Windows Driver Framework

Windows Driver Modes (WDM) is the core driver model for Windows
KMDF (Kernel Mode Driver Framework) does not replace WDM

WDF consists of:
- DriverEntry routine (WDM)
- A collection of WDF objects
- A set of callback functions (as defined by the WDF)
  - It appears that driver writers do not subclass WDF objects
  - It appears that driver writers do not have direct access to WDF objects

- WDF driver object
  - Created in DriverEntry with WdfDriverCreate
  - Requires "attributes" and "config" passed in
  - "config" includes a pointer to an DeviceAdd callback
    - Naming convention: <drivername>DrvEvt<eventname>
    - eg: PciDrvEvtDeviceAdd
- WDF queue and request objects
  - Drivers using read, write of device I/O control must use a WDF queue
  - WdfIoQueueCreate -- creates a queue
  - An IRP produces a request object that is queued
  - A request callback is called with the request object

- WDF object properties
  - Wdf<object>Get<value>, Wdf<object>Set<value>
  - WDFDRIVER WdfDeviceGetDriver (_In_ WDFDEVICE Device)
  - ULONG_PTR WdfRequestGetInformation (_In_ WDFREQUEST Request)
  - Reference counts ... if needed
    - WdfObjectReference(), WdfObjectDereference()

- WdfObjectDelete -- for objects that drivers may delete
  - Drivers can’t delete a number of Wdf objects
Simple DriverEntry routine (general/echo ~phil/public/cs513/Widows-driver-samples-master.zip)

NTSTATUS DriverEntry( IN PDRIVER_OBJECT DriverObject, IN PUNICODE_STRING RegistryPath )
{
    WDF_DRIVER_CONFIG config;
    NTSTATUS status;

    WDF_DRIVER_CONFIG_INIT(&config, EchoEvtDeviceAdd);

    status = WdfDriverCreate(DriverObject, RegistryPath, WDF_NO_OBJECT_ATTRIBUTES,
                              &config, WDF_NO_HANDLE);
    if (!NT_SUCCESS(status)) {
        KdPrint("Error: WdfDriverCreate failed 0x%x\n", status);
        return status;
    }

    EchoPrintDriverVersion();

    return status;
}
WDF (page 4)

WDF_DRIVER_CONFIG -- structure
- EvtDriverDeviceAdd
- EvtDriverUnload
- DriverInitFlags
- DriverPoolTag

All "create" methods have a optional WDF_OBJECT_ATTRIBUTES structure as input
- EvtCleanupCallback / EvtDestroyCallback (if needed)
- ExecutionLevel -- IRQL
- SynchronizationScope (InheretFromParent, Device, Queue, None)
- ParentObject
- ContextSizeOverride -- Driver specific storage for each object!
- ContextTypeInfo -- (struct describing the type)
  - In an H file: struct _FDO_DATA {.....} FDO_DATA, *PFDO_DATA;
  - WDF_DECLARE_CONTEXT_TYPE_WITH_NAME(FDO_DATA, FdoGetData)
  - Creates the accessor method for getting the object context
EvtDriverDeviceAdd callback
- Create a framework device object (WdfDeviceCreate)
  - Initialize many callbacks and "data" (FdoData)
- Create I/O Queue
- Create a device interface or symbolic link (WdfDeviceCreateSymbolicLink)
- Create driver-defined interfaces
  - API to talk to driver without Device I/O Control
- Initialize Windows Management Instrumentation (WMI) if wanted
- Create interrupt objects
- Enable direct memory access (if needed)
- Set IO Type (WdfDeviceInitSetIoType(DeviceInit, WdfDeviceIoBuffered)
- Set up the PNP system and call backs.

Look at several versions in code:
- Echo
- PciDrv
- toaster -- featured
Configuring the device in DeviceAdd

- WDF_FILEOBJECT_CONFIG structure
- WDF_FILEOBJECT_CONFIG_INIT macro ..

```
WDF_FILEOBJECT_CONFIG_INIT( &fileConfig, ToasterEvtDeviceFileCreate,
                            ToasterEvtFileClose, WDF_NO_EVENT_CALLBACK /*Cleanup*/ );
```

- By default, WDF takes care of Create/Close/Cleanup

Registering for IRP events

- Part of the WDF Queue configuration before queue creation
- see toaster/toastDrv/kmdf/func/featured
  - queueConfig.EvtIoRead = ToasterEvtIoRead;
  - queueConfig.EvtIoWrite = ToasterEvtIoWrite;
  - queueConfig.EvtIoDeviceControl = ToasterEvtIoDeviceControl;
PNP callbacks and issues

- WDF_PNPPOWER_EVENT_CALLBACKS_INIT(&pnpPowerCallbacks);
  - EvtDevicePrepareHardware (START_DEVICE)
  - EvtDeviceReleaseHardware (STOP_DEVICE)
  - EvtDeviceSelfManagedIoInit (??)
- Variety of others / power events
  - awake from S0 .....  
- Other possible set up before creating device
  - xxxEvtDeviceContextCleanup()
EvtDeviceFileCreate, EvtDeviceFileClose
- Default -- done by WDF
- Can get a call -- e.g. to return BUSY ...

VOID NameEvtDeviceFileCreate ( IN WDFDEVICE Device,
    IN WDFREQUEST Request, IN WDFFILEOBJECT FileObject )
  - FileObject created for the Create/Close session

VOID WdfRequestComplete(Request, STATUS_SUCCESS) -- call to complete the IRP

EvtIoRead:
- VOID NameEvtIoRead (WDFQUEUE Queue, WDFREQUEST Request, size_t Length)
- Length -- length of data buffer with request (no 0 length requests)

Core code:
```c
    status = WdfRequestRetrieveOutputMemory(Request, &memory);
    if(NT_SUCCESS(status) ) {
        // Copy data into the memory buffer using WdfMemoryCopyFromBuffer
    }
    WdfRequestCompleteWithInformation(Request, status, bytesCopied);
```
EvtIoWrite --

VOID NameEvtIoWrite (WDFQUEUE Queue, WDFREQUEST Request, size_t Length)

Core code:

status = WdfRequestRetrieveInputMemory(Request, &memory);
if(NT_SUCCESS(status) ) {
    //
    // 1) Use WdfMemoryCopyToBuffer to copy data from the request
    // to driver buffer.
    // 2) Or get the buffer pointer from the request by calling
    // WdfRequestRetrieveInputBuffer to transfer data to the hw
    // 3) Or you can get the buffer pointer from the memory handle
    // by calling WdfMemoryGetBuffer to transfer data to the hw.
    //
}
WdfRequestCompleteWithInformation(Request, status, Length);
Set of calls -- I/O Buffers from Request packets

- \texttt{WdfRequestRetrieveInputBuffer} -- address of "input" buffer in request
- \texttt{WdfRequestRetrieveInputMemory} -- \texttt{WDFMEMORY} object for input buffer
- \texttt{WdfRequestRetrieveInputWdmMdl} -- Gets the MDL from the IRP
- Similar set for Output -- which means "output from driver"
- \texttt{WdfRequestRetrieveUnsafeUserInputBuffer} -- for NEITHER I/O

\texttt{WDFMEMORY} object allows use of \texttt{WdfMemoryXxx} routines.

- \texttt{WdfMemoryGetBuffer} -- two step for above
- \texttt{WdfMemoryCopyTo/FromBuffer} --

For Buffered I/O input and output buffers are the same ...

- \texttt{WdfRequestProbeAndLockUserBufferXxx}

Device Control

- \texttt{VOID NameEvtIoDeviceControl(IN WDFQUEUE Queue, IN WDFREQUEST Request,}
  \hspace{1em} IN size_t OutputBufferLength, IN size_t InputBufferLength,  
  \hspace{1em} IN ULONG IoControlCode )
- \texttt{WdfRequestRetrieveInput/OutputBuffer} -- gets the buffer pointer
Interrupts

- Connected during DeviceAdd
- WDF_INTERRUPT_CONFIG
- WDF_INTERRUPT_CONFIG_INIT (&obj, InterruptIsr, InterruptDpc)
- EvtInterruptEnable, EvtInterruptDisable if needed
  - May not need these in your driver
- WdfInterruptCreate(device, &intConfig, &ObjAttributes, &retObjectPointer)
- BOOLEAN NameEvtInterruptIsr(IN WDFINTERRUPT Interrupt, IN ULONG MessageID)
  - WdfInterruptGetDevice(Interrupt)
  - Get the context from the Device Object
  - WdfInterruptQueueDpcForIsr( Interrupt ) -- queues the Dpc
- VOID NameEvtInterruptDpc(IN WDFINTERRUPT WdfInterrupt, IN WDFOBJECT WdfDevice)
  - WdfIoQueueRetrieveNextRequest()
  - WdfRequestCompleteWithInformation() -- to finish a request
Various topics

Security of device objects

- Part of the "Device Init" information passed to WdfDeviceCreate
- SDDL_DEVOBJ_SYS_ALL_ADM_RWX_WORLD_R_RES_R -- default for device object
- WdfDeviceInitAssignName() uses SDDL_DEVOBJ_SYS_ALL_ADM_ALL
- WdfDeviceInitAssignSDDLString()
- An INF file can override the default

Self-Managed I/O

- plug-and-play and power state transitions
  - not "normal" activity
  - sounds very specialized

I/O Targets

- One driver calling another driver for service..
- receiving driver is the target
- methods for creating I/O requests to I/O targets
Driver Tracing

Event Tracing for Windows (ETW) -- general event tracing system

- software tracing
- real time or logs
- Drivers get a simpler interface ... WPP
- Most of the driver examples have WPP tracing code

"Advantages"

- Dynamic and Flexible control -- no reboot required
- real time or log file
- "rich information" auto file name and line numbers ...
- WPP messages are binary format .. may help protect intellectual property
- "easy migration from debug print statements"
- WPP can be used "in the field" both on free and checked
- Minimal performance impact
Review of routines:

In general, you XXX_OBJECT_ATTRIBUTES_INIT before the Create call
- May be different versions for init variations

DriverEntry
- InitWPP tracing if wanted
- create Wdf Driver object

DeviceAdd (M$ code calls "CreateDevice" .... same difference/same IRP call)
- Init device attributes and fileobject config
- set up PNP/Power callbacks
  - PrepareHardware callback needed
- create the device using Attributes set by WdfDeviceInitSetFileObjectConfig()
- get and initialize the device context
- create device interface and symbolic link
- create a queue for I/O requests, initialize for device I/O control, read
- config_init and create an interrupt object, pointers to isr and dpc
Review of routines (page 2)

**PrepareHardware**
- Get the actual addresses of BAD1 and BADR2 (save in device context)
- Initialize the hardware on BADR1

**CreateFile**
- Check if open, complete request as BUSY if it is open
- Reset defaults, re-initialize the device context to known values
- Tell hardware default values

**CloseFile**
- If you have a device context, stop sampling, stop interrupts ...

**IoRead**
- Transfer if you have data (helper function to do it is good)
- EOF if no data and not sampling
Review of routines (page 3)

**Isr**
- Same things as for NetBSD
- wakeup/condition var replaced by start a DPC.

**Dpc**
- Transfer data (helper function same as for IoRead)

**DeviceIoControl**
- Very similar to NetBSD
- get/set channel -> return or store values in device context
- get/set rate -> return or store values in device context
- Start sampling -> set up clock, start sample, latch clock, ...
- Stop sampling -> stop interrupts

Decorate with KdPrint()s or TraceEvents()

**WdfRequestRetrieveOutputBuffer vs WdfRequestRetrieveOutputMemory**
- pointer to memory vs object
- RtlCopyMemory() should work with pointer, object needs WdfMemoryCopyFromBuffer()