



WINDOWS 7 DEVICE DRIVER

RONALD D. REEVES



WINDOWS 7 DEVICE DRIVER



WINDOWS 7 DEVICE DRIVER

Ronald D. Reeves, Ph.D.

◆◆Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside the United States, please contact:

International Sales
international@pearsoned.com

Visit us on the Web: informit.com/aw

Library of Congress Cataloging-in-Publication Data

Reeves, Ron.

Windows 7 device driver / Ronald D. Reeves.

p. cm.

Includes bibliographical references and index.

ISBN-13: 978-0-321-67021-2 (pbk. : alk. paper)

ISBN-10: 0-321-67021-3 (pbk. : alk. paper)

1. Microsoft Windows device drivers (Computer programs)

I. Title.

QA76.76.D49R44 2011

005.7'1—dc22

2010039109

Copyright © 2011 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, write to:

Pearson Education, Inc.
Rights and Contracts Department
501 Boylston Street, Suite 900
Boston, MA 02116
Fax: (617) 671-3447

ISBN-13: 978-0-321-67021-2

ISBN-10: 0-321-67021-3

Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana.

First printing, November 2010

*I would like to dedicate this book to my best friend, and partner in life,
my wife, Paulette. Her untiring support and love over the years have been
a great source of inspiration.*

This page intentionally left blank

CONTENTS

| | |
|---|-----------|
| Preface | xv |
| About the Author | xix |
| Introduction | 1 |
| PART I DEVICE DRIVER ARCHITECTURE OVERVIEW | 5 |
| Chapter 1 Objects | 7 |
| 1.1 Nature of an Object | 7 |
| 1.2 What Is a Software Object? | 8 |
| 1.3 Gaining an Understanding | 10 |
| 1.4 Software Components | 11 |
| Chapter 2 Windows Driver Foundation (WDF) Architecture | 13 |
| 2.1 WDF Component Functions | 13 |
| 2.2 Design Goals for WDF | 14 |
| 2.3 Device and Driver Support in WDF | 15 |
| 2.4 WDF Driver Model | 16 |
| 2.5 WDF Object Model | 17 |
| 2.5.1 Kernel Mode Objects | 19 |
| 2.5.2 User Mode Objects | 19 |
| 2.6 Plug and Play and Power Management Support | 20 |
| 2.6.1 Plug and Play/Power Management State Machine | 21 |
| 2.7 Integrated I/O Queuing and Cancellation | 22 |
| 2.7.1 Concurrency | 22 |
| 2.7.2 I/O Model | 23 |
| 2.7.3 I/O Request Flow | 24 |
| 2.7.4 Device I/O Requests | 25 |
| 2.7.5 Plug and Play and Power Management Requests | 26 |

| | | |
|--------|---|----|
| 2.8 | WMI Requests (Kernel Mode Drivers Only) | 27 |
| 2.9 | Driver Frameworks | 28 |
| 2.9.1 | Kernel Mode Framework | 29 |
| 2.9.2 | User Mode Framework | 31 |
| 2.10 | Windows Kernel | 32 |
| 2.10.1 | Reflector | 32 |
| 2.10.2 | Driver Host Process | 32 |
| 2.10.3 | Driver Manager | 33 |
| 2.11 | Tools for Development and Testing | 33 |
| 2.11.1 | PREfast for Drivers | 34 |
| 2.11.2 | Static Driver Verification (SDV) | 35 |
| 2.11.3 | Frameworks Verifier | 36 |
| 2.11.4 | Trace Logging | 36 |
| 2.11.5 | Debugger Extensions | 37 |
| 2.11.6 | Serviceability and Versioning | 37 |

PART II USER MODE DRIVERS 39

Chapter 3 Windows 7 User Mode Drivers Overview and Operation 41

| | | |
|-------|--|----|
| 3.1 | Devices Supported in User Mode | 42 |
| 3.2 | UMDF Model Overview | 43 |
| 3.2.1 | UMDF Object Model | 45 |
| 3.2.2 | UMDF Objects | 45 |
| 3.3 | Driver Callback Interfaces | 47 |
| 3.4 | UMDF Driver Features | 49 |
| 3.4.1 | Impersonation | 50 |
| 3.4.2 | Device Property Store | 50 |
| 3.5 | I/O Request Flow | 51 |
| 3.5.1 | I/O Request Dispatching | 53 |
| 3.5.2 | Create, Cleanup, and Close Requests | 53 |
| 3.5.3 | Create, Read, Write, and Device I/O Control Requests | 56 |
| 3.6 | I/O Queues | 56 |
| 3.6.1 | Dispatch Type | 58 |
| 3.6.2 | Queues and Power Management | 59 |
| 3.7 | I/O Request Objects | 60 |
| 3.7.1 | Retrieving Buffers from I/O Requests | 61 |
| 3.7.2 | Sending I/O Requests to an I/O Target | 61 |
| 3.7.3 | Creating Buffers for I/O Requests | 63 |

| | | |
|--------|---|----|
| 3.7.4 | Canceled and Suspended Requests | 64 |
| 3.7.5 | Completing I/O Requests | 66 |
| 3.7.6 | Adaptive Time-Outs | 66 |
| 3.8 | Self-Managed I/O | 67 |
| 3.9 | Synchronization Issues | 68 |
| 3.10 | Locks | 70 |
| 3.11 | Plug and Play and Power Management Notification | 70 |
| 3.12 | Device Enumeration and Startup | 71 |
| 3.13 | Device Power-Down and Removal | 72 |
| 3.13.1 | Surprise-Removal Sequence | 74 |
| 3.14 | Build, Test, and Debug | 75 |
| 3.14.1 | Installation and Configuration | 76 |
| 3.14.2 | Versioning and Updates | 77 |

Chapter 4 Programming Drivers for the User Mode Driver Framework 79

| | | |
|-------|--|-----|
| 4.1 | Windows I/O Overview | 79 |
| 4.2 | Brief COM Information | 81 |
| 4.3 | UMDF Architecture | 82 |
| 4.4 | Required Driver Functionality | 84 |
| 4.5 | UMDF Sample Drivers | 87 |
| 4.5.1 | Minimal UMDF Driver: The Skeleton Driver | 88 |
| 4.5.2 | Skeleton Driver Classes, Objects, and Interfaces | 89 |
| 4.6 | Driver Dynamic-Link Library and Exports | 91 |
| 4.6.1 | Driver Entry Point: DllMain | 91 |
| 4.6.2 | Get Class Object: DllGetClassObject | 93 |
| 4.7 | Functions for COM Support | 95 |
| 4.7.1 | IUnknown Methods | 95 |
| 4.7.2 | IClassFactory Interface | 96 |
| 4.7.3 | Driver Callback Object | 96 |
| 4.7.4 | Device Callback Object | 100 |
| 4.8 | Using the Skeleton Driver as a Basis for Development | 106 |
| 4.8.1 | Customize the Exports File | 107 |
| 4.8.2 | Customize the Sources File | 107 |
| 4.8.3 | Customize the INX File | 108 |
| 4.8.4 | Customize the Comsup.cpp File | 108 |
| 4.8.5 | Add Device-Specific Code to Driver.cpp | 109 |
| 4.8.6 | Add Device-Specific Code to Device.cpp | 109 |

| | | |
|------------------|--|------------|
| Chapter 5 | Using COM to Develop UMDF Drivers | 111 |
| 5.1 | Getting Started | 111 |
| 5.1.1 | COM Fundamentals | 112 |
| 5.1.2 | HRESULT | 114 |
| 5.2 | Using UMDF COM Objects | 116 |
| 5.2.1 | Obtaining an Interface on a UMDF Object | 117 |
| 5.2.2 | Reference Counting | 119 |
| 5.3 | Basic Infrastructure Implementation | 120 |
| 5.3.1 | DllMain | 121 |
| 5.3.2 | DllGetClassObject | 121 |
| 5.3.3 | Driver Object's Class Factory | 122 |
| 5.3.4 | Implementing a UMDF Callback Object | 122 |
| 5.3.5 | Implementing QueryInterface | 125 |
| | | |
| PART III | KERNEL MODE DRIVERS | 127 |
| Chapter 6 | Windows 7 Kernel Mode Drivers | |
| | Overview and Operations | 129 |
| 6.1 | KMDF Supported Devices | 129 |
| 6.2 | KMDF Components | 131 |
| 6.3 | KMDF Driver Structure | 132 |
| 6.4 | Comparing KMDF and WDM Drivers | 132 |
| 6.5 | Device Objects and Driver Roles | 135 |
| 6.5.1 | Filter Drivers and Filter Device Objects | 136 |
| 6.5.2 | Function Drivers and Functional Device Objects | 136 |
| 6.5.3 | Bus Drivers and Physical Device Objects | 137 |
| 6.5.4 | Legacy Device Drivers and Control Device Objects | 138 |
| 6.6 | KMDF Object Model | 139 |
| 6.6.1 | Methods, Properties, and Events | 139 |
| 6.6.2 | Object Hierarchy | 141 |
| 6.6.3 | Object Attributes | 144 |
| 6.6.4 | Object Context | 145 |
| 6.6.5 | Object Creation and Deletion | 146 |
| 6.7 | KMDF I/O Model | 147 |
| 6.7.1 | I/O Request Handler | 149 |
| 6.7.2 | I/O Queues | 152 |
| 6.7.3 | I/O Request Objects | 154 |
| 6.7.4 | Retrieving Buffers from I/O Requests | 155 |

| | |
|---|------------|
| 6.7.5 I/O Targets | 156 |
| 6.7.6 Creating Buffers for I/O Requests | 157 |
| 6.7.7 Canceled and Suspended Requests | 158 |
| 6.7.8 Completing I/O Requests | 160 |
| 6.7.9 Self-Managed I/O | 161 |
| 6.7.10 Accessing IRPs and WDM Structures | 161 |
| | |
| Chapter 7 Plug and Play and Power Management | 163 |
| 7.1 Plug and Play and Power Management Overview | 163 |
| 7.2 Device Enumeration and Startup | 164 |
| 7.2.1 Startup Sequence for a Function or Filter Device Object | 165 |
| 7.2.2 Startup Sequence for a Physical Device Object | 166 |
| 7.2.3 Device Power-Down and Removal | 167 |
| 7.3 WMI Request Handler | 172 |
| 7.4 Synchronization Issues | 173 |
| 7.4.1 Synchronization Scope | 175 |
| 7.4.2 Execution Level | 177 |
| 7.4.3 Locks | 178 |
| 7.4.4 Interaction of Synchronization Mechanisms | 179 |
| 7.5 Security | 180 |
| 7.5.1 Safe Defaults | 180 |
| 7.5.2 Parameter Validation | 180 |
| 7.5.3 Counted UNICODE Strings | 181 |
| 7.5.4 Safe Device Naming Techniques | 181 |
| | |
| Chapter 8 Kernel Mode Installation and Build | 183 |
| 8.1 WDK Build Tools | 183 |
| 8.2 Build Environment | 185 |
| 8.3 Building a Project | 186 |
| 8.4 Building Featured Toaster | 187 |
| 8.4.1 Makefile and Makefile.inc | 187 |
| 8.4.2 The Sources File | 188 |
| 8.4.3 The Build | 190 |
| 8.5 Installing a KMDF Driver | 190 |
| 8.5.1 The WDF Co-Installer | 191 |
| 8.5.2 The INF | 191 |
| 8.5.3 INFs for KMDF Drivers | 192 |
| 8.5.4 wdffeatured.inf | 192 |

| | | |
|-------|--------------------------------------|-----|
| 8.6 | Catalog Files and Digital Signature | 193 |
| 8.7 | Installing Featured Toaster | 194 |
| 8.8 | Testing a KMDF Driver | 196 |
| 8.8.1 | PREfast | 196 |
| 8.8.2 | Static Driver Verifier | 197 |
| 8.8.3 | KMDF Log | 198 |
| 8.8.4 | KMDF Verifier | 198 |
| 8.8.5 | Debugging a KMDF Driver | 198 |
| 8.8.6 | Kernel Debugging | 200 |
| 8.8.7 | KMDF Driver Features | 201 |
| 8.9 | Debugging Macros and Routines | 203 |
| 8.10 | WDF Debugger Extension Commands | 204 |
| 8.11 | Using WPP Tracing with a KMDF Driver | 205 |
| 8.12 | Using WinDbg with Featured Toaster | 205 |
| 8.13 | Versioning and Dynamic Binding | 208 |

Chapter 9

Programming Drivers for the Kernel

| | | |
|-------|--|------------|
| | Mode Driver Framework | 211 |
| 9.1 | Differences Between KMDF and WDM Samples | 216 |
| 9.2 | Macros Used in KMDF Samples | 218 |
| 9.3 | KMDF Driver Structure and Concepts | 219 |
| 9.3.1 | Object Creation | 220 |
| 9.3.2 | Object Context Area | 221 |
| 9.3.3 | I/O Queues | 222 |
| 9.3.4 | I/O Requests | 224 |
| 9.4 | A Minimal KMDF Driver: The Simple Toaster | 224 |
| 9.4.1 | Creating a WDF Driver Object: DriverEntry | 225 |
| 9.4.2 | Creating the Device Object, Device Interface, and I/O Queue: EvtDriverDeviceAdd | 227 |
| 9.4.3 | Device Object and Device Context Area | 229 |
| 9.4.4 | Device Interface | 231 |
| 9.4.5 | Default I/O Queue | 232 |
| 9.4.6 | Handling I/O Request: EvtIoRead, EvtIoWrite, EvtIoDevice Control | 233 |
| 9.5 | Sample Software-Only Driver | 235 |
| 9.5.1 | File Create and Close Requests | 235 |
| 9.5.2 | Additional Device Object Attributes | 237 |
| 9.5.3 | Setting Additional Device Object Attributes | 240 |

| | | |
|-------------------|--|------------|
| Chapter 10 | Programming Plug and Play and Power Management | 243 |
| 10.1 | Registering Callbacks | 243 |
| 10.1.1 | Sample Code to Register Plug and Play and Power Callbacks | 245 |
| 10.2 | Managing Power Policy | 248 |
| 10.2.1 | Code to Set Power Policy | 249 |
| 10.3 | Callbacks for Power-Up and Power-Down | 250 |
| 10.4 | Callback for Wake Signal Support | 251 |
| | | |
| Chapter 11 | Programming WMI Support | 253 |
| 11.1 | WMI Architecture | 253 |
| 11.2 | Registering as a WMI Data Provider | 254 |
| 11.3 | Handling WMI Requests | 255 |
| 11.4 | WMI Requirements for WDM Drivers | 256 |
| 11.5 | WMI Class Names and Base Classes | 257 |
| 11.6 | Firing WMI Events | 260 |
| 11.7 | Troubleshooting Specific WMI Problems | 265 |
| 11.7.1 | Driver's WMI Classes Do Not Appear in the \root\wmi NameSpace | 265 |
| 11.7.2 | Driver's WMI Properties or Methods Cannot Be Accessed | 266 |
| 11.7.3 | Driver's WMI Events Are Not Being Received | 267 |
| 11.7.4 | Changes in Security Settings for WMI Requests Do Not Take Effect | 267 |
| 11.8 | Techniques for Testing WMI Driver Support | 268 |
| 11.8.1 | WMI IRPs and the System Event Log | 269 |
| 11.8.2 | WMI WDM Provider Log | 269 |
| 11.9 | WMI Event Tracing | 269 |
| | | |
| Chapter 12 | Programming KMDF Hardware Driver | 273 |
| 12.1 | Support Device Interrupts | 274 |
| 12.1.1 | Creating an Interrupt Object | 274 |
| 12.1.2 | Code to Create an Interrupt Object | 275 |
| 12.1.3 | Enabling and Disabling Interrupts | 276 |
| 12.1.4 | Code to Enable Interrupts | 276 |
| 12.1.5 | Code to Disable Interrupts | 277 |

- 12.1.6 Post-Interrupt Enable and Pre-Interrupt
 Disable Processing 277
- 12.2 Handling Interrupts 278
 - 12.2.1 Code for EvtInterruptlSr Callback 279
 - 12.2.2 Deferred Processing for Interrupts 281
- 12.3 Mapping Resources 283
 - 12.3.1 Code to Map Resources 284
 - 12.3.2 Code to Unmap Resources 288

Chapter 13 Programming Multiple I/O Queues and Programming I/O 291

- 13.1 Introduction to Programming I/O Queues 291
- 13.2 Creating and Configuring the Queues 293
 - 13.2.1 Code to Create Queues for Write Requests 294
 - 13.2.2 Code to Create Queues for Read Requests 296
 - 13.2.3 Code to Create Queues for Device I/O
 Control Requests 297
- 13.3 Handling Requests from a Parallel Queue 298
 - 13.3.1 Code to Handle I/O Requests 299
 - 13.3.2 Performing Buffered I/O 301
- 13.4 Forwarding Requests to a Queue 302
- 13.5 Retrieving Requests from a Manual Queue 303
 - 13.5.1 Code to Find a Request 304
- 13.6 Reading and Writing the Registry 308
 - 13.6.1 Code to Read and Write the Registry 309
- 13.7 Watchdog Timer: Self-Managed I/O 312
 - 13.7.1 Self-Managed I/O Device Startup and Restart 313
 - 13.7.2 Self-Managed I/O During Device
 Power-Down and Removal 314
 - 13.7.3 Implementing a Watchdog Timer 315

Appendix Driver Information Web Sites 323

Bibliography 331

Index 333

PREFACE

This book provides the technical guidance and understanding needed to write device drivers for the new Windows 7 Operating System. It takes this very complex programming development, and shows how the Windows Driver Framework has greatly simplified this undertaking. It explains the hardware and software architecture you must understand as a driver developer. However, it focuses this around the actual development steps one must take to develop one or the other of the two types of drivers. Thus, this book's approach is a very pragmatic one in that it explains the various software APIs and computer and device hardware based upon our actual device handler development.

There has been great progress in the art of creating and debugging device drivers. There is now a great deal of object-oriented design techniques associated with the driver frameworks that are available to the device driver developer. Much of the previous grunt work, thank goodness, is now being handled by the latest device development framework Windows Driver Foundation (WDF). We will be covering both the user mode and kernel mode of device driver development. WDF has excellent submodels contained within it, called the User Mode Driver Framework and the Kernel Mode Driver Framework.

It is really great to see a Windows Driver Framework involved in the creation of Windows Device Drivers. I started working with Windows in 1990 and we primarily used the Win32 System APIs to communicate and control the Windows Operating System for our applications. We used the Device Driver Kit (DDK) to create the Windows drivers. Because I had my own company to create application software, I obviously was very concerned about the time it took to develop application software, and the robustness of the application. There were more than 2,000 Win32 APIs to be used for this task.

Then in about 1992, Microsoft came out with the Microsoft Framework Classes (MFC). In these 600+ classes, most of the Win32 APIs were encapsulated. Of course, prior to this, around 1988, the C++ compiler came out, and Object Oriented Programming started to come

into its own. By using the MFC Framework, we could produce more application software faster and with better quality. My return on investment (ROI) went up, and I made more money. This sure made a believer of me in the use of frameworks. I used MFC until the .NET Framework came out, and for the last nine years I have been using this great collection of classes. All along, Microsoft was working to bring this same kind of software development improvements to developing device drivers. We came from the DDK, to the Windows Driver Model, to the Windows Driver Foundation Framework.

Therefore, this book shows how to create Windows 7 Device Drivers using the Windows Driver Foundation Framework. This should give us driver developers a little more sanity when meeting our deadlines.

The book is broken into three major parts as follows:

- **Part I, “Device Driver Architecture Overview”**—This part lays out the architecture involved in both software and hardware for device handler development. It also covers the driver development environment needed for driver development, for both types of drivers that are normally developed—that is, User Mode and Drivers. This section also covers the two Windows driver frameworks that are most commonly used for driver device development today, which are part of the Windows Driver Framework (WDF). These two Windows Driver Frameworks are the User Mode Driver Framework (UMDF) and the Kernel Mode Driver Framework (KMDF).

- **Part II, “User Mode Drivers”**—This part outlines the approach, design, development, and debug of User Mode Drivers. This part takes the driver programmer from start to finish in developing User Mode Drivers. We primarily use the User Mode Driver Framework for all of this work. The code is done in C++ because it is the best way to develop these types of drivers. Discussions are based on a USB User Mode Driver that we will develop using the UMDF. We will use a USB hardware learning kit from Open Systems Resources, Inc. (OSR). This provides a hardware simulation to test our User Mode Drivers. This part is primarily stand-alone and could be read and used without reading any other parts of the book. However, you will probably want to read Part I to get a feel for what we are using.

- **Part III, “Kernel Mode Drivers”**—This part outlines the approach, design, development, and debug of Kernel Mode Drivers. The intent again is to take the driver programmer from start to finish in developing Kernel Mode Drivers. For this section, we primarily use the Kernel Mode Driver Framework for all of this work. The code is done in C because this is the best way to develop these types of drivers. Discussions are based on a Kernel Mode Driver that we develop using the KMDF. We use a Peripheral Component Interconnect (PCI) hardware learning kit from OSR. This provides a hardware simulation to test our Kernel Mode Drivers. The section is also primarily stand-alone and could be read and used without reading any other parts of the book. Again, you will probably want to read Part I to get a feel for what we are using.

ACKNOWLEDGMENTS

I am most grateful to my editor Bernard Goodwin at Pearson Education for giving me the opportunity to write this book. His support during the preparation was great. I would also like to thank his assistant Michelle Housley for her timely fashion in getting me reference books and material. Also, I would like to thank John Herrin, Video Project Manager at Pearson Education, for support and help in creating the book video. Thanks to Michael Thurston, my development editor, for making the book sound very polished.

This page intentionally left blank

ABOUT THE AUTHOR

Ronald D. Reeves, Ph.D., is founder and president of Software Genesis, LLC, a software development and consulting company based in Brighton, Michigan. Dr. Reeves has some forty years of experience in designing and developing computer hardware and software applications. He holds degrees in engineering and computer science and is a nationally recognized author, consultant, and teacher.

If you have questions, comments, or suggestions for improving this book, we would like to hear from you. You can contact the author by U.S. Mail or by email at the following addresses:

Dr. Ronald D. Reeves
PO Box 2425
Brighton, MI 48116
Email: software.genesis@att.net

This page intentionally left blank

INTRODUCTION

Device drivers are where the rubber meets the road, and are very specialized pieces of software that allow your application programs to communicate to the outside world. Any communications your Windows 7 makes to the outside world requires a Device Driver. These devices include such things as mouse, display, keyboard, CD-ROMS, data acquisition, data network communication, and printers. However, Microsoft has written and supplied a great many drivers with the Windows 7 Operating System. These drivers support most of what we call the standard devices, and we will not be covering them in this book.

This book is about how we create device drivers for the nonstandard devices—devices that are not typically found on standard PCs. Quite often, the market is too small for Microsoft to create a standard device driver for these types of devices—such things as data acquisition boards, laboratory equipment, special test equipment, and communications boards.

This discussion will highlight the significant features of interest to the device driver developers. Figure I.1 shows a general block diagram of Windows 7. We develop more detailed block diagrams in the discussions in various parts of the book.

In Figure I.1 the user applications don't call the Windows 7 Operating System Services directly. They go thru the Win32 subsystem dynamic-linked libraries (DLL). The User Mode Device Drivers, discussed later, go through this same communication channel.

The various Windows 7 services that run independently are handled by the Service Processes. They are typically started by the service control manager.

The various Windows 7 System Support Processes are not considered Windows 7 services. They are therefore not started by the service control manager.

The Windows 7 I/O Manager actually consists of several executive subsystems that manage hardware devices, priority interfaces for both the system and the applications. We cover this in detail in Parts II and III of this book.

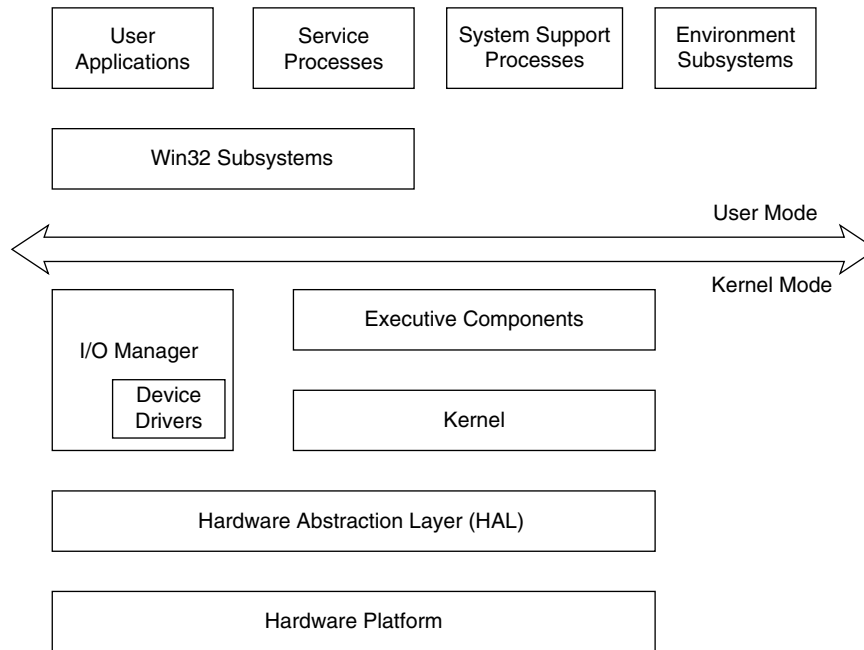


Figure I.1 System Overview Windows 7

The Device Driver block shown in the I/O Manager block is primarily what this book is all about—that is, designing, developing, and testing Windows 7 Device Drivers. The drivers of course translate user I/O function calls into hardware device I/O requests.

The Hardware Abstraction Layer (HAL) is a layer of code that isolates platform-specific hardware differences from the Windows 7 Operating System. This allows the Windows 7 Operating System to run on different hardware motherboards. When device driver code is ported to a new platform, in general, only a recompile is necessary. The device driver code relies on code (macros) within HAL to reference hardware buses and registers. HAL usage in general is implemented such that inline performance is achieved.

The Windows 7 performance goals often impact device driver writers. When system threads and users request service from a device, it's very important that the driver code not block execution. In this case, where the driver request cannot be handled immediately, the request must be

queued for subsequent handling. As we will show in later discussions, the I/O Manager routines available allow us to do this.

Windows 7 gives us a rich architecture for applications to utilize. However, this richness has a price that device driver authors often have to pay. Microsoft, realizing this early on some 14 years ago, started developing the driver development models and framework to aid the device driver author. The earliest model, the Windows Driver Model (WDM) had a steep learning curve, but was a good step forward. Microsoft has subsequently developed the Windows Driver Foundation (WDF) that makes developing robust Windows 7 drivers easier to implement and learn. This book is about developing Windows 7 Device Driver using WDF.

This page intentionally left blank

- [read Passions and Tempers: A History of the Humours pdf, azw \(kindle\), epub, doc, mobi](#)
- [Bold Spirit: Helga Estby's Forgotten Walk Across Victorian America pdf, azw \(kindle\)](#)
- [read Mastering Data Analysis with R](#)
- [Silk Road here](#)
- [read Immortality online](#)
- [click Knit Wit: 30 Easy and Hip Projects \(Hands-Free Step-By-Step Guides\) pdf, azw \(kindle\)](#)

- <http://paulczajak.com/?library/Passions-and-Tempers--A-History-of-the-Humours.pdf>
- <http://www.1973vision.com/?library/Deeply--Desperately--Lucy-Valentine--Book-2-.pdf>
- <http://chelseaprintandpublishing.com/?freebooks/Oxford-Case-Histories-in-Rheumatology--Oxford-Case-Histories-.pdf>
- <http://patrickvincitore.com/?ebooks/Lair-of-Dreams--The-Diviners--Book-2-.pdf>
- <http://damianfoster.com/books/Compromised.pdf>
- <http://flog.co.id/library/The-Concise-Book-of-Yoga-Anatomy--An-Illustrated-Guide-to-the-Science-of-Motion.pdf>