnP requests
  - Provides descriptions of devices → resource requirements
- PnP manager sends *start-device* command to driver's PnP dispatch routine
- Once started, PnP manager can send driver more commands:
  - *query-remove* command
  - *remove* command
  - *query-stop* command (resource re-assignment)
  - *stop* command (resource re-assignment)
- Commands guides the device through different states

# Plug and Play (PnP) State Transitions

- PnP manager recognizes hardware, allocates resources, loads driver, notifies about config. changes



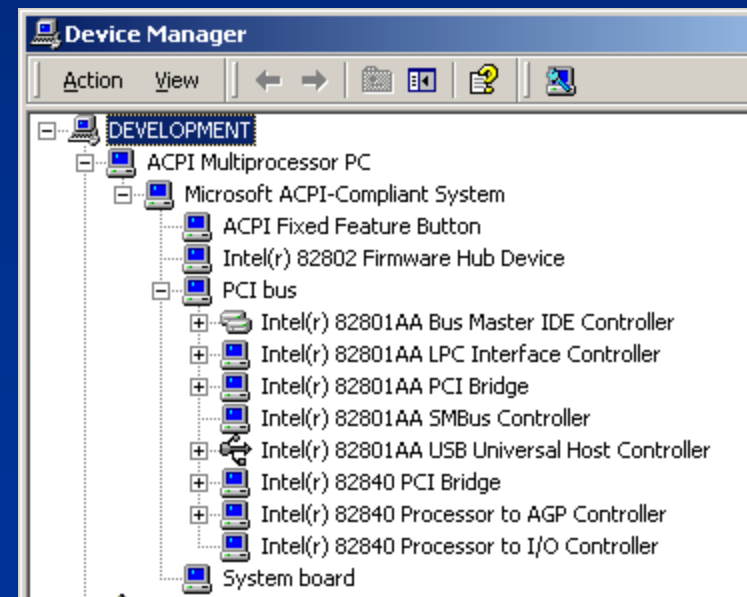
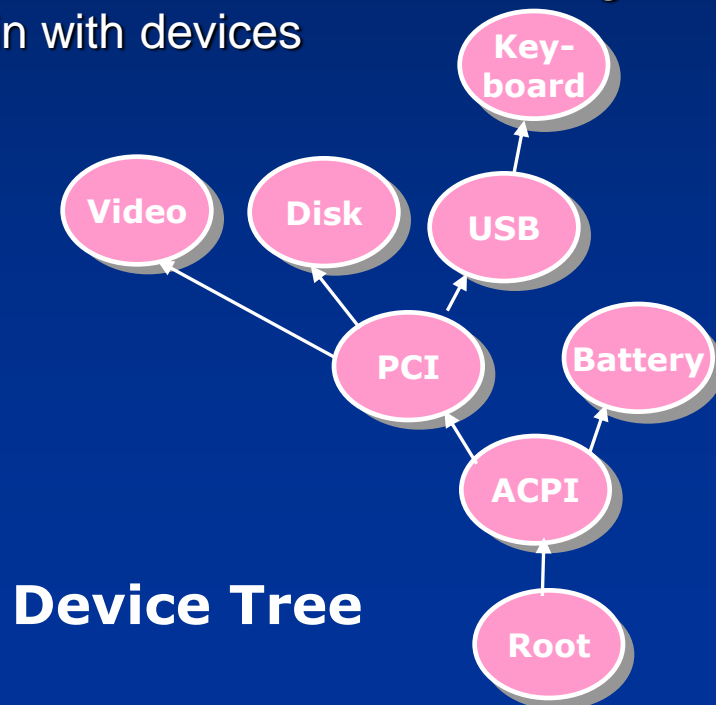
# Driver Loading & Initialization – Explicit

- Explicit loading
  - HKLM\SYSTEM\CurrentControlSet\Services
  - Specify start value:
    - boot-start(0), e.g. system bus drivers, boot file system driver
    - system-start(1), e.g. serial port driver
    - auto-start(2), e.g. non-PnP driver/file system driver
    - demand-start(3), e.g. network adaptor driver
  - Use Group and Tag values to control order of loading
  - Value 0 means OS loader loads the driver
  - Value 1 means I/O manager loads the driver

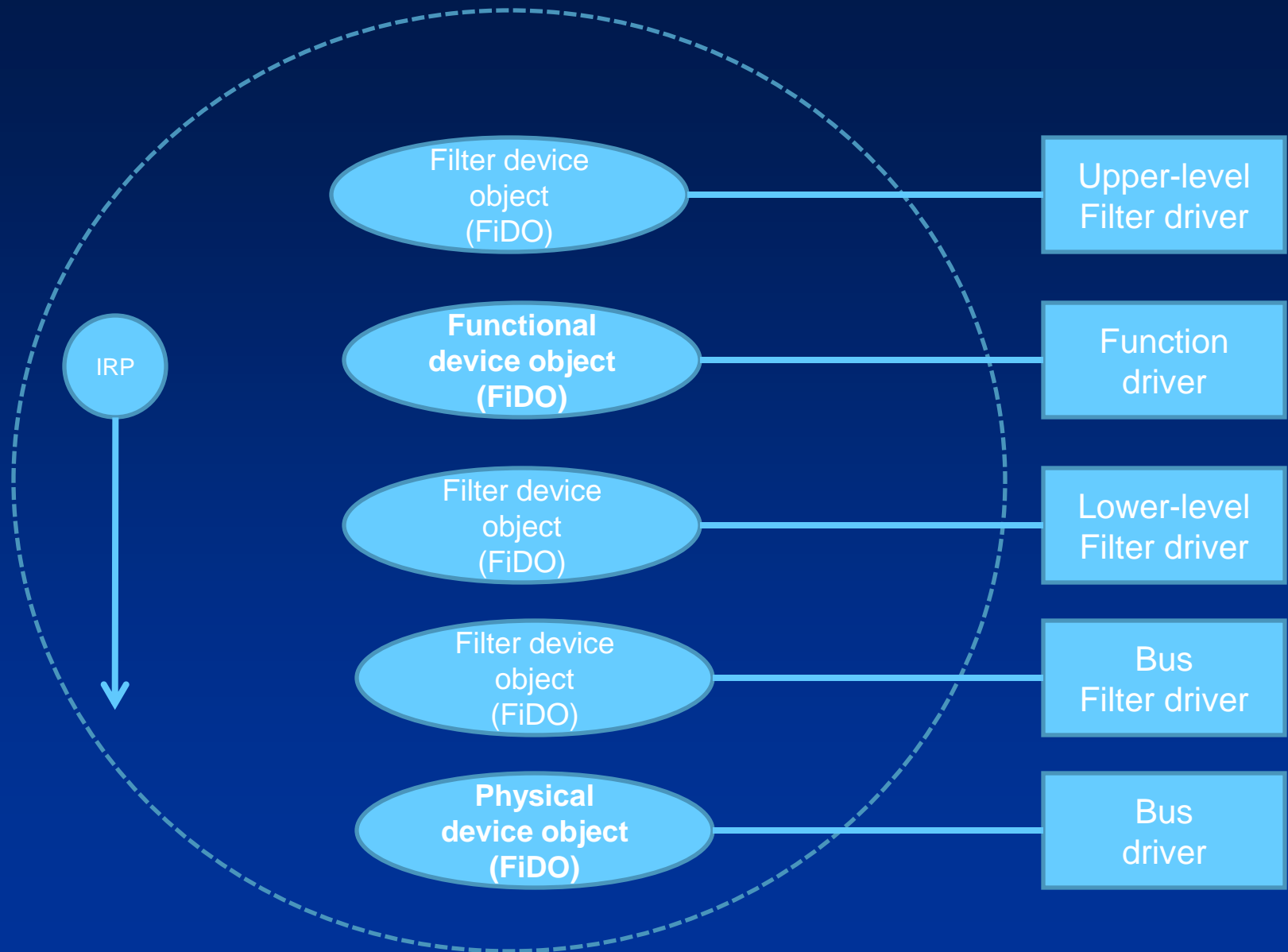


# Driver Loading & Initialization – Enumeration-based

- Enumeration is recursive, and directed by bus drivers
  - Bus drivers identify device on a bus
  - PnP manager initializes drivers for the device
  - Driver can detect additional devices
- As buses and devices are registered, a device tree is constructed, and filled in with devices

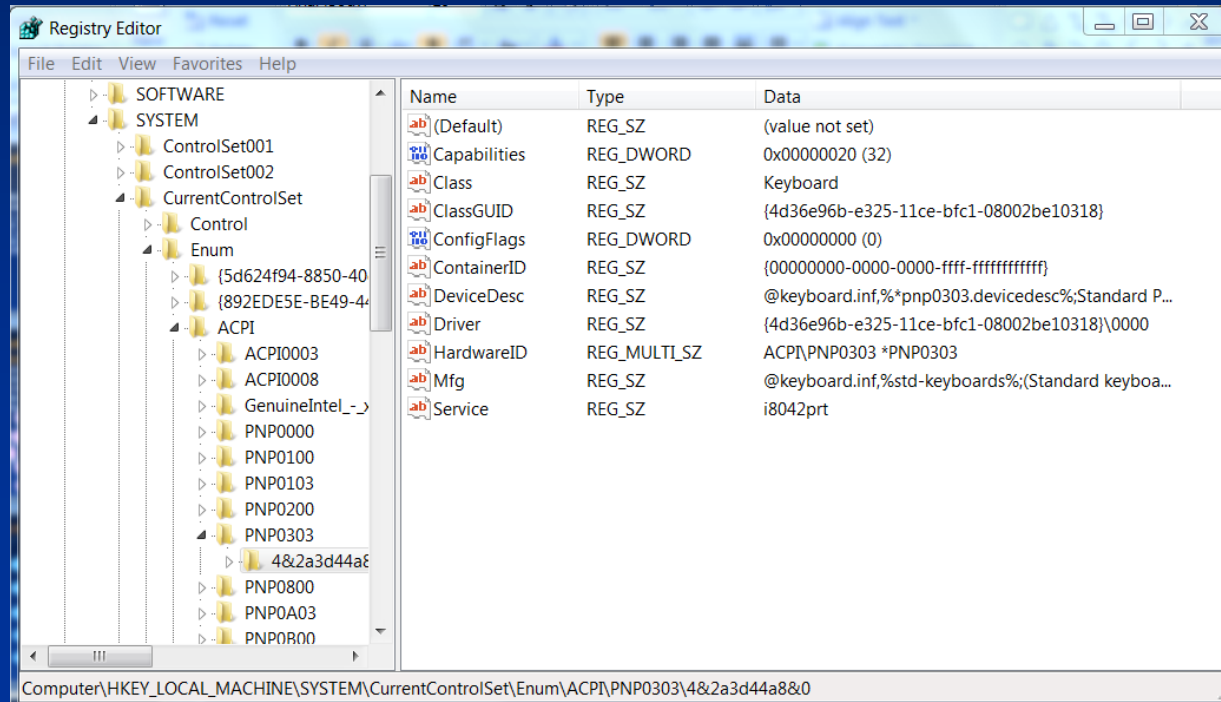


# Devnodes Internals



# Devnode Driver Loading

- Device enumeration produces:
  - *vendor ID + product ID* → *device ID*
- PnP Manager queries the bus driver for *instance ID*
- *device ID + instance ID = device instance ID (DIID)*
- Device key located in `HKML\SYSTEM\CurrentControlSet\Enum`



# Devnode Driver Loading (cont'd)

- Device's key includes Service and ClassGUID
- ClassGUID allows PnP manager to locate device's class key in HKLM\SYSTEM\CurrentControlSet\Control\Class
- Device enum key + Class key → load the necessary drivers
- Driver loading order:
  - Lower level filter drivers specified in LowerFilter value of the enum key
  - Lower level filter drivers specified in LowerFilter value of the class key
  - Function driver specified by the Service value in the enum key
  - Upper level filter drivers specified in the UpperFilter value of the enum key
  - Upper level filter drivers specified in the UpperFilter value of the class key

# Driver Installation

1. Bus driver informs PnP manager of a device it enumerates using a DIID
2. PnP manager checks registry for presence of a driver, if not found, informs the user-mode PnP manager of the DIID
3. User-mode PnP manager launches Rundll32.exe (hardware installation wizard)
4. H.I.W. uses Setup and CfgMgr APIs to locate the INF files of the driver that corresponds to the device
5. Imports package into driver store
6. System installs the driver using Drvinst.exe
7. User-mode PnP manager checks the system's driver signing policy
  - a) if block or warn unsigned driver → locates .cat files containing signatures
  - b) Decrypts signature and compare the hash with the hash of the driver file to be installed.

# Power Manager

- System power state definitions based on the Advanced Configuration and Power Interface (ACPI)

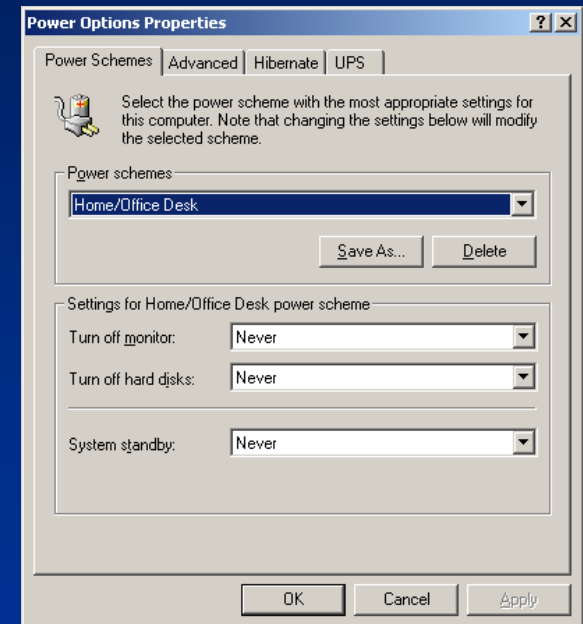
State	Power Consumption	Software Resumption	HW Latency
S0 (fully on)	Maximum	Not applicable	None
S1 (sleeping)	Less than S0, more than S2	System resumes where it left off (returns to S0)	Less than 2 sec.
S2 (sleeping)	Less than S1, more than S3	System resumes where it left off (returns to S0)	2 or more sec.
S3 (sleeping)	Less than S2, processor is off	System resumes where it left off (returns to S0)	Same as S2
S4 (hibernating)	Trickle current to power button and wake circuitry	System restarts from hibernate file and resumes where it left off (returns to S0)	Long and undefined
S5 (fully off)	Trickle current to power button	System boot	Long and undefined

# The Power States

- A system must have an Advance Configuration and Power Interface (ACPI)-compliant BIOS for full compatibility (APM gives limited power support)
- There are different system power states:
  - On (S0)
    - Everything is fully on
  - Standby (S1, S2 and S3)
    - Intermediate states
    - Lower standby states must consume less power than higher ones
  - Hibernating (S4)
    - Save memory to disk in a file called hiberfil.sys in the root directory of the system volume
  - Off (S5)
    - All devices are off
  - Devices have their own 4 power states: D0 through D3
    - D0 fully on, D3 fully off, D1 and D2 up to individual drivers and devices

# Power Manager Operation

- A number of factors guide the Power Manager's decision to change power state:
  - System activity level
  - System battery level
  - Shutdown, hibernate, or sleep requests from applications
  - User actions, such as pressing the power button
  - Control Panel power settings





# Driver and Application Control of Device Power

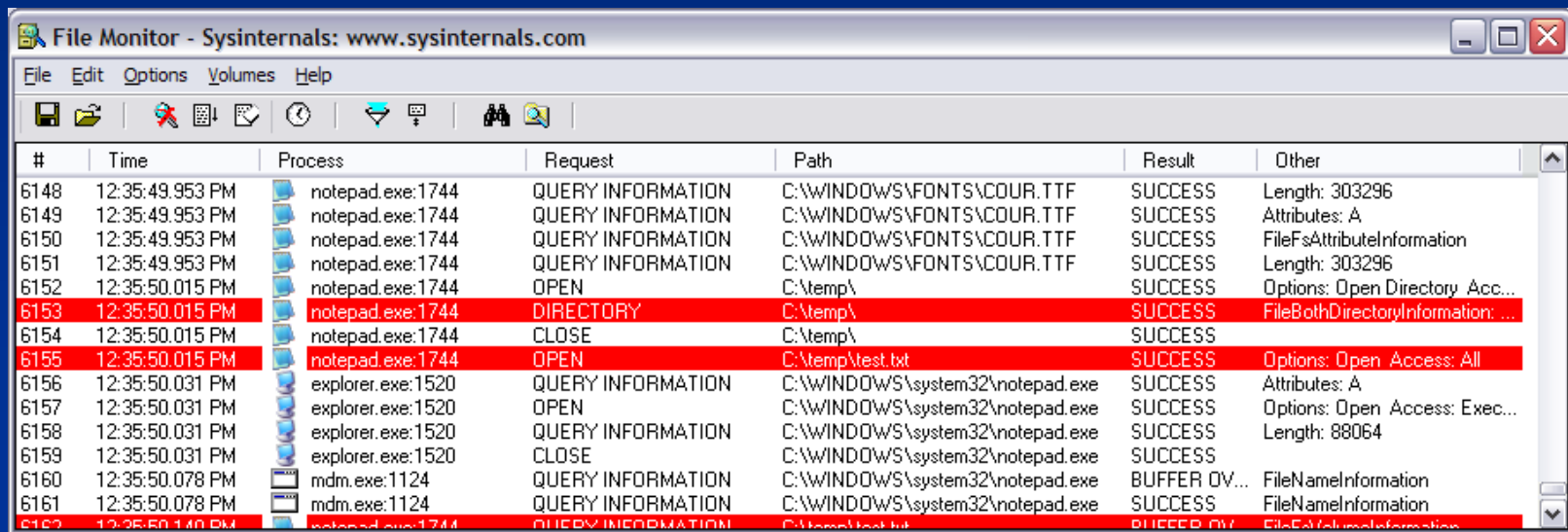
- The system can go into low power modes, and it *notifies* the driver designated as the device power-policy owner
  - The policy owner decides the power state of the device
  - Device drivers manage their own power level
    - Only a driver knows the capabilities of their device
    - Some devices only have “on” and “off”, others have intermediate states
- Drivers can control their own power independently of system power
  - Display can dim, disk spin down, etc.
  - Cannot manipulate system power state or prevent system state transition
- Applications can provide input
  - Register for power notification e.g. battery low, switched from DC to AC, etc

# Troubleshooting I/O Activity

- Filemon can be a great help to understand and troubleshooting I/O problems
- Two basic techniques:
  - Go to end of log and look backwards to where problem occurred or is evident and focused on the last things done
  - Compare a good log with a bad log
- Often comparing the I/O activity of a failing process with one that works may point to the problem
  - Have to first massage log file to remove data that differs run to run
    - Delete first 3 columns (they are always different: line #, time, process id)
      - Easy to do with Excel by deleting columns
  - Then compare with FC (built in tool) or Windiff (Resource Kit)

# Filemon

- # - operation number
- Process: image name + process id
- Request: internal I/O request code
- Result: return code from I/O operation
- Other: flags passed on I/O request



The screenshot shows the File Monitor application window with a table of file operations. The table has columns for #, Time, Process, Request, Path, Result, and Other. Several rows are highlighted in red, including operations 6153, 6154, 6155, 6161, and 6162.

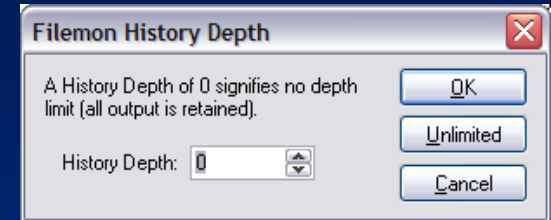
#	Time	Process	Request	Path	Result	Other
6148	12:35:49.953 PM	notepad.exe:1744	QUERY INFORMATION	C:\WINDOWS\FONTS\COUR.TTF	SUCCESS	Length: 303296
6149	12:35:49.953 PM	notepad.exe:1744	QUERY INFORMATION	C:\WINDOWS\FONTS\COUR.TTF	SUCCESS	Attributes: A
6150	12:35:49.953 PM	notepad.exe:1744	QUERY INFORMATION	C:\WINDOWS\FONTS\COUR.TTF	SUCCESS	FileFsAttributeInformation
6151	12:35:49.953 PM	notepad.exe:1744	QUERY INFORMATION	C:\WINDOWS\FONTS\COUR.TTF	SUCCESS	Length: 303296
6152	12:35:50.015 PM	notepad.exe:1744	OPEN	C:\temp\	SUCCESS	Options: Open Directory Acc...
6153	12:35:50.015 PM	notepad.exe:1744	DIRECTORY	C:\temp\	SUCCESS	FileBothDirectoryInformation: ...
6154	12:35:50.015 PM	notepad.exe:1744	CLOSE	C:\temp\	SUCCESS	
6155	12:35:50.015 PM	notepad.exe:1744	OPEN	C:\temp\test.txt	SUCCESS	Options: Open Access: All
6156	12:35:50.031 PM	explorer.exe:1520	QUERY INFORMATION	C:\WINDOWS\system32\notepad.exe	SUCCESS	Attributes: A
6157	12:35:50.031 PM	explorer.exe:1520	OPEN	C:\WINDOWS\system32\notepad.exe	SUCCESS	Options: Open Access: Exec...
6158	12:35:50.031 PM	explorer.exe:1520	QUERY INFORMATION	C:\WINDOWS\system32\notepad.exe	SUCCESS	Length: 88064
6159	12:35:50.031 PM	explorer.exe:1520	CLOSE	C:\WINDOWS\system32\notepad.exe	SUCCESS	
6160	12:35:50.078 PM	mdm.exe:1124	QUERY INFORMATION	C:\WINDOWS\system32\notepad.exe	BUFFER OV...	FileNameInformation
6161	12:35:50.078 PM	mdm.exe:1124	QUERY INFORMATION	C:\WINDOWS\system32\notepad.exe	SUCCESS	FileNameInformation
6162	12:35:50.140 PM	notepad.exe:1744	QUERY INFORMATION	C:\temp\test.txt	BUFFER OV...	FileFsAttributeInformation

# Using Filemon

- Start/stop logging (Control/E)
- Clear display (Control/X)
- Open Explorer window to folder containing file:
  - Double click on a line does this
- Find – finds text within window
- Save to log file
- Advanced mode
- Network option

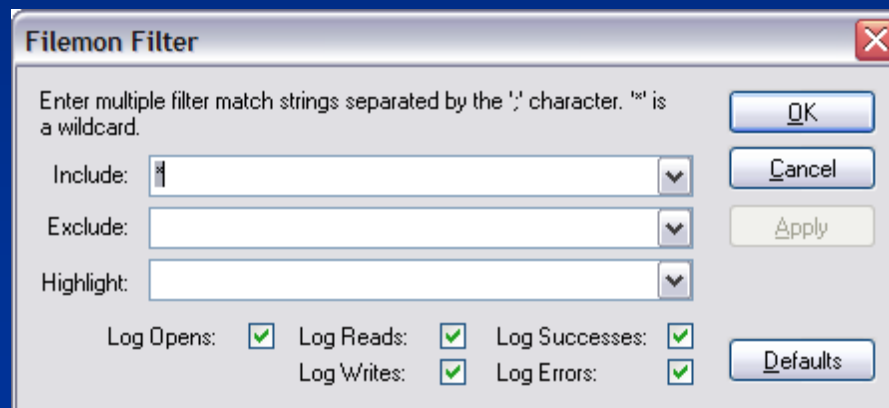
# What Filemon Monitors

- By default Filemon traces all file I/O to:
  - Local non-removable media
  - Network shares
- Stores all output in listview
  - Can exhaust virtual memory in long runs
  - You can limit captured data with history depth
- You can limit what is monitored:
  - What volumes to watch in Volumes menu
  - What paths and processes to watch in Filter dialog
  - What operations to watch in Filter dialog (reads, writes, successes and errors)



# Filemon Filtering and Highlighting

- Include and exclude filters are substring matches against the process and path columns
  - Exclude overrides include filter
- Be careful that you don't exclude potentially useful data
  - Capture everything and save the log
  - Then apply filters (you can always reload the log)
- Highlight matches all columns



# Basic vs Advanced Mode

- Basic mode messages output to be sysadmin-friendly and target common troubleshooting
- Things you don't see in Basic mode:
  - Raw I/O request names
  - Various internal file system operations
  - Activity in the System process
  - Page file I/O
  - Filemon file system activity

# Understanding Disk Activity

- Use Filemon to see why your hard disk is crunching
  - Process performance counters show I/O activity, but not to where
  - System performance counters show which disks are being hit, but not which files or which process
  - Filemon pinpoints which file(s) are being accessed, by whom, and how frequently
- You can also use Filemon on a server to determine which file(s) were being accessed most frequently
  - Import into Excel and make a pie chart by file name or operation type
  - Move heavy-access files to a different disk on a different controller



# Polling and File Change Notification

- Many applications respond to file and directory changes
  - A poorly written application will “poll” for changes
  - A well-written application will request notification by the system of changes
- Polling for changes causes performance degradation
  - Context switches including TLB flush
  - Cache invalidation
  - Physical memory usage
  - CPU usage
- Alternative: file change notification
- When you run Filemon on an idle system you should only see bursty system background activity
  - Polling is visible as periodic accesses to the same files and directories
  - File change notification is visible as directory queries that have no result

# Example: Word Crash

- While typing in the document Word XP would intermittently close without any error message
- To troubleshoot ran Filemon on user's system
  - Set the history depth to 10,000
  - Asked user to send Filemon log when Word exited

# Solution: Word Crash

- Working backwards, the first “strange” or unexplainable behavior are the constant reads past end of file to MSSP3ES.LEX

	Time	Process	Request	Path	Result	Other
0	2:31:48 PM	WINWORD.EX...	READ	C:\Program Files\Common Files\Microsoft Shared\Proof\MSSP3ES.LEX	END OF FILE	Offset: 1251810 Length: 457
1	2:31:48 PM	WINWORD.EX...	READ	C:\Program Files\Common Files\Microsoft Shared\Proof\MSSP3ES.LEX	END OF FILE	Offset: 1251810 Length: 457
2	2:31:48 PM	WINWORD.EX...	READ	C:\Program Files\Common Files\Microsoft Shared\Proof\MSSP3ES.LEX	END OF FILE	Offset: 1251810 Length: 457
3	2:31:48 PM	WINWORD.EX...	READ	C:\Program Files\Common Files\Microsoft Shared\Proof\MSSP3ES.LEX	END OF FILE	Offset: 1251810 Length: 457
4	2:31:48 PM	WINWORD.EX...	READ	C:\Program Files\Common Files\Microsoft Shared\Proof\MSSP3ES.LEX	END OF FILE	Offset: 1251810 Length: 457
5	2:31:48 PM	WINWORD.EX...	READ	C:\Program Files\Common Files\Microsoft Shared\Proof\MSSP3ES.LEX	END OF FILE	Offset: 1251810 Length: 457
6	2:31:48 PM	WINWORD.EX...	READ	C:\Program Files\Common Files\Microsoft Shared\Proof\MSSP3ES.LEX	END OF FILE	Offset: 1251810 Length: 457
7	2:31:48 PM	WINWORD.EX...	READ	C:\Program Files\Common Files\Microsoft Shared\Proof\MSSP3ES.LEX	END OF FILE	Offset: 1251810 Length: 457
8	2:31:48 PM	WINWORD.EX...	READ	C:\Program Files\Common Files\Microsoft Shared\Proof\MSSP3ES.LEX	END OF FILE	Offset: 1251810 Length: 457

- User looked up what .LEX file was
  - Related to Word proofing tools
  - Uninstalled and reinstalled proofing tools & problem went away

# Example: Useless Excel Error Message

- Excel reports an error “Unable to read file” when starting



# Solution: Useless Excel Error Message

- Filemon trace shows Excel reading file in XLStart folder
  - All Office apps autoload files in their start folders
- Should have reported:
  - Name and location of file
  - Reason why it didn't like it

Process	Request	Path	Result	Other
EXCELE.EXE:2548	DIRECTORY	C:\Documents and Settings\Administrator\Application Data\Microsoft\Excel\XLSTART\	SUCCESS	FileBothDirectoryInformation: ...
EXCELE.EXE:2548	CLOSE	C:\Documents and Settings\Administrator\Application Data\Microsoft\Excel\XLSTART\	SUCCESS	
EXCELE.EXE:2548	QUERY INFORMATION	C:\Documents and Settings\Administrator\Application Data\Microsoft\Excel\XLSTART\New Text Document.txt	SUCCESS	Attributes: A
EXCELE.EXE:2548	OPEN	C:\Documents and Settings\Administrator\Application Data\Microsoft\Excel\XLSTART\New Text Document.txt	SUCCESS	Options: Open Access: All
EXCELE.EXE:2548	CLOSE	C:\Documents and Settings\Administrator\Application Data\Microsoft\Excel\XLSTART\New Text Document.txt	SUCCESS	
EXCELE.EXE:2548	OPEN	C:\Documents and Settings\Administrator\Application Data\Microsoft\Excel\XLSTART\New Text Document.txt	SUCCESS	Options: Open Access: All
EXCELE.EXE:2548	QUERY INFORMATION	C:\Documents and Settings\Administrator\Application Data\Microsoft\Excel\XLSTART\New Text Document.txt	SUCCESS	Length: 1
EXCELE.EXE:2548	QUERY INFORMATION	C:\Documents and Settings\Administrator\Application Data\Microsoft\Excel\XLSTART\New Text Document.txt	SUCCESS	Length: 1
EXCELE.EXE:2548	QUERY INFORMATION	C:\Documents and Settings\Administrator\Application Data\Microsoft\Excel\XLSTART\New Text Document.txt	SUCCESS	Length: 1
EXCELE.EXE:2548	CLOSE	C:\Documents and Settings\Administrator\Application Data\Microsoft\Excel\XLSTART\New Text Document.txt	SUCCESS	
EXCELE.EXE:2548	QUERY INFORMATION	C:\Documents and Settings\Administrator\Application Data\Microsoft\Excel\XLSTART\New Text Document.txt	SUCCESS	Attributes: A
EXCELE.EXE:2548	OPEN	C:\Documents and Settings\Administrator\Application Data\Microsoft\Excel\XLSTART\New Text Document.txt	SUCCESS	Options: Open Access: All
EXCELE.EXE:2548	QUERY INFORMATION	C:\Documents and Settings\Administrator\Application Data\Microsoft\Excel\XLSTART\New Text Document.txt	SUCCESS	Length: 1
EXCELE.EXE:2548	LOCK	C:\Documents and Settings\Administrator\Application Data\Microsoft\Excel\XLSTART\New Text Document.txt	SUCCESS	Excl: Yes Offset: 0 Length: -1
EXCELE.EXE:2548	READ	C:\Documents and Settings\Administrator\Application Data\Microsoft\Excel\XLSTART\New Text Document.txt	SUCCESS	Offset: 0 Length: 16384
EXCELE.EXE:2548	READ	C:\Documents and Settings\Administrator\Application Data\Microsoft\Excel\XLSTART\New Text Document.txt	END OF FILE	Offset: 16384 Length: 16384

# Labs

- View installed Drivers
  - Msinfo32.exe
  - Process explorer
  - livekd
- View Device name mapping (winobj)
- View driver dispatch routines (!drvobj kbdclass 7)
- Find an IRP (!irpfind, !irp)
- Viewing device tree (control panel, !devnode 0 1)
- View device power mapping (control panel)
- View power policy and capabilities (!pocaps, !popolicy)
- Driver Verifier
- FileMon

# Further Reading

- Penny Orwick and Guy Smith. *Developing Drivers with Windows Driver Foundation*.
- Mark E. Russinovich *et al.* *Windows Internals*. 5th Edition, Microsoft Press, 2009.