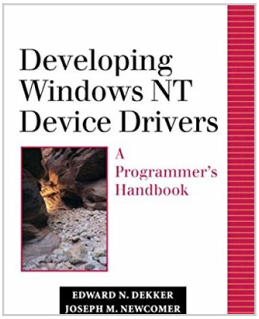

Developing Windows NT Device Drivers: A Programmer's Handbook

By Edward N. Dekker, Joseph M. Newcomer

Developing Windows NT Device Drivers: A Programmer's Handbook By Edward N. Dekker, Joseph M. Newcomer

Developing Windows NT Device Drivers: A Programmer's Handbook offers programmers a comprehensive and in-depth guide to building device drivers for Windows NT. Written by two experienced driver developers, Edward N. Dekker and Joseph M. Newcomer, this book provides detailed coverage of techniques, tools, methods, and pitfalls to help make the often complex and byzantine "black art" of driver development straightforward and accessible. This book is designed for anyone involved in the development of Windows NT Device Drivers, particularly those working on drivers for nonstandard devices that Microsoft has not specifically supported. Because Windows NT does not permit an application program to directly manipulate hardware, a customized kernel mode device driver must be created for these nonstandard devices. And since experience has clearly shown that superficial knowledge can be hazardous when developing device drivers, the authors have taken care to explore each relevant topic in depth. This book's coverage focuses on drivers for polled, programmed I/O, interrupt-driven, and DMA devices.The authors discuss the components of a kernel mode device driver for Windows NT, including background on the two primary bus interfaces used in today's computers: the ISA and PCI buses. Developers will learn the mechanics of compilation and linking, how the drivers register themselves with the system, experience-based techniques for debugging, and how to build robust, portable, multithread- and multiprocessor-safe device drivers that work as intended and won't crash the system. The authors also show how to call the Windows NT kernel for the many services required to support a device driver and demonstrate some specialized techniques, such as mapping device memory or kernel memory into user space. Thus developers will not only learn the specific mechanics of high-quality device driver development for Windows NT, but will gain a deeper understanding of the foundations of device driver design. 0201695901B04062001

Developing Windows NT Device Drivers: A Programmer's Handbook

By Edward N. Dekker, Joseph M. Newcomer

Developing Windows NT Device Drivers: A Programmer's Handbook By Edward N. Dekker, Joseph M. Newcomer

Developing Windows NT Device Drivers: A Programmer's Handbook offers programmers a comprehensive and in-depth guide to building device drivers for Windows NT. Written by two experienced driver developers, Edward N. Dekker and Joseph M. Newcomer, this book provides detailed coverage of techniques, tools, methods, and pitfalls to help make the often complex and byzantine "black art" of driver development straightforward and accessible. This book is designed for anyone involved in the development of Windows NT Device Drivers, particularly those working on drivers for nonstandard devices that Microsoft has not specifically supported. Because Windows NT does not permit an application program to directly manipulate hardware, a customized kernel mode device driver must be created for these nonstandard devices. And since experience has clearly shown that superficial knowledge can be hazardous when developing device drivers, the authors have taken care to explore each relevant topic in depth. This book's coverage focuses on drivers for polled, programmed I/O, interrupt-driven, and DMA devices.The authors discuss the components of a kernel mode device driver for Windows NT, including background on the two primary bus interfaces used in today's computers: the ISA and PCI buses. Developers will learn the mechanics of compilation and linking, how the drivers register themselves with the system, experience-based techniques for debugging, and how to build robust, portable, multithread- and multiprocessor-safe device drivers that work as intended and won't crash the system. The authors also show how to call the Windows NT kernel for the many services required to support a device driver and demonstrate some specialized techniques, such as mapping device memory or kernel memory into user space. Thus developers will not only learn the specific mechanics of high-quality device driver development for Windows NT, but will gain a deeper understanding of the foundations of device driver design. 0201695901B04062001

Developing Windows NT Device Drivers: A Programmer's Handbook By Edward N. Dekker, Joseph M. Newcomer Bibliography

- Rank: #2323327 in Books
- Published on: 1999-04-09
- Original language: English
- Number of items: 1
- Dimensions: 9.20" h x 2.00" w x 7.60" l, 4.40 pounds
- Binding: Hardcover
- 1280 pages

Editorial Review

Amazon.com Review
Developing Windows NT Device Drivers is an authoritative and clearly written resource on how to write device drivers for Windows NT. The book begins with an excellent high-level overview of how Windows NT device drivers work and how to create them. The text concentrates on "generic" device drivers written in C and excludes specialized drivers for graphics, file system, and network hardware.

Eventually, the book turns to device registers, device memory, and different PC busses (such as PCI). A section on I/O Request Packets (IRPs) and interrupt handling within Windows NT shows how to do asynchronous I/O. The authors offer a simple "Hello World" example for a device driver and present various debugging techniques.

Subsequent chapters deepen the reader's knowledge on topics such as device I/O, synchronization (including spin locks), device-driver initialization and cleanup, and direct memory access (DMA). These chapters also instruct you on how to access hardware ports and interrupt processing (a crucial topic) and how to move device memory into system memory (along with a working example). Discussion of more specialized topics--ISA and PCI busses, serialization, driver threads, and the advantages of the new "layered" driver model--follows.

Authors Edward Dekker and Joseph M. Newcomer offer plenty of excellent real-world advice. (Material on how to log device-driver events and manage the infamous Windows "Blue Screen of Death" is indispensable.) They present a "hardware simulator" that lets readers develop device drivers without an actual hardware device. The book closes with information on Windows 2000, universal serial bus (USB) devices, the Win32 driver model, and over 300 pages of reference material, including device-driver kernel functions. Overall, this comprehensive text provides a solid introduction to the way Windows NT device drivers interact with hardware; it gives you all you need to start building custom device drivers. --*Richard Dragan*

Review
"A gentle introduction... an attempt to encapsulate everything and will be fairly high in the pile of documentation on your workbench" -- *Dr. Dobb's Electronic Review of Computer Books*

Read the full review for this book.

Compared to other Windows NT device-driver development books, *Developing Windows NT Device Drivers* is hardware-oriented. A lot of real hardware-related information is presented, including small circuit diagrams and views about throwing hardware at debug problems with extender cards and bus/logic analyzers using products from various tool vendors such as Vmetro and Catalyst Enterprises. A lot of products are mentioned, including libraries and debugging tools from vendors such as Bluewater Systems and Compuware/NuMega.

Dekker and Newcommer could have subtitled this book, "My life in device-driver development." There are a lot of digressions, including brief and mostly interesting references to DEC system-10, RS/6000, Windows 3.x, CP/M, and a CMU symmetric multiprocessor research project. Everything related to driver development gets a mention, even if briefly, according to the authors, "for completeness." The authors appear to be quite

fond of footnotes, many of which are off topic comments. A lot of the asides include general advice, from which compiler to use (Microsoft will have no sympathy, if you don't use Visual C++) to which books to read on specific buses. (They like the mindshare books, as do I.) The authors are adamant on some subjects such as ISA plug-and-play (just don't do it) and Hungarian notation. I never trust other people's code and take it at face value. If you don't like Hungarian notation, ignore it. --*Regan Russell*, Dr. Dobb's Electronic Review of Computer Books -- *Dr. Dobb's Electronic Review of Computer Books*

From the Inside Flap
Welcome to the World of Device Drivers

This is a book on how to write Device Drivers. Device Drivers are those very specialized pieces of software that live inside the operating system and allow it, and thus your programs, to communicate to the outside world. Every communication NT makes to the outside world, including the keyboard, mouse, display, disks, CD-ROMS, and printers requires a Device Driver. Most of the key ones have been written by Microsoft, and NT supports all the "standard" devices.

But what about "nonstandard" devices? This doesn't mean the devices don't conform to a standard; what it means is that these are devices that are not found on "standard" PCs. Often they are specialized devices whose market is far too small for Microsoft to devote any energy to supporting. These could be devices such as drivers that support IEEE-488 communications (used for laboratory equipment and test equipment), analog-to-digital data input boards, digital-to-analog data output boards, specialized communications boards, and the like.

The core problem in Windows NT is that if you don't have a Device Driver you can't talk to a device. In Windows 3.x and Windows 9x you can cheat; you can write an application program that goes "for the bare metal" and directly manipulates the hardware interface to the device. But in Windows NT, because of the requirements for being certified at C2 Security (an important certification commercially and for government sales), an application program is simply not permitted to manipulate the hardware directly. So you are confronted with the problem of writing a device driver.

That's what this book is all about. How This Book Happened

One evening after a long day at Comdex, Ed and I (jmn) were sitting around discussing our Latest Adventures. Ed was telling me about all the really cool, largely non-obvious (not undocumented, but obscurely documented) things he'd learned about writing device drivers. I said to him, "Ed, you should write a book". (At this point I was well into the writing of Win32 Programming, so writing books was at the top of my consciousness). He looked at me and said "Joe, I don't know how to write a book". I immediately responded "Well, I don't know how to write a device driver!" The conclusion was obvious. So we wrote a proposal to Addison-Wesley, and you are holding the result. What We Cover

The purpose of this book is to allow you to write NT Device Drivers for a new device that you would want to connect to an NT machine. These include polled, programmed I/O, interrupt-driven, and DMA devices. Modern interrupt drivers add complexity, because interrupts may be shared by several devices. DMA has changed from the use of the PC's onboard DMA controller to external DMA controllers and Bus Mastering controllers. Many drivers for modern devices need access to onboard memory, and we show how to do that. In some cases, you might want to write a "user level" system that does most of the work, calling the I/O subsystem only to initiate certain input or output transactions, and using it to handle interrupts. We show how to construct these drivers.

We talk about how to build robust, portable, multithread- and multiprocessor-safe drivers. We talk about what you have to do to ensure that your driver won't bring the system crashing down around the user. We lay

out the protocols you must use in communicating to the rest of the system; if you don't follow these fairly exactly, the system won't work as expected. It may crash, it may corrupt the disk, it may ignore the device, it may lose information on input or output, or it may misinterpret the intention of the application programmer who calls the services of the driver. Writing a Device Driver is fairly exacting. We try to capture much of the Art of building a device driver here, as well as the Technology.

In addition, we talk about how to actually build drivers, that is, the mechanics of compilation and linking; what they have to do to register themselves with the system, and most importantly, how to debug them. Most of the useful techniques for debugging device drivers is either undocumented or difficult to find; we've summarized our experiences in this book. What We Don't Cover

There is only so much that one book or set of books can cover. We've chosen our goals to cover what most device driver writers want or need. But there are several aspects we don't cover at all:

Graphics drivers. These include display drivers and printer driver. The Graphics Driver contains the code to render an image into the internal frame buffer of the Video Adapter or printer engine. Most of the effort goes into understanding the incredibly clever hacks needed to get optimal efficiency on a particular piece of hardware. Graphics drivers are in the range of tens of thousands of lines of code. Generally, unless you work for a display card vendor or printer vendor, you will never encounter a need to write one of these. And while there is often a desire to support some old printing or plotting device, frankly, the cost of developing one of these drivers would buy several modern equivalents of those old printers. Chapter 25 contains a very high-level overview of graphics drivers. To do them justice would take another book about this size.

File System Drivers would be worthy of an entire book on their own. In fact, this has already been done (see Further Reading). To write an entire file system is far more complex and is beyond the scope of this book. We've never written a file system, and are not now qualified to write about how to do it. Network Drivers. Network drivers are worthy of an entire book devoted just to that subject (there is a pattern here). The chances that most device writers will ever need to write a full network driver are fairly slim. At the lowest level (for example, the card driver) a network card driver bears a strong resemblance to most other devices, but there are some additional complications that require a deep understanding of network protocols. There is just not enough space available in this book to cover network drivers.

In addition, recognize that this is software. There are dozens of ways to do anything. We present the core methods, and possibly one or two interesting alternatives. We cannot tell you every possible way to do every possible thing (for one thing, we haven't done every possible thing in every possible way ourselves!) Many techniques are unique to particular devices, and in some cases our ability to talk about some of the more interesting advanced techniques is circumscribed by various Non-Disclosure Agreements we have signed. What we hope to give you here is a sound footing in basic driver technology and illustrate a few of the more interesting advanced techniques that you may need. But the coverage here is not "complete". If we started to write a "complete" book on version n of NT, we would not have delivered the manuscript before version n+1 was released. This wouldn't help anyone. The Physics Model for Software

No, not "Software Physics". That's something else entirely. The "Physics Model" refers to a comment by a physics professor, who said "We have an advantage in physics; we teach it by an ever more refined set of lies". What he was referring to is that when physics is first taught, you learn about Newtonian physics. In this model, we have phenomena such as the wave model of light, the particle model of light, and so on. Physical objects are real, and their masses interact according to specified laws of gravity or of energy transfer. But relativity theory tells us that Newtonian physics is really relative to a particular inertial frame, and what is really going on is something different. In Quantum Mechanics, you learn that at a certain scale, physical objects don't exist, and phenomena like the Heisenberg Uncertainty Principle come into play, and you get an

explanation of what is really going on. Then there's Quantum Electrodynamics, in which the real model of what is going on is explained, and then...well, it's been too long since our physics courses and anyway you should have the idea by now.

We apply this same principle to teaching software. So don't be surprised if there are some apparent discrepancies between the illustrations and text on one page and the illustrations and text on a later page. We're trying to avoid giving you infinite detail too early and thus avoid being too confusing. Ideally, the earlier text or illustration is a subset of the later one. About URLs

While many people know about URLs, not everyone might be familiar. A URL is a Universal Resource Locator, a designator for a page of information on the World Wide Web. A URL has a form like mumble/here/and/there/and/everywhere.html mumble/here/and/there/and/everywhere.htm mumble

To access a Web page for which we give a URL, you must have a Web Browser, such as Microsoft Internet Explorer or NetScape (to name the two most popular browsers for Win32 platforms), and a connection to the Internet. We are not going to explain those details here, but if you don't have access, you want to find an Internet Service Provider (ISP) who can provide this service to you. We have investigated a number of interesting Web sites that provide useful information for Device Driver writers. We have included these URLs in various places in the book.

Once you have your Internet connection, you bring up your browser and you type the URL, and as if by magic, the Internet tells you it doesn't exist. Well, actually, if you typed it correctly, and the Web page wasn't removed, you will actually get the page. We can't guarantee that all of the URLs we give will continue to exist. We're giving you the best information we have as we went to press, but the Web changes daily, and it is impossible to be instantly current in a print publication. Documents on the Web

We often cite URLs as a source of documents. The Web page will usually give you a link, such as the name of the document or a button, that allows you to download the document. Documents come down in a variety of formats, but the most popular formats are Microsoft Word (.doc) files and Adobe Acrobat (.pdf) files. If you have Microsoft Word, it is easy to read the documents; if you don't, you will need to download the Microsoft Word Viewer. For Acrobat, you need an Acrobat Viewer. This is available as a download from Adobe. We give the URLs under Further Reading.

Documents on the Web are often compressed, usually using the PKZip utility from PKWare. We have uncompressed all the information we included on the CD-ROM, but if you download a document you will need the PKZip utility. This is available in both public-domain versions and registered versions. The folks at PKWare have done Good Things for the computing community. Support them by buying a licensed version. The licensed version is not that expensive; if you're a device driver writer, you can recover the cost very quickly. Icons for insertions

We have done a number of text insets. These elaborate on points that have specific audiences, and rather than put everything inline in the text, we have pulled some discussions out into these sidebar-like annotations. To clue you as to their relevance, we have a series of icons that we use to indicate the contents.

One indicates a potential bug. Sometimes it is a bug in the documentation, sometimes it is a bug in a particular release of the operating system.

Another indicates some informational aside.The information may be useful to you, but it is not as important as the main-line text.

Yet another indicates a potential pitfall. Often it is a compatibility issue, such as a Win16/Win32 difference

that is otherwise undocumented, or obscurely documented, or a difference between two Win32 versions. Occasionally, we use it to indicate other possible failures that would otherwise be hard to discover. These failures include obscure or undocumented limitations, or places where you are likely to get into trouble In some cases, the behavior of a Windows operation is not "intuitively obvious", and if you do what you think is "right", it won't work as expected.

Another icon indicates that there is a potential hazard. You should take note of this when using the related material.

Finally, yet another icon indicates the authors are expressing a (usually controversial) opinion. We want you to know it is an opinion, rather than a statement of some fact. You may even disagree with the opinion, but one advantage of being an author is that we get to say, in public, how we feel about certain issues. A note on "Hungarian Notation"

This whole section should have one of the "flame" insets, but it is too long and too important to reduce to a mere sidebar.

The so-called "Hungarian Notation" (developed by a Hungarian programmer at Microsoft) is one of the worst ideas to have hit software development in many years. This is that horrid notation that tries to encode the data type in the name, for example, "nCount" indicating an integer count. This is a mistake; only two other popular languages in history ever encoded data type in a name, FORTRAN and BASIC, and both of them abandoned it as a fundamentally bad idea. Fortunately, the coders of the NT kernel had the good sense to avoid this notation. You will not find it used in any of our examples, except those user-level examples where we have to interface to Microsoft's data structures that use the convention, or when some published standard has fallen prey to this insidious notation.

Why is Hungarian Notation bad? Aesthetics aside (it is singularly ugly), it creates maintenance nightmares, which is a cardinal sin in our opinion. Software is hard enough to maintain without introducing artifices that make that more difficult. For example, you might declare a short signed integer counter and call it "nCount". But you may then discover that you need more than 32,767 values, but less than 65,535 values in the counter. So in a sane world, you would do the obvious: change the declaration from "short" to "unsigned short" or equivalently "WORD" (the declaration WORD always means "unsigned short"). But according to Microsoft's standards, you now need to change the name to "wCount". A global substitution won't work, because there may be other contexts in which the variable "nCount" is simply a short. If you don't change it, you'll wonder why you get a compiler warning if you compare what is "obviously" a signed number with another signed value. And if you need more than 64K values, you might want to redeclare it a "long" or "unsigned long" ("DWORD") in which case you need to change the name to "lCount" or "dwCount". If you don't change the name, you'll wonder why assigning a 16-bit variable wCount to another 16-bit variable wOldCount or passing it to a function that requires a 16-bit parameter generates a compiler warning about value truncation. If the names did not encode the type, you would not make this kind of error because you would have to refer to the declaration to find the type. You say that changing a single name is trivial? Not so. And the possibility of making an error and not changing the name, or perhaps changing the name in too many places, introduces a complication to maintenance that is unnecessary. It is important to understand how real maintenance is done by real programmers in the real world, rather than legislating how a theoretical model of maintenance ought to be, in some ideal but unrealizable world.

But there are other, even more significant, maintenance headaches. If this value was in a structure, you can't change the name because this would force all the users of the structure to change their code. So you will find in the Microsoft interfaces names like "wCount" that were 16-bit unsigned integers in Win16, but changed to 32-bit unsigned integers in Win32. But for compatibility, the prefix had to remain the same (the classic one

for GUI programmers is "wParam", which the prefix suggests is a 16-bit unsigned value but which is really a 32-bit unsigned value! And, in the world of 64-bit NT, will be a 64-bit unsigned value!) It is not uncommon in the Microsoft interfaces to find declarations in structures like "int dwCount", "LPVOID dwData", "DWORD lpszData", "WORD nCount", and "int wCount". Many of these represent the failure to fully transition from the 16-bit world to the 32-bit world. None of these would have been a problem if Hungarian Notation had not existed. By using a naming convention that says the name does encode the type, then violating that convention, you create programs that require additional care in maintenance. If that additional care, which should be completely unnecessary, is not exercised, you can get unmaintainable code, or code that contains subtle bugs.

Historically, this naming convention arose in the days before compilers did cross-module type checking on function calls, and it made it possible for the programmers to "visually check" that parameters had the correct types. Using ANSI compilers with full function prototypes, this need is gone (and has been gone for many years) and there is little reason to use this convention. We strongly discourage introducing any Hungarian Notation to any component you write. It is an unfortunate historical aberration that has no place in modern programming practice.

Perhaps the best summary we saw of this was in a book on programming, where the author titled the section "Hungarian Notation: Just Say No". We loved this heading, and considered plagiarizing it, but decided not to. It is too good, and he deserves to have it to himself. But we'll leave you with our expression: "Friends don't let friends program in Hungarian Notation". NULL and NUL

The value NULL is a pointer to nothing. Although traditionally zero, in fact the ANSI C Standard does not require it to be zero. And while nearly every C program would break if it were changed from zero, that is because people like to write code of the form "if(!pointer)" instead of the (actually, only correct) form "if(pointer == NULL)". (The reason for the former is that on the first PDP-11 compilers, which had no optimization, the compiler could generate one fewer instructions for that form, which was important when the application had only 64K of address space to share for code and data. In modern compilers on modern architectures there is no difference in the code generated, and the former is simply a type-unsafe computation based on a tradition that a NULL pointer is the same as a 0 value).

NUL, on the other hand, is a character value. There is no character value NULL. In fact, the use of the NULL macro to represent a character NUL is incorrect, although we see it all the time. For example, the assignment stringi = NULL when string is a char (ANSI) or wchar_t (Unicode) array. The value NUL is the 3-letter abbreviation for the character whose value is 0x00. When used in the context of Unicode, it represents the character whose value is 0x0000. In fact, many C compilers formally define NULL as ((void *)0), in conformance with the ANSI standard, which makes it impossible to actually assign it to a character lvalue. Thus the assignment we just gave would actually cause a compilation error, and in any decent C compiler this would be an error, not a warning. Instead, the correct assignment would be either stringi = '' (ANSI) or stringi = L'' (Unicode). The INs and OUTs of parameters

You will find all of the function headers specified by Microsoft have an apparent keyword attached to each: IN, OUT, or both. Starting with NT 5.0, some parameters may be followed by the keyword OPTIONAL. These keywords don't really exist; in fact, they are macros that have empty bodies, so they are effectively thrown away by the preprocessor. They are, however, incredibly useful as documentation aids. A reference parameter that is declared OUT means that it will be written but not read; but if it is declared IN OUT it means you must initialize it before making the call, and it will be updated upon return. Little things like this help a lot. You will find that much of our code uses them also. Unicode

Internally, Windows NT uses the Unicode character set. This is a 16-bit "universal" character code designed

to support multiple languages concurrently. The familiar 8-bit character set is referred to as the ANSI character set, and is a proper subset of the Unicode set. This introduces some interesting wrinkles you will have to be conscious of, for example, that there is no longer a one-to-one relationship between the number of bytes and the number of characters. Furthermore, even Microsoft is a bit inconsistent at the driver level. At the application level, the phrase "characters" always means "the number of characters", and an application running as a native Unicode application requires two bytes per character, but the APIs always use or return character counts. In the kernel, sometimes character counts are used and sometimes byte counts are used. We even point out where Microsoft has incorrectly confused characters and bytes in documenting a critical data structure (the UNICODE_STRING). But basically, you will want to "think Unicode" while within the kernel.

Unicode is not the same as the "multibyte character set" mechanism (MBCS) supported by Microsoft and other C libraries. The MBCS allows for embedded "shift" bytes that indicate whether the string is current in one-byte, two-byte, or potentially longer encodings. Unicode strings are always 16-bit characters, and contain no embedded shift sequences. Indentation Style

Yes, we recognize this is an issue of deeply-held beliefs, defended in general with incredible ferocity. The style used largely throughout the book is one which was developed by one of the authors (jmn) over many years. I (jmn) strongly favor this style. Ed and I actually have quite different styles of indentation (but since I got to annotate all the code I got to choose the style). I believe that unlike many of the competing styles of indentation, this is the only one based on fundamental principles of human cognition. While I don't expect to convert anyone to this style, I believe that understanding the principles may encourage thinking more deeply about indentation styles and how they are used in practice.

Think about this: how many times have you had to draw little arrows connecting the { }s of a C program? How many times have you had to print out hardcopy solely for this purpose? How many times have you gotten it wrong? Many years ago, I used a compiler for a language called SAIL, developed at the Stanford AI Lab. It was a classic Algol-class language with additional features of supporting backtracking search, associative memory, and many other bells and whistles. The compiler, and its executables, ran on a 256Kword (roughly 1MByte) DECSystem-10 or DECSystem-20. This language had an incredibly useful feature: you could follow any begin keyword with a string which was the "block name", and any end keyword with the same string. If the strings didn't match, the compiler complained, including both names and line numbers in the error message. This Was Wonderful.

After struggling with trying to find a "C style" that was right for me, I recalled how useful the block name feature was, and programmed my editor so that when I typed an open-brace I got a comment following it, into which I could type the "block name". When I typed a close-brace, the editor would find the matching open brace, indent the close brace to be directly under it, and copy the comment. While not as good as the checking done by the compiler it has greatly reduced brace errors in my code to virtual non-existence even on the first compilation, and has made finding the mismatched braces in those few cases where I get sloppy to time expressed in single digit seconds. But more significantly, it makes it possible to easily read large blocks of code, such as switch statements and loops, that span many screens or printed pages. I know that if I see a comment on a closing brace that there is a corresponding open brace with the same comment. It is trivial to find it. This conforms to the human cognitive processes which don't handle counting and nesting very well but can trivially match "flat" patterns. Since many of our examples span many pages, a bare close-brace is virtually incomprehensible without these annotations.

So think about it. Does your indentation style match human perception, or is it a representation of a machine-oriented perception? I, for one, have little interest in making life easy for machines; I'd rather optimize my own time.

-jmn Acknowledgments

No book this complex can be written by two people in the time we had to write it. We are grateful to a large number of people for their help, support, suggestions, and tolerance during this process.

Mike Hendickson and Ben Ryan of Addison Wesley Longman, who between them saw the book through its initial stages.

Carter Shanklin, our current editor, who helped us through to completion.

Laura Michaels for copy editing and style suggestions.

Avinash Chopde, Scott Thieret, Ron Reeves, and Bruce Rosen whose comments on the book helped us write more clearly on some topics.

Special thanks to Rick Jones and Richard Page whose suggestions and comments helped make this a far better book.

The folks at OSR, whose NT Newsletter is full of useful tips.

Jim McCollum for use of his example of a driver that detected a subtle problem in DPC queuing that would cause a system crash (and the entire driver community should all thank him for having discovered this, and the workaround! A Nice Bit Of Work).

Brian Bussey and Frank de Alderete of Technology Exchange Company who provided the support to develop a course based on this book.

The folks in the comp.os.ms-windows.programmer.nt.kernel-mode newsgroup, many of our questions have been answered without asking, just by searching the archives. Jim Boemler (jboemler@halcyon) who maintains the file pcicode.h, which we used for our PCI Explorer.

And especially the folks of MindShare, most notably Tom Shanley and Don Anderson, who have done an utterly invaluable series of books on PC architecture. We needed some of that data to get our explanations right. (I think that we now each own the complete series, and as they extend it we will extend our collections. This is a great set of books! Run, do not walk, to your nearest technical bookstore and get them). We also thank them for their permission to include much of the detailed technical data that appears in our USB chapter from their book on the USB Architecture.

We also want to particularly thank Jeff Ross, of VMetro, Incorporated, for the loan of the VMetro board we used for illustrations in the debugging chapter.

I need to thank my co-author, without whose encouragement (and strong push) I never would have started this project. Joe has clearly done the hard part of this book. I also need to thank Joe for all he has taught me over the years we have worked together. I can name few people who have taught me as much. I also thank my parents; they started me on the path to this book. When I was a small boy, long before first grade, my father started asking me how things work. This would continue for many years. We would see a machine. Sometimes we would be in the car and drive past something, a road building machine or a factory, he would ask me to explain how the machine works, or the factory builds a product, or how the product the factory builds works. After I answered the question he would give his answer. When he started, he didn't expect me to have the right answer. He taught me how to come up with possible answers. This also inspired me to read more so I would know the answers to the next question. My answers have been getting better as I got older.

(He may have regretted this, by the time I started college he observed that if asked for the time I would give instructions on building clocks).

My mother, who put up with my early book addiction (three trips to the library a week was about right, they only let me take 10 books at a time) as well as my electronics habit and the need for small electronic parts (not all of them were tangled in the carpet).

It seems to be tradition for an author to thank his dog, so I will thank my dog who has been as understanding as possible. He would clearly have preferred walking (well, running in circles for him) through the woods instead of sleeping in my office as I worked on this book. He helped provide the comic relief I needed at times. He has appeared with his Kong at the times I needed a break from the book, and be satisfied if I spend just a few minutes playing fetch. Well it resembles fetch, I throw the Kong once and he runs past me with the Kong and he is happy if I move my arm to attempt to take it from him as he runs past at high speed.

--end

As usual, I need to thank my Little Gray Cat (a.k.a. Bernadette G. Callery) for her patience and endurance during Yet Another Book (this is my third). Mew! She also provided invaluable assistance for this book by providing the lower trace shown on the oscilloscope of Figure 9.25 (by holding the Channel 2 probe).

And my co-author for some truly amazing spicy meals during the assorted weeks we spent working together at his place. His years of experience in writing NT Device Drivers are what made this book possible, and his many hours of research on the topics we covered has been invaluable. He has clearly done the hard part of this book.

Like Ed, my parents and my Uncle Bill set me on this path many years ago by encouraging reading and experimentation; Erector sets and TinkerToy sets were an important part of my life, as were the salvaged TV sets (for parts). I just have a bigger toy set to play with these days.

--jmn About the Authors: Joseph M. Newcomer and Edward N. Dekker

Dr. Joseph M. Newcomer is a consultant, instructor, and author with 33 years' experience in computer science. Joe has been a contributor to Dr. Dobb's Journal, published over 70 articles and technical reports, and written several books, including Win32 Programming (co-authored with Brent Rector; Addison-Wesley, 1997). He has worked on operating systems, device drivers, compilers, computer music (MIDI), real-time and embedded systems, and programming tools. Joe has been active in the Microsoft online forums and has been named a Most Valuable Professional (MVP) by Microsoft.

Edward N. Dekker offers consulting services through his company Eclectic Engineering, Inc. Ed specializes in systems programming and real-time systems. Ed has over 20 years' computer software experience. Ed has spent the majority of his time in the past few years focusing on device driver work for Windows NT. He has written device drivers for a variety of operating systems including: Windows NT, Windows (3.0 and 3.1), DOS, VMS, RSX-11M, UNIX, and VXWORKS. Ed's experience also includes programming and design of real-time systems, networking, data-base, programming tools, electronic mail systems, application programs, and consulting on software design. Contacting Us

If you find errors, or have suggestions for clarifications or material you'd like to see, we'd love to hear from you. We will probably be putting up errata sheets, new software, and the like on our Web sites as well. While we've done our best to interpret some very complex documentation on a very complex problem, we too are fallible. And while we think we've found all the problems (especially with those paragraphs written late at

night or early in the morning, depending on your viewpoint), a book, like the software it describes, can't be proven correct, it can only be tested. We hope we have made your job of writing a device driver easier.
Joseph M.Dekker, New Ipswich NH, dekker@eclectic-eng, eclectic-eng
Further Reading/Software Sources

Adobe Systems, Acrobat Viewer. Available from adobe/prodindex/acrobat

Campbell, Mary K. (ed.), Legacies, Institute of Electrical and Electronic Engineers, 1994. ISBN 0-7803-9996-X.

Hummel, Robert L., Programmer's Technical Reference: Data and Fax Communication, Ziff-Davis, 1993. ISBN 1-56276-077-7.

Microsoft Corporation, Word Viewer. Available from microsoft/word/internet/viewer/viewer97

Nagar, Rajeev, Windows NT File System Internals, O'Reilly and Associates, 1997. ISBN 1-56592-249-2.

Newcomer, Joseph M., Letter to the Editor, Post-Gazette, November 27, 1991, quoted in Legacies. Legacies was a special publication of the IEEE celebrating their life members. One of the contributors lives in Pittsburgh, saw my letter, and said that it characterized much of how engineers became engineers. He asked permission to reprint it. I was particularly pleased because this book is a set of interviews with some of the greatest engineers of the century, all of whom are Life Members of the IEEE, on how they became engineers and what their careers meant.

Open System Resources, Inc., The NT Insider, currently available by free subscription from: Open System Resources, Inc., 105 State Route 101A, Suite 19, Amherst, New Hampshire, 03031.

PKWARE, Incorporated, PKZip. Available from: PKWARE, Inc., 9025 N. Deerwood Dr., Brown Deer, WI 53223, (414) 354-8699, pkware.

Schildt, Herbert, The Annotated ANSI C Standard, Osborne/McGraw-Hill, publication date unknown but suspected to be 1993 from the notes in the Introduction. ISBN 0-07-881952-0.

VMetro, Inc., VMETRO Inc., 1880 Dairy Ashford, Suite 535, Houston, TX 77077, vmetro. For More Help

The authors hope you will get an introduction to NT drivers from this book. If you need additional help, the authors of this book are consultants and instructors. In addition to this book we have also written and teach a course on Windows NT device drivers. The course is offered through Technology Exchange Company. We offer consulting and software development services through our own companies.

For information about this course, contact: Technology Exchange Company, 20 Burlington Mall Road, Suite 210, Burlington MA 01830-4123, TechnologyExchange. Course Description

Windows NT 4.0: Device Drivers, A 5-day course. Key Benefits Learn the basic principles of Windows NT Device Driver programming. Understand how device drivers fit into the Windows NT system. Use the Microsoft driver build environment and Numega's SoftIce to produce a device driver. Develop strategies for solving driver problems. Write demonstration Windows NT Device Drivers. Who Should Attend

This course is designed for programmers responsible for developing new Windows NT drivers. Students must be fluent in C and an understanding of peripheral interface hardware and I/O programming for at least one hardware platform is necessary. Knowledge of operating systems principles is helpful, but issues such as Interrupt Services, Memory Management, DMA, Intel and Digital Alpha Hardware Architecture, Symmetric

Multiprocessor (SMP) machines, caches, and virtual memory are covered during the course overview.
Course Overview

This course presents an accelerated introduction to programming Windows NT device drivers. Students learn the basic principles of Windows NT I/O system architecture. Course content encompasses Drivers, Class Drivers, MiniDrivers, the Hardware Abstraction Layer (HAL), I/O Request Packets (IRPs), Deferred Procedure Calls (DPCs), Interrupt Service Routines (ISRs), the I/O manager, and the use of the Registry to maintain device information. Knowledge gained in this course is a prerequisite for writing WDM Drivers for Windows NT 5 and Windows 98.

0201695901P04062001

Users Review

From reader reviews:

Florence Wiggins:

Book will be written, printed, or created for everything. You can recognize everything you want by a publication. Book has a different type. As you may know that book is important matter to bring us around the world. Beside that you can your reading talent was fluently. A guide Developing Windows NT Device Drivers: A Programmer's Handbook will make you to end up being smarter. You can feel a lot more confidence if you can know about every little thing. But some of you think this open or reading any book make you bored. It isn't make you fun. Why they might be thought like that? Have you trying to find best book or suited book with you?

John Charles:

Information is provisions for folks to get better life, information currently can get by anyone from everywhere. The information can be a expertise or any news even restricted. What people must be consider if those information which is within the former life are challenging to be find than now is taking seriously which one is acceptable to believe or which one typically the resource are convinced. If you get the unstable resource then you have it as your main information we will see huge disadvantage for you. All those possibilities will not happen in you if you take Developing Windows NT Device Drivers: A Programmer's Handbook as the daily resource information.

Clarine Davidson:

Spent a free a chance to be fun activity to complete! A lot of people spent their spare time with their family, or their very own friends. Usually they performing activity like watching television, likely to beach, or picnic from the park. They actually doing same task every week. Do you feel it? Do you want to something different to fill your free time/ holiday? Could be reading a book may be option to fill your totally free time/ holiday. The first thing you will ask may be what kinds of reserve that you should read. If you want to consider look for book, may be the guide untitled Developing Windows NT Device Drivers: A Programmer's Handbook can be good book to read. May be it can be best activity to you.

Jeanne Pratt:

As a scholar exactly feel bored in order to reading. If their teacher requested them to go to the library or even make summary for some reserve, they are complained. Just little students that has reading's spirit or real their passion. They just do what the teacher want, like asked to the library. They go to generally there but nothing reading seriously. Any students feel that looking at is not important, boring in addition to can't see colorful photographs on there. Yeah, it is to be complicated. Book is very important for you personally. As we know that on this period, many ways to get whatever we really wish for. Likewise word says, ways to reach Chinese's country. Therefore this Developing Windows NT Device Drivers: A Programmer's Handbook can make you experience more interested to read.

Read Developing Windows NT Device Drivers: A Programmer's Handbook By Edward N. Dekker, Joseph M. Newcomer for online ebook

Developing Windows NT Device Drivers: A Programmer's Handbook By Edward N. Dekker, Joseph M. Newcomer Free PDF d0wnl0ad, audio books, books to read, good books to read, cheap books, good books, online books, books online, book reviews epub, read books online, books to read online, online library, greatbooks to read, PDF best books to read, top books to read Developing Windows NT Device Drivers: A Programmer's Handbook By Edward N. Dekker, Joseph M. Newcomer books to read online.

Online Developing Windows NT Device Drivers: A Programmer's Handbook By Edward N. Dekker, Joseph M. Newcomer ebook PDF download

Developing Windows NT Device Drivers: A Programmer's Handbook By Edward N. Dekker, Joseph M. Newcomer Doc

Developing Windows NT Device Drivers: A Programmer's Handbook By Edward N. Dekker, Joseph M. Newcomer Mobipocket

Developing Windows NT Device Drivers: A Programmer's Handbook By Edward N. Dekker, Joseph M. Newcomer EPub

BF3D59TP2RN: Developing Windows NT Device Drivers: A Programmer's Handbook By Edward N. Dekker, Joseph M. Newcomer