Writing WDF Drivers II: Advanced Implementation Techniques

This fast-paced 4-day lab seminar is designed for students with experience writing WDF drivers or who have previously taken OSR's *Writing WDF Drivers I: Core Concepts* seminar. Through a mixture of lectures and lab assignments, students expand their knowledge into new areas of WDF and learn practical methods for dealing with design challenges that are common in more complex WDF drivers.

Picking up from where our **Writing WDF Drivers I: Core Concepts** seminar leaves off, this seminar takes you beyond the simple use of Queues, Requests, I/O Event Processing Callbacks, and I/O Targets. The class does not limit itself to WDF-based solutions, but also demonstrates how specific WDM concepts can be used within WDF drivers to build optimal solutions. The class describes architectural concepts while stressing practical techniques and solutions applicable to WDF drivers. For example, different methods of driver to driver communication are described, compared, the advantages and disadvantages discussed, and suggestion for when using each might be most appropriate are presented. The class follows this same approach for user-mode to kernel-mode communications: alternative methods are described, the strengths and challenges inherent in each are reviewed, and recommendations on when each might be the best choice are discussed.

The class also takes the time to delve into more complex areas that are not covered our **Writing WDF Drivers I: Core Concepts** seminar. These includes Bus Drivers, DMA (including both the Windows DMA abstraction), and the latest in terms of best practices for driver testing and validation.

Details

Length: 4 days

Format: Lecture and Lab

Target Audience

This seminar is designed for engineers who design, develop or test WDF drivers and already know basic WDF concepts. Engineers who want to understand multiple techniques (and current best practices) for solving a variety of common driver problems using WDF will maximally benefit from this seminar. Please carefully review the required prerequisites, below, as this seminar is not suitable for beginners.

Prerequisites

Previous attendance at OSR's **Writing WDF Drivers I: Core Concepts** seminar is explicitly assumed. Because this is a hands-on class, a working knowledge of C/C++, Visual Studio, and Windows are required.

Please note, that due to the intensive nature of this seminar, the instructor will not be able to slow-down the pace of this seminar to accommodate students who have not taken the prerequisite OSR seminar.

As an alternative to having taken OSR's **Writing WDF Drivers I: Core Concepts**, attendees may substitute actual hands-on experience implementing WDF drivers with the approval of the OSR seminar staff. This experience includes a solid understanding of Windows OS and I/O subsystem architecture, as well as working knowledge of basic WDF concepts (the object model, the use of Queues, Requests, and I/O Event Processing Callbacks). Also assumed is a solid understanding of IRQLs, DPCs, multi-threaded operation, and locking in WDF drivers. A basic understanding of driver installation procedures and INF files are also assumed.

If you have not taken OSR's **Writing WDF Drivers I: Core Concepts**, please talk with or email the OSR seminar staff before registering to ensure you'll be comfortable with the starting level of this seminar.

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/

Seminar Outline

- **Introduction**
  Welcome remarks, seminar goals and objectives.

- **Review of WDF and I/O Subsystem Basics**
  All the basics of WDF and the Windows I/O Subsystem in about two hours, with some explicit descriptions of how WDF concepts relate to WDM added to keep things interesting. This fast-paced module reviews how device stacks are built, how Requests are passed from driver to driver, the WDF object model, common Event Processing Callbacks, I/O Targets and Completion Callbacks are all discussed. In addition, some details about the WDM PnP and Power Management events that stimulate specific WDF events are presented.

- **Driver to Driver Communication – Going Beyond I/O Targets**
  A discussion of different methods driver can use to communicate, the advantages and disadvantages of each, and when each might be most appropriate. PnP notification, bus interfaces, and driver-defined through-call structures are all discussed. This module ends with a code walk-through of a chosen solution.

- **WDF Bus Drivers**
  In this module, we discuss the WDF bus driver model including static and dynamic enumeration of child devices. Providing resources to child devices, and raw PDO usage are discussed. When it makes sense to implement a software-only bus driver. The module ends with a code walk-though of a KMDF bus driver that does static child enumeration.

- **Best Practices for WDF Driver Quality, Debugging and Serviceability**
  A brief discussion of how best to use the myriad tools available to WDF driver developers. Practical advice on and current best practices for the use of KMDF Verifier, Windows Driver Verifier, Static Driver Verifier, Code Analysis, and even the checked build of Windows are provided.

  Lab: Driver to Driver Communications, Bus Drivers

- **Kernel-Mode to User-Mode Communication – Going Beyond Simple Requests**
  A discussion of how to safely use Neither I/O (validating user buffers and the use of structured exception handling), Fast I/O for Device Control, and optimizations in the inverted call model. How to safely share memory and certain specific objects between user-mode and kernel-mode is also discussed. The module ends with the review of a case study of a low-latency process control device and its WDF driver.

- **Debugging Invalid Memory Access Problems**
  No crash is more common in a Windows driver than one due to dereferencing a bad pointer, or referencing a good pointer at the wrong IRQL. In this module, we'll discuss how Windows traps invalid memory references, and the specific differences between the bug check codes IRQL_NOT_LESS_OR_EQUAL, PAGE_FAULT_IN_NONPAGED_AREA, and KERNEL_DATA_INPAGE_ERROR, with the goal of making it easier and faster to debug commonly encountered driver problems.

  Lab: User to Kernel Communications

- **Busmaster DMA**
  This module starts with a discussion of the Windows DMA abstraction including map registers, system scatter/gather support, and more. We then review the WDF DMA model (Enabler and Transaction Objects). A few brief words on System DMA (yes, it still exists!) are also presented. The module concludes with a code walk-through demonstrating typical KMDF DMA data handling.

- **Practical Problems, Practical Solutions**
  This module comprises a discussion of some common problems and good solutions to these problems. Included among the topics discussed: Why relying on WDF object parenting for child object destruction isn't always as simple as it seems. The cause of bugcheck code 0x9F and the when you need to implement EvtIoStop. Work Queues and the High Priority Worker Effect. Using reader/writer spinlocks to efficiently read and update shared data. How to protect state teardown using Rundown references.

- **Introducing: UMDF V2.0**
  UMDF is new with Windows 8.1, and most engineers haven't yet had a chance to experiment with it. In this module, we briefly discuss this new user-mode driver model. Specifically, we discuss what's supported and when UMDF 2 is most applicable. We also provide hints for moving from KMDF to UMDF 2.

  Lab: UMDF V2.0