Writing WDF Drivers I: Core Concepts

The Windows Driver Foundation (WDF) is the modern standard for developing Windows drivers, and is the preferred way to implement most new drivers for Windows. WDF enables developers to write drivers that execute in either kernel-mode using the Kernel Mode Driver Framework (KMDF) or user-mode using the User Mode Driver Framework V2 (UMDF V2).

The goal of this seminar is to help engineers learn both the fundamental and practical details of KMDF and UMDF V2. This includes both basic architectural information about Windows and the Windows I/O Subsystem as it relates to WDF, plus the specific details of how to write, test, diagnose, and support drivers using KMDF and UMDF V2.

The seminar specifically focuses specifically on KMDF and UMDF V2 software drivers, filter drivers, and drivers for USB devices, as well as drivers for programmed I/O type devices (interfaced via either PCI-family or motherboard-based buses). The core concepts needed for developing drivers for these devices on SoC systems are also discussed in the seminar.

Hardware Included: A valuable part of your learning experience in this course will be the lab exercises targeted for use with the OSR USB FX2 Learning Kit (check out the OSR Online Store for more information. Of course, we want your learning experience to extend beyond the classroom, so each student will walk away with their very own OSR USB FX2 Learning Kit.

This is the most recent version of the seminar that we've been teaching on-campus to Microsoft's own internal software development and test engineers for the last ten years.

Target Audience

This seminar is designed for engineers who need to understand how to design, develop, and test Windows drivers using the WDF Kernel Mode Driver Framework (KMDF) or the WDF User Mode Driver Framework V2 (UMDF V2).

Prerequisites

Students attending this seminar will be assumed to have a good working knowledge of general O/S concepts (user mode versus kernel mode, virtual memory concepts, and concurrency issues) and device concepts (such as buses, interrupts, and registers). Previous experience developing device drivers (on any operating system) will definitely be an advantage, but is not required.

Due to the hands-on orientation of this seminar, attendees will be assumed to be able to use Windows at a user level, including how to use Microsoft's Visual Studio. Working knowledge of the C programming language, and how to read and write to a file using Win32 APIs (CreateFile, ReadFile, WriteFile) are also assumed.

Hardware Requirement: Each student must bring a laptop with at least one available USB 2.0 port, running Windows 7 or later. We recommend a system with at least 4GB of memory and 60GB of free disk space. A complete laptop setup requirements may be found at http://www.osr.com/seminars/osr-driver-development-seminar-laptop-requirements/

After This Seminar

This seminar provides you with the necessary information to attend OSR's Kernel Debugging and Crash Analysis seminar, as well as OSR's Writing WDF Drivers II: Advanced Implementation Techniques seminar.

Seminar Outline

- **Introduction**
  Welcome remarks, seminar goals and objectives. Introduction to WDF, as well as KMDF and UMDF V1 and UMDF V2. When each most is most applicable.

- **Windows OS Architecture for Driver Writers**
  A review of core Windows architecture concepts of specific importance to driver developers. Topics include:
  - General organization of NTOS: Executive, Kernel, HAL
  - Processes and threads
  - Memory management
  - Introduction to key I/O Manager objects (File Object, Device Object, Driver Object)
  - Overview of the PnP process
  - Power Management
  - Executive Services
  - Object Manager – How an object name (such as a file specification) is processed
  - How File Objects are created and the process-specific handle table
  - Windows Kernel introduction
  - Windows HAL introduction

- **The Windows Device Tree**
  This module comprises a detailed view of how the Windows PnP subsystem discovers and enumerates devices and loads their associated drivers. Details for devices on dynamically enumerable buses (such as PCIe, USB, or Bluetooth) and non-dynamically enumerable buses (such as SPI, I2C, or GPIO) are described. All about Physical Device Objects (PDOs), Function Device Objects (FDOs), and filter Device Objects. How filter drivers work their magic. How requests are processed, and passed from driver to driver within the Windows I/O Subsystem. I/O request completion handling. Further discussion and description (expanding on information

from the previous module) of how system requests in general and I/O request in particular are passed from user-mode to kernel-mode, initially in the context of the requesting thread.

- **Driver Installation**
  How to create installation control files for KMDF drivers. The Ten Most Frequently Used INF File Sections are discussed. The KMDF Co-Installer, and how to specify it in an INF file, is described.

- **The WDF Object Model**
  WDF object characteristics and taxonomy. How objects are instantiated and used in KMDF and UMDF V2. An overview of the most common WDF objects.

- **Driver Initialization**
  How a WDF driver and its associated device are initialized. Also, handling typical PnP and power management events such as device discovery (EvtDriverDeviceAdd), power-up (EvtDeviceD0Entry), and power-down (EvtDeviceD0Exit). How WDF drivers are notified of and claim their hardware resources, including registers, ports, connections, and interrupts.

- **Building and Debugging**
  Students are assumed to already be familiar with the basics of how to use Visual Studio. We'll review the details, as well as discuss how to setup and use WinDbg, the Windows kernel debugger. We'll also discuss specific issues about building and debugging drivers for Windows using the WDK and Visual Studio, as well as tools such as Code Analysis and Static Driver Verifier. We also discuss the WDF Kernel Debugger Extensions (WDFKD), including retrieving the WDF Log from the "in flight recorder."

  Lab: Building and Debugging, Driver Initialization (DriverEntry, EvtDeviceD0Entry, etc.).

- **Interrupt Request Levels & Deferred Procedure Calls**
  In this module, we discuss the all-important concept of Interrupt Request Levels (IRQLs), and the specific uses that Windows makes of various IRQLs. We also discuss Deferred Procedure Calls (DPCs) and how they're used in Windows for Interrupt Service Routine completion (DPCforISR). We also discuss passive-level interrupts and its associated Work Item for ISR.

- **Queues and Requests**
  How a WDF driver gets I/O requests, and how those requests are processed. Topics include: how WDF Queues are instantiated, Queue dispatch types, and how Queues can be used to sort Requests; Framework Requests and how Requests are processed and completed; how the user data buffer associated with a Request is accessed and in what context this access is allowed; how Requests are completed with data, status, and information returned.

- **Buffer Methods and Device Controls**
  The different ways that requestor data buffers can be described are discussed. Direct I/O, Buffered I/O, and Neither I/O are described, compared, and contrasted. Also discussed is how to define custom Device IO Control Codes (IOCTLs), and how the previously described buffering methods apply to IOCTLs.

  Lab: Request Processing and Completion, Filtering

- **Case Study 1: Programmed I/O Device**
  This case study reviews and demonstrates all the topics discussed in the seminar to this point. To do this, the instructor takes students on a guided walk-through of the code for a driver for "the world's simplest device", reviewing and reinforcing concepts along the way. All phases of driver operation are reviewed from device discovery, to claiming hardware resources, processing power state changes, and processing and completing requests both synchronously and asynchronously.

- **I/O Targets**
  How a driver sends Requests to other drivers in the system, and optionally receives the results. Local, Remote, and Special I/O Targets. Both synchronously and asynchronously send operations are discussed. Completion routines are also covered.

- **Case Study 2: USB**
  - **Part 1: USB Concepts**
    The basics of USB are discussed including device, configuration, and interface descriptors. Endpoints and pipes are described.
  - **Part 2: Implementing WDF USB Drivers**
    In this section, we describe how USB drivers are implemented in KMDF. This includes how a configuration and interface is chosen, and how endpoints are retrieved. How to send vendor commands to a device via Endpoint 0. Using Bulk and Interrupt endpoints. The WDF Continuous Reader is briefly discussed, as is supporting Selective Suspend (USB device power management).

  Lab: USB, Using the OSR USB-FX2 device

- **Case Study 3: SPB/GPIO Device**
  How Simple Peripheral Bus clients and GPIO-connected devices (such as senors on SPI, I2C, and GPIO buses) are supported in Windows. SPBs and the RESOURCE_HUB. Controller drivers and Client drives. Walk-through of the initialization and I/O processing a Client driver that receives interrupts via a GPIO line and collects data from an I2C or SPI bus. And introduction to the Power Management Framework (PoFx), including review of example configurations and callbacks for single-component devices.

- **Serialization**
  In this module, we discuss issues relating to synchronizing access to shared data within a driver. The much misunderstood topic of WDF Synchronization Scope is fully described, as is extending sync scope to other callback routines via the Automatic Serialization parameter. WDFSPINLOCKs and WDFWAITLOCKs are discussed, along with the underlying implementations of each and how they're used and the implications of using each of these serialization mechanisms from user-mode drivers.

- **Cleanup, Close and Cancel**
  Strategies for handling queued and in-progress requests are discussed, as what processing typically takes place as part of cleanup and close processing.

- **Helpful Classes**
  A brief description of a few Framework classes such as WDFCOLLECTION, WDFWORKITEM, and WDFTIMER that might be useful for driver developers to know.

  Lab: USB Continued, Open/Close Processing