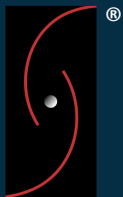


The NT Insider

A publication of OSR Open Systems Resources, Inc.



```

extern "C"
ULONG
DriverEntry(
    _In_ PVOID DriverObject,
    _In_ PVOID RegistryPath)
{
    HW_INITIALIZATION_DATA hwInitData = {0};

    OsrNVMeTraceFunctionEntry();

    //
    // TODO:
    // Read this special edition of The NT Insider dedicated to NVMe
    //
    hwInitData.HwInitializationDataSize = sizeof(HW_INITIALIZATION_DATA);
    hwInitData.AdapterInterfaceType     = PCIBus;

    hwInitData.HwFindAdapter      = OsrNVMeHwFindAdapter;
    hwInitData.HwInitialize       = OsrNVMeHwInitialize;

    //
    // Peter's at it again
    //
    hwInitData.PeterPontificates = "I Don't Care I Want My Internet"; // Page 4

    //
    // Inside NVMe Technology: How it works.
    //
    hwInitData.ArticleOne = "Introduction to NVMe Technology"; // Page 6

    //
    // When is an SRB not an SRB?
    //
    hwInitData.ArticleThree = "Win 7 vs Win 8 StorPort Miniports"; // Page 8

    //
    // Our journey to create the ultimate NVMe Driver
    //
    hwInitData.ArticleFour = "If You Want Something Done Right..."; // Page 10

    //
    // Can an $800 software package take the place of an $85K LeCroy?
    //
    hwInitData.ArticleTwo = "Software vs Hardware Bus Analyzers"; // Page 12

    hwInitData.HwBuildIo = OsrNVMeHwBuildIo;
    hwInitData.HwStartIo = OsrNVMeHwStartIo;

    hwInitData.SpecificLuExtensionSize = sizeof
    hwInitData.NumberOfAccessRanges = OSRNVME_
    hwInitData.MapBuffers = STOR_MAP_NO_BUFFER

#ifdef (NTDDI_VERSION >= NTDDI_WIN8)
    hwInitData.SrbTypeFlags = SRB_TYPE_FLAG_
                          SRB_TYPE

    hwInitData.AddressTypeFlags = ADDRESS_TYPE_FLAG_
    hwInitData.HwUnitControl = OsrNVMeHwUnitControl;

#endif

```

Other Special Features

- [The NVMe Issue...P.2](#)
- [Get Social with OSR...P.3](#)
- [Seminar Schedule...P.24](#)

Published by
OSR Open Systems Resources, Inc.
105 Route 101A, Suite 19
Amherst, New Hampshire USA 03031
(v) +1.603.595.6500
(f) +1.603.595.6503

<http://www.osr.com>

Consulting Partners
W. Anthony Mason
Peter G. Viscarola

Executive Editor
Daniel D. Root

Contributing Editors
Scott J. Noone
OSR Associate Staff

Send Stuff To Us:
NTInsider@osr.com

Single Issue Price: \$15.00

The NT Insider is Copyright ©2014 All rights reserved. No part of this work may be reproduced or used in any form or by any means without the written permission of OSR Open Systems Resources, Inc.

We welcome both comments and unsolicited manuscripts from our readers. We reserve the right to edit anything submitted, and publish it at our exclusive option.

Stuff Our Lawyers Make Us Say

All trademarks mentioned in this publication are the property of their respective owners. "OSR", "OSR Online" and the OSR corporate logo are trademarks or registered trademarks of OSR Open Systems Resources, Inc.

We really try very hard to be sure that the information we publish in *The NT Insider* is accurate. Sometimes we may screw up. We'll appreciate it if you call this to our attention, if you do it gently.

OSR expressly disclaims any warranty for the material presented herein. This material is presented "as is" without warranty of any kind, either expressed or implied, including, without limitation, the implied warranties of merchantability or fitness for a particular purpose. The entire risk arising from the use of this material remains with you. OSR's entire liability and your exclusive remedy shall not exceed the price paid for this material. In no event shall OSR or its suppliers be liable for any damages whatsoever.

It is the official policy of OSR Open Systems Resources, Inc. to safeguard and protect as its own, the confidential and proprietary information of its clients, partners, and others. OSR will not knowingly divulge trade secret or proprietary information of any party without prior written permission. All information contained in *The NT Insider* has been learned or deduced from public sources...often using a lot of sweat and sometimes even a good deal of ingenuity.

OSR is fortunate to have customer and partner relations that include many of the world's leading high-tech organizations. As a result, OSR may have a material connection with organizations whose products or services are discussed, reviewed, or endorsed in *The NT Insider*.

Neither OSR nor *The NT Insider* is in any way endorsed by Microsoft Corporation. And we like it that way, thank you very much.

An Issue Entirely Dedicated to...

NVMe

Every once in a while we get our act together and are able to come up with an entire issue dedicated to one specific topic that we think will be of interest to the community we serve. In this instance, that topic is Non-Volatile Memory Express, or NVMe.

As you'll read in the pages within, NVMe is the most recent expression of a high-speed storage specification for Solid State Drives (SSDs) on the PCI Express bus. It's getting a lot of attention in the storage community as a whole, and in the Windows storage space in particular, for both what is and is not supported. And, just as in our popular training seminars where "we practice what we teach", this issue provides a teaching moment, and a reasonable stepping off point for kernel devs and product managers alike, who are interested in or contemplating projects involving the development and support of NVMe-based devices. Enjoy!

LIKE WHAT YOU SEE?

It's been several years since we've been able to manage to publish 4 issues of *The NT Insider* – while we're proud of this, what we really want to know is what YOU think.

So, if you appreciate our efforts, we'll really appreciate you giving us a like, mention, follow or what-have-you on your favorite social media app.

Follow us!



WE KNOW WHAT WE KNOW

We are not experts in everything. We're not even experts in everything to do with Windows. But we think there are a few things that we do pretty darn well. We understand how the Windows OS works. We understand devices, drivers, and file systems on Windows. We're pretty proud of what we know about the Windows storage subsystem.

What makes us unique is that we can explain these things to your team, provide you new insight, and if you're undertaking a Windows system software project, help you understand the full range of your options.

And we also write kick-ass kernel-mode Windows code. Really. We do.

Whether you're looking for training, consulting, or somebody for the development of your project end-to-end... why not fire-off an email and find out how we can help. If we can't help you, we'll tell you that, too.

Contact: sales@osr.com

Get Social with OSR

Real-Time Updates

Follow us!



Just in case you're not already following us on Twitter, Facebook, LinkedIn, or via our own "osrhints" distribution list, below are some of the more recent contributions that are getting attention in the Windows driver development community.

Do As I Say, Not As I Do (article + video)

Even seasoned pros need to consider following their own advice.

<https://www.osr.com/blog/2014/10/22/do-as-i-say/>

Overlooked Features: Device Name Spaces (article + video)

Not an every day "need to know" topic, but a hidden gem for sure.

<https://www.osr.com/blog/2014/10/08/device-name-spaces/>

Setting the WinDbg Symbol Search Path (article + video)

For when "fix your symbols" isn't sage enough advice...

<https://www.osr.com/blog/2014/10/01/setting-the-windbg-symbol-search-path/>

WinHEC Returns!

It was dead. Now it isn't.

<https://www.osr.com/blog/2014/09/26/winhec-returns/>

What is Arbitrary Thread Context? (article + video)

A confusing topic for newbie driver developers explained.

<http://www.osr.com/blog/2014/09/08/arbitrary-thread-context-article-video/>

Scan memory for ASCII strings

"s -sa StartAddress EndAddress" (-su for Unicode). Good for when you don't know what else to look for!

<http://twitter.com/OSRDrivers/status/537262179821842434>

Using IoCreateDeviceSecure?

Don't forget you STILL need to specify FILE_DEVICE_SECURE_OPEN...this IS NOT set by default.

<http://twitter.com/OSRDrivers/status/530025827090444288>

COME JOIN US!

Writing WDF Drivers:
Core Concepts

Palo Alto, CA
12-16 January

seminars@osr.com

TECHNICAL TIPS FROM OSR VIA EMAIL

Not everyone has the time to keep up with various flavors of social media - and in some parts of the world, it's not even possible.

For such folks, OSR created the OSRHINTS mailing list, where we will duplicate our Twitter posts of technical hints, tips, tricks and other useful industry or OSR-related updates.

To options to join:

1. Send a blank email to: join-osrhints@lists.osr.com
2. Visit OSR Online and sign-up interactively: <http://www.osronline.com/custom.cfm?name=listJoin.cfm>

Peter Pontificates: I Don't Care, I Want My Internet



Here in the States we just finished celebrating Thanksgiving Day. The new media here loves this holiday, and for so many reasons. They regularly refer to it as "The Most American of Holidays," presumably because it's only celebrated in the US. Never mind that Canada celebrates Thanksgiving Day too, even if they did steal the idea from us and try to camouflage the fact by having it on a different day.

The idea of Thanksgiving Day is that you get together with family and friends, share a meal, spend time enjoying each other's company, and spend a few moments being thankful for – the simple things that really matter in life. You're supposed to focus on what you have, not what you

don't have. What's good about your life, not what's lacking. It's a great idea, in concept. As we'd say in the computer biz, it's a terrific design. The implementation, on the other hand, often falls a bit short.

This year, here on the East Coast of the US, our Most American of Holidays got screwed-up because of a major snow storm that dumped about half a meter of heavy wet snow directly on my house (See **Figure 1**). The fact that we had this snow storm on Thanksgiving Day was taken as proof by a number of my friends that Major Climate Change had **already begun**, and that the Earth was headed for immediate annihilation. Initially, this made me pretty upset because I did **not** want to miss the rest of this season of [Homeland](#). If I acted fast, perhaps there was still time to get Rosetta to download the next episode or two onto [67P/C-G](#) where I can view it "sometime later."

Turns out I didn't need to worry. The panic here about snow on Thanksgiving Day was just that: Panic over nothing. Snow on Thanksgiving Day? It's been known to happen. In fact, it's been known to happen a lot. Maybe not in the past few years (who can really remember much further back than that), but definitely in the past. In fact, there's an American folk song know as [Over the River and Through the Wood](#), that talks about traveling to "Grandfather's house" for Thanksgiving Day "through white and drifted snow." That song (originally a poem) was written in 1844. Snow on Thanksgiving Day in New England was pretty common back in the 19th Century, apparently. So much for this snow storm being a harbinger of Major Climate Change. When I mentioned these facts to my friends, they told me that it was perhaps the fact that we do **not** frequently get snow by Thanksgiving Day that indicates we're all about to be drowned by Major Climate Change events. They said I should be worried. I poured myself another hot buttered rum. But I digress.

So, the snow started on Wednesday afternoon, the day before Thanksgiving Day. By 5PM, there were several inches of snow on the ground at my house and most of the area had lost power. Given that I live in a rural area and that I do not find it either quaint or amusing to live without electricity, I have a backup generator. So, losing power isn't a big deal for me. What was worse, far worse in fact, was that I **lost my Internet connectivity**. I can live with Major Climate Change. But no Internet? Losing Internet connectivity is a Big Deal.

I get up on Thanksgiving Day. The usual trash is on TV. I go to check my Twitter feed to see what's going on in the world. Ahhhh... no Internet. Not to be discouraged, I decided to just check the local weather forecast... on the Internet. So that wasn't going to work. Then it occurred to me: My iPad has mobile broadband. 4G support even.

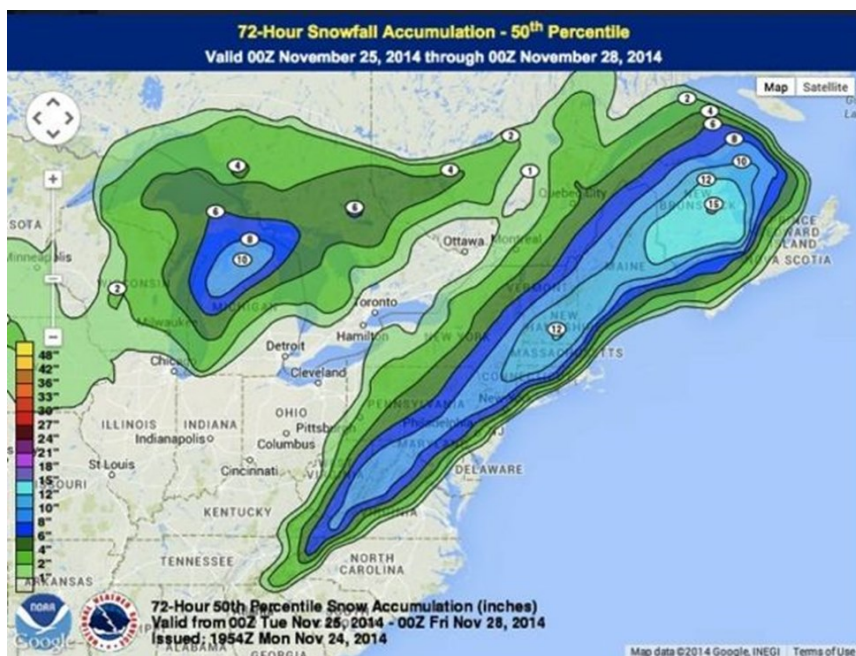


Figure 1 – Lame Excuse for Losing Internet Connectivity

[\(CONTINUED ON PAGE 5\)](#)

Peter Pontificates... (Cont.)

[\(CONTINUED FROM PAGE 4\)](#)

Apparently I was not the only person with this idea, because the wonders of 4G/LTE were S-L-O-W. And the signal where I live wasn't exactly strong.

So, there I was, standing on my couch with my arms in the air pointing my iPad to the North (where the signal was a **bit** stronger) trying to find out when my power was going to be restored. Well, again, really, what I cared most about was when my Internet was going to be restored. Because standing at awkward angles to get a good 4G signal on my iPad really wasn't a reasonable long-term solution for Internet connectivity.

In frustration, I grabbed a turkey leg from the refrigerator. I cut a slice of apple pie and mostly ignored it. I made believe I didn't care. I moped. I talked to my dog. The power stayed off all day Thursday (Thanksgiving Day). And so did the Internet. No Netflix. No Amazon. No forums. No random visits to make fun of something on Wikipedia.

Thank goodness for TiVo. If it hadn't gone off on its own and record a bunch of episodes of [Tattoo Nightmares](#) and random movies, I would have gone out of my mind. When there's no Internet, you'd be surprised how comforting it is to see how well [Big Gus](#) can cover-up a youthful indiscretion.

On Friday morning, the day after Thanksgiving Day, the power **and the Internet** were still both out. My wife was asleep. But I had a plan:

*Hi Honey,
I've gone to the office. Not much to do
here and lots to do at work.
Love,
Me*

That's not my actual handwriting, of course. My actual handwriting is much more manly and ... well, whatever. But the point is the same: OSR World Headquarters never lost power. OSR had Internet connectivity. Thus I developed a sudden compunction to go to work, even on a holiday.

And while I was at work, for a few blissful hours the world was once again my oyster. I fired-up TweetDeck, followed-up on forums, fetched the long-term weather forecast for my home town, and just because I could, I checked the weather in Hong Kong

[\(CONTINUED ON PAGE 23\)](#)

WINDOWS INTERNALS & SOFTWARE DRIVERS For SW Engineers, Security Researchers, & Threat Analysts

Scott is extremely knowledgeable regarding Windows internals. He has the communications skills to provide an informative, in-depth seminar with just the right amount of entertainment value.

- Feedback from an attendee of THIS seminar

Next Presentation:

Seattle, WA
16-20 February

Twenty-First Century Storage

Introduction to NVMe Technology

Big changes in the world of storage happen at the rate of about one per decade. Many of you will remember the IDE interface, which was later called ATA, and which was even later became known as PATA. The IDE/ATA/PATA interface was first documented in 1989, and it took until 1994 for the standard to be agreed upon and completed. About ten years later, in 2003, SATA was initially introduced.

Now it's 2014 and SATA is nice, but it just isn't optimal anymore. It's time for a whole new interface. An interface that takes into account not only the advances in storage technology, but also the advances in how computers work these days. That interface is NVM Express, which everyone refers to as NVMe.

NVMe is a storage interface specification for Solid State Drives (SSDs) on the PCI Express (PCIe) bus. The standard defines both a register-level interface and a command protocol used to communicate with NVMe devices. An interesting aspect of NVMe is that, in current implementations, the NVMe controller and drive are fully integrated. So, you don't buy an NVMe controller and then one or more NVMe drives. Rather, the drive and the controller form a single unit that plugs into the PCIe bus. While this may change in the future, all current NVMe products follow this model.

In this article, we'll describe some of the technical features of NVMe focusing on how it's been designed for one thing in mind: Speed.

Why It's Cool – Getting Commands To and From the Device

So what's cool about NVMe? The coolest thing about it is that everything in the spec has been designed with the goal of getting the most performance out of a storage device running on a modern computer system. The interface between the host and the SSD is based on a series of paired Submission and Completion queues that are built by the driver and shared between the driver (running on the host) and the NVMe device. Interestingly enough, the queues themselves can be located either in traditional host shared memory or in memory provided by the NVMe device.

Once the Submission Queues and Completion Queues are configured, they are used for almost all communication between the driver and the NVMe device. When new entries are placed on a Submission Queue, the driver tells the device about them by writing the new tail pointer to a hardware doorbell register. This register is specific to the Submission Queue being updated.

When the NVMe drive completes a command, it puts an entry on a Completion Queue (that has been previously associated with the Submission Queue from which the command was retrieved) and generates an interrupt. After the driver completes processing a group of Completion Queue entries, it signals this to the NVMe device by writing the Completion Queue's updated head pointer to a hardware doorbell register. This is shown in **Figure 1** (for IDF 14).

[\(CONTINUED ON PAGE 7\)](#)

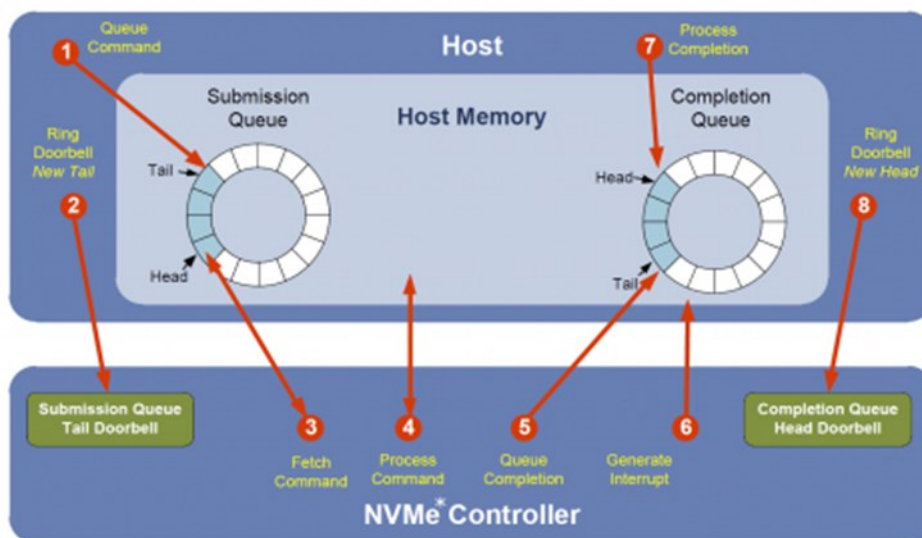


Figure 1 – Processing an NVMe I/O Command

Introduction to NVMe... (Cont.)

[\(CONTINUED FROM PAGE 6\)](#)

There are separate queues for Administration operations (such as creating and deleting queue or updating firmware on the device) and for ordinary I/O operations (such as Read and Write). This ensures that I/O operations don't get "stuck" behind long-running Admin activities. The NVMe specification allows up to 64K individual queues, and each queue can have as many as 64K entries! Most devices don't allow this many, but Windows Certification requires NVMe devices to support at least 64 queues (one Admin Queue and 63 I/O Queues) for server systems and 4 queues (one Admin Queue and 3 I/O Queues) for client systems.

The advantage of allowing lots of entries per I/O Submission Queue is probably obvious: The ability to start many I/O operations simultaneously allows the device to stay busy and schedule its work in the most efficient way possible. This maximizes device throughput.

However, you **might** be wondering why having so many Submission Queues and Completion Queues is advantageous. The answer to this is surprisingly simple: On multiprocessor systems, the ability to have multiple queues allows you to have one Submission and Completion Queue pair per processor (core). Having a pair of queues that are unique to each core allows the driver to process I/O initiations and completions without having to worry about contention from other processors, and thus allows these actions to be performed almost entirely without acquiring and holding any locks. What makes this work **particularly** well is that NVMe strongly favors the use of MSI-X for interrupts. MSI-X allows the NVMe device to direct its interrupts to a particular core. This is such a significant advantage for NVMe that support for MSI-X is required for NVMe controller Certification on Windows Server systems. With MSI-X (and proper driver support), a request that is submitted by the NVMe driver on a given CPU will result in an interrupt indicating its completion on that same CPU. The drivers DpcForlsr will also run on that same CPU. And, the Completion Queue indicating the request's status will be unique to that CPU. You have to admit: That's very cool. Lock free, and optimal for NUMA systems as well.

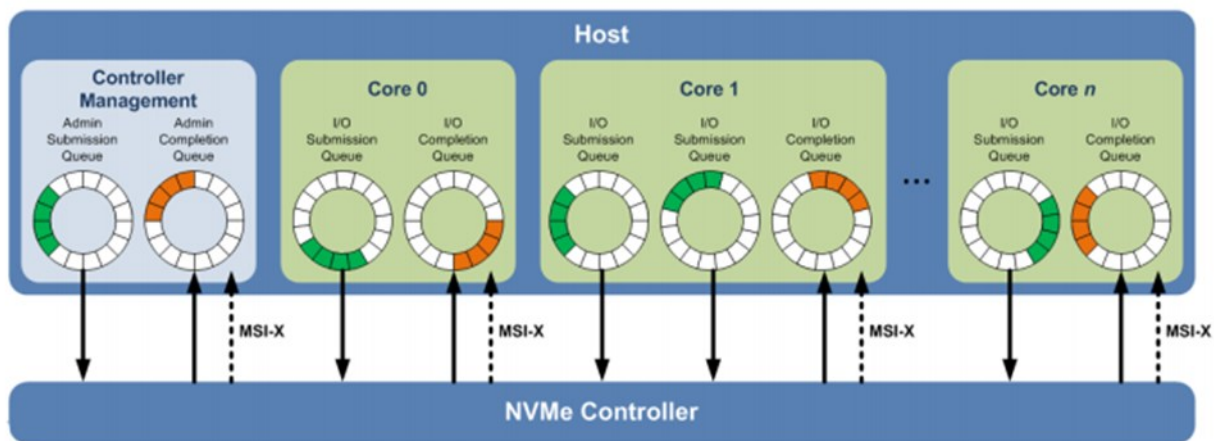


Figure 2 – NVMe Queues per Core

[\(CONTINUED ON PAGE 20\)](#)



DID YOU KNOW?

As a special thank you for attending an OSR seminar, previous students receive an extra 5% discount when registering for future OSR seminars. Just register by the payment discounts indicated on our seminars updates or website and you will automatically receive the extra discount.

When is an SRB NOT an SRB? Win7 vs. Win8 StorPort MiniPorts

Designing a driver model for an operating system is always a balancing act. Provide too much flexibility and you end up with unnecessary complexity in both the drivers and the operating system. However, if you don't provide enough flexibility you stifle the industry's ability to innovate on your platform, which either hurts adoption of your platform or pushes developers into finding unsupported ways to circumvent the model.

Windows is an interesting example of this struggle as we actually have both extremes: a flexible, native driver model that ultimately became impossible to support (I'm looking at you, WDM) and a series of locked down "mini" models that failed to evolve with the industry (SCSI Port, anyone?).

Not surprisingly, over time both of these approaches have proven problematic. The increasing complexity of the native driver model led to system stability issues. Attempts to push the driver development community to add features to their drivers that would help overall system performance (e.g. Fast Resume) fell on deaf ears. The drivers are complicated enough already, who wants to add more features? The mini-models had the opposite effect: developers were **dying** to add product differentiating features to their code, but the mini-models weren't designed to support these features and they were out of luck.

The complexity of the native driver model has, thankfully, been tamed quite nicely by the Windows Driver Framework. With WDF we get a well-defined interface that makes it easy and safe to get a simple driver working, but always with the flexibility to escape the Framework and increase our complexity.

WDF does not, however, attempt to address the issues of the mini-models. WDF is a generic driver mode for writing generic drivers, but the mini-models exist to make it easier to write drivers for *specific* device types. Microsoft has repeatedly chosen to not simply abandon these models, thus we're still stuck with locked in mini-models for certain device types.

That is not to say that these models have remained static. In fact, it is interesting to watch how each mini-model has evolved over the years to address the needs of the market. The evolution of the storage driver models has been almost painful to watch (and definitely painful to experience), though with Windows 8.1 we appear to **finally** be on track to having a single storage driver model that meets the evolving needs of the industry.

Unfortunately, there is a downside to all of this progress: a poor story for backwards compatibility. As it turns out, it is now quite tricky to write a single, high performance storage driver that optimally supports Windows 7 and later. While there are some technical reasons for this, the primary frustrations come from the StorPort build environment and a lack of clarity in the documentation. In the interest of clearing up the story a bit, we'll highlight our main issues that we've dealt with in creating our high performance NVMe controller driver for Windows 7 and later.

Good Luck with Binary Compatibility

Evolution is messy business, as is evident in the header file StorPort.h. As the driver model has evolved, there have been multiple different methods used to add APIs, change fields of structures, etc. Sometimes an attempt was made to provide single binary compatibility on all platforms, and sometimes binary compatibility was clearly not a priority

My favorite example of the chaos is the **HW_INITIALIZATION_DATA** structure, which each driver is required to initialize and pass to **StorPortInitialize**. Initially, this structure was inherited verbatim from the SCSI Port model. When virtual storage adapter support was added in Windows Server 2003 SP1, this structure needed modification to support some new, virtual adapter specific callbacks. Instead of simply changing the structure, the existing structure was copy/pasted and renamed **VIRTUAL_HW_INITIALIZATION_DATA**. Additional fields were then added to this structure to support virtual adapter operations. The virtual adapter driver still calls the same **StorPortInitialize** routine as always to initialize the driver, so how does StorPort know which one you want? By the *HwInitializationDataSize* field, of course! The driver sets this field prior to calling **StorPortInitialize** and StorPort uses the size to determine which structure the driver is passing.

In Windows 8, StorPort has some additional fields to be added to **both** of the initialization structures. Instead of simply adding the fields to both structures, the developers instead chose to abandon the multiple structures and go back to a single **HW_INITIALIZATION_DATA** structure. This structure now includes the virtual extensions as well as the Windows 8 specific fields. StorPort detects that a driver is using the new structure by checking the *HwInitializationDataSize* for the new structure's size.

[\(CONTINUED ON PAGE 9\)](#)

Win7 vs. Win8 StorPort MiniPorts... (Cont.)

[\(CONTINUED FROM PAGE 8\)](#)

This model makes it very difficult to provide a single binary that takes advantage of the new features on Windows 8 while maintaining backwards compatibility on Windows 7. A single driver can't have both structure sizes without making their own private copy of one of the structures, which opens another can of worms. And even if you **do** decide to solve this one you're not even out of **DriverEntry** yet!

Inconsistent Version Information in Headers

StorPort is constantly being updated to add features and support. Huge changes can even come in hotfixes and service packs.

For the most part, the documentation does a fine job of indicating which releases of the operating system support which changes. The header files are a different story though. They often don't provide compile time checks, meaning you can call functions or set options that don't exist or, worse, set options that *do* exist but don't work properly.

STOR_PERF_CONCURRENT_CHANNELS is a good example of the trouble this can cause. This feature was added to the headers in Windows 7 and allows the driver to avoid some significant locking in the StorPort wrapper, thus increasing driver throughput. However, the current documentation states that you must not set it prior to Windows 8 and posts online indicate that there are stability issues under stress with this option set on earlier platforms. Setting this option in your driver prior to Windows 8 is a time bomb, but the build environment will happily let you do it even when compiling for Windows 7 as your target.

Is that an SRB or an SRB?

StorPort is an evolution of SCSIPort, which was the driver model for writing storage adapter drivers back when everything was SCSI. To this day, StorPort still treats all storage devices as if they are SCSI devices, sending the drivers **SCSI_REQUEST_BLOCK** structures containing Command Data Blocks (CDBs) and targeting the controller's devices via their Bus, Target, and LUN (BTL) address.

It is no longer the 1980s and our storage devices are no longer SCSI. Given that our 1980s based driver model continues to talk to us in terms of SCSI, our storage controller drivers spend much effort translating SCSI semantics and operations into something that is specific to our bus.

Windows 8 does not do anything to change this, we're still stuck dealing with handling SCSI requests in our storage driver. However, the groundwork for a new future has been laid by the introduction of the **STORAGE_REQUEST_BLOCK**. As the name implies, this structure represents a **storage** request to the controller. Of course, the only type of storage request you can currently send is SCSI, but the structure is extensible enough to add a different technology in the future. The structure may also be extended to support device addressing schemes other than BTL.

Drivers must opt in to this new request block structure and, to maintain backwards compatibility, all StorPort entry points continue to pass a **SCSI_REQUEST_BLOCK** pointer as the parameter. The driver must determine which type of structure they actually receive based on the structures' shared *Function* member. **STORAGE_REQUEST_BLOCKS** have a function of **SRB_FUNCTION_STORAGE_REQUEST_BLOCK**, everything else is a **SCSI_REQUEST_BLOCK** (or one of its variants).

[\(CONTINUED ON PAGE 18\)](#)

DESIGN AND CODE REVIEWS When You Can't Afford Not To

Have a great product design, but looking for validation before bringing it to your board of directors? Or perhaps you're in the late stages of development of your driver and are looking to have an expert pour over the code to ensure stability and robustness before release to your client base. Consider what a team of internals, device driver and file system experts can do for you.

Contact OSR Sales — sales@osr.com

If You Want Something Done Right... Our Journey to Writing a Windows NVMe Driver

If you've been reading *The NT Insider* for a while, you probably already know that here at OSR we're keenly interested in all things to do with storage and file systems. Over the years, we've worked closely with both Microsoft and a number of storage vendors to help evolve these disciplines. So, when NVM Express started to evolve out of the previous NVMHCI working group that we had been a part of, we were eager to see the outcome.

When the first version of NVMe was released in 2011, we realized it was an important starting point. But we had seen cool technical specs come and go over the years, so we weren't about to get overly excited. But by the time the NVMe V1.1 spec was approved in 2013, we had already realized this technology was going to be significant.

Clearly, Microsoft agreed with our assessment, because Windows 8.1 (which RTM'ed in August of 2013) had a fully functioning NVMe driver included.

The importance of Microsoft's early support cannot be overstated. Microsoft has always been loath to jump on the bandwagon when new technology specifications are released. This is particularly true when it comes to storage. While those of us who like to live on the bleeding edge might bemoan this approach, it makes perfect sense: If support for some up-and-coming technology is included in Windows today, that support is going to have to remain in Windows for a good long while. And the technology has to be widely enough available, and sufficiently well tested, that it can be used by potentially millions of users world-wide. Say what you want about versions of Windows, but even the most ardent penguinite can't complain about the extensive hardware compatibility support that Windows provides.

So, Microsoft's support for NVMe in Windows 8.1 signaled that this technology was not only significant, but that it was here to stay.

It was about this time that we started hearing from colleagues and technology partners that they wanted us to provide them with an NVMe driver. Why did they want OSR to write an NVMe driver for them, when Windows was shipping with an NVMe driver as part of Windows 8.1? Well, it turns out there are a lot of reasons:

- Most users out in the world aren't running Windows 8.1 – Even forward-thinking users who've done recent upgrades are running Windows 7 (or Windows Server 2008 R2, AKA S08 R2). Microsoft doesn't provide an NVMe driver that's compatible with older systems.
- The Windows driver doesn't provide support for all the (optional) NVMe features that most hardware vendors and many large data centers need. NVMe might only have a few required commands, but there are a lot of really cool optional features. Folks want to be able to take advantage of these features. Today.
- The Windows driver can't be customized (obviously) to support vendor-specific optimizations and behaviors. Sure, NVMe is a standard. But there wouldn't be any competition if each developer of an NVMe device didn't try to gain some advantage for their technology implementation over those of others. Tailoring a driver to the specifics of how a device is implemented is an excellent way to maximize a device's potential. And, no surprise: Hardware vendors want to do this.
- In addition to the above, the Windows driver doesn't support an NVMe pass-through capability, beyond that which can be mapped from a limited set of SCSI pass-through operations. NVMe pass-through is vital for those who want to perform unique commands, especially management and administration commands. In fact, we've never talked to a client interested in NVMe technology that didn't think NVMe pass-through was vital.
- Not every NVMe device vendor or large data center wants to invest the time and money (mostly the time) to develop their own in-house NVMe driver.
- Even if they have a custom driver, most storage vendors or data center teams don't want to maintain it. Unless driver development is part of their core skill set, they'd rather not be bothered.

[\(CONTINUED ON PAGE 11\)](#)

Our Journey to Writing... (Cont.)

(CONTINUED FROM PAGE 10)

- Folks told us that if OSR wrote it, they'd be confident that the driver would be both well designed and reasonably well performing. This made us happy.

About this same time, work was also progressing on an open source NVMe driver for Windows. Some of the early promoters of NVMe worked on this driver collaboratively as part of the Open Fabrics Alliance (OFA), prior to a Windows driver being available from Microsoft. Some colleagues, clients, and partners told us that they did not trust the stability of this driver sufficiently to use it in production. For an example of two very different public assessments of this driver, see the NTDEV thread at <http://www.osronline.com/showthread.cfm?link=253628>).

Our own review and testing of the open source OFA driver (which started in early 2014) concluded that the driver suffered from some unusual design quirks, was uneven in its implementation quality, and was in general not production ready. It also suffered from having undergone multiple revisions, and needed a general clean-up. Our ultimate conclusion was that, to produce the best possible driver, a clean start was necessary. We thus decided not to contribute to the OFA effort, or even use that code as the basis for our own work, but to rather undertake our own, unique, built from scratch, NVMe driver project.

That's how we here at OSR made the decision to develop our own Windows NVMe driver. We took what we heard from vendors and data centers, and we brought all our knowledge of the Windows storage stack, I/O Subsystem architecture, and all our experience in StorPort driver development, to focus on the problem.

That driver is in internal test right now, and is scheduled to be released to external Beta Test on 12 January 2015.

The OSR NVMe driver seeks to meet the needs of those companies – NVMe device vendors or data centers – that don't want to use either the Microsoft or the OFA driver. With our NVMe Driver Solution Kit, clients get a driver that works out of the box on Windows 7 (including S08 R2) and later systems. The kit is royalty free, and comes with full source code and the rights to make any modifications a client may desire. On request, we're able to customize the driver, adding advanced NVMe features or custom features unique to a particular use. And, of course, we can provide on-going support, maintenance, and upgrades for the driver – including support for new OS versions – long into the future.

(CONTINUED ON PAGE 17)



JOIN THE OSR NVME DRIVER SOLUTION KIT BETA PROGRAM

The OSR NVMe Driver Solution Kit provides both full source code and pre-built binaries for a high-performance Windows NVMe driver, right out of the box. You may use and redistribute the OSR NVMe Driver as provided or customize it using your own software team's resources or with help from the team at OSR.

First Code Drop Available in January 2015!



\$85K or \$800?

Hardware vs. Software Bus Analyzers: Alternatives

When faced with any project related to hardware, the engineers here at OSR immediately start pushing for access to a hardware analyzer for the project. While it is generally believed that only the hardware engineers need access to this ~~key~~ equipment, we've found having one for software development to be incredibly valuable. Trace messages and single stepping in the debugger can only show you so much. If you **really** want to know if you're putting the bits in the right place you need to see things from the device's perspective.

Depending on the technology involved, picking up a hardware analyzer "just in case you need it" is a no brainer. If you're working with a USB 2.0 device, our recommendation for a hardware analyzer can be picked up for \$800(USD) (Ellisys USB Explorer 200: <http://ellisys.com/products/usbex200/buy.php>). If you're working with I2C you're in even better luck, you can pick up an analyzer for \$330 (Beagle I2C/SPI Protocol Analyzer: <http://www.totalphase.com/products/beagle-i2cspi/>).

Working with PCIe? No problem! Just grab a Gen3 x4 analyzer with the correct interposer for about \$85,000 (LeCroy Summit T3: <http://teledynelecroy.com/protocolanalyzer/protocoloverview.aspx?seriesid=357>).

OK, well, maybe that's a **little** hard to accept as reasonable, but you never know until you ask, right? And before you do, I suspect that you want to have some argument as to why you need this on your desk. Is it really that useful in writing a driver for a PCIe device? Also, because you know they're going to ask, is there a cheaper alternative?

Case Study: NVMe

Of course, being the NVMe issue of *The NT Insider*, our target device for this article is an NVMe controller. Different devices using different technologies may have more or less support in the products discussed in this article.

Hardware Analyzers: Damn they're cool!

It's easy to understand the power of a hardware analyzer once you see it in action. For example, let's discuss a pretty standard NVMe command: retrieving the NVMe controller's Identify data structure. There are seven basic steps that must be followed in order to retrieve this data, involving both **Host** and **Device** actions:

- Host** – Place an Identify Controller Command in the Admin Submission Queue (host memory)
- Host** – Ring the Admin Submission Queue doorbell (device memory)
- Device** – Read Identify Controller Command from Admin Submission Queue
- Device** – Return Identify Controller structure into requestor's buffer
- Device** – Place command status response in Admin Completion Queue (host memory)
- Device** – Write to host memory to generate the MSI-X interrupt
- Host** – Ring Admin Completion Queue doorbell (device memory)

When you're writing the driver for this, you set some structures up, write to some registers, and hope for the best. If the expected result comes back from the device, then you're pretty sure that you got things right.



Choose Your Weapon...

WANNA KNOW KMDF?

Tip: you can read all the articles ever published in *The NT Insider* and STILL not learn as much as you will in one week in our **KMDF** seminar. So why not join us!

Next presentation:

Palo Alto, CA
12-16 January

[\(CONTINUED ON PAGE 13\)](#)

Hardware vs. Software Bus... (Cont.)

(CONTINUED FROM PAGE 12)

Figure 1 – Steps 1-4 (click to enlarge)

But, you don't really know if everything is right or not, you just have faith that it is.

However, place this device on a hardware analyzer and you can see each and every one of these steps occur. Figure 1 shows steps 1-4 above, starting with a ring of the doorbell by the driver to the start of the transfer of the Identify Controller structure.

Figure 2 – NVMe Decoding Enabled (click to enlarge)

It obviously takes a considerable amount of trace spelunking and understanding in order to interpret some reads and writes in a trace as a set of higher level operations. However, you don't always need to do this manually. In our case the LeCroy's PETracer application has NVMe decoding capabilities and, once we turn on NVMe transaction decoding, PETracer does the work for us. Figure 2 shows the same point in the trace, but now we see this as an NVMe transaction.

(CONTINUED ON PAGE 14)

Hardware vs. Software Bus... (Cont.)

(CONTINUED FROM PAGE 13)

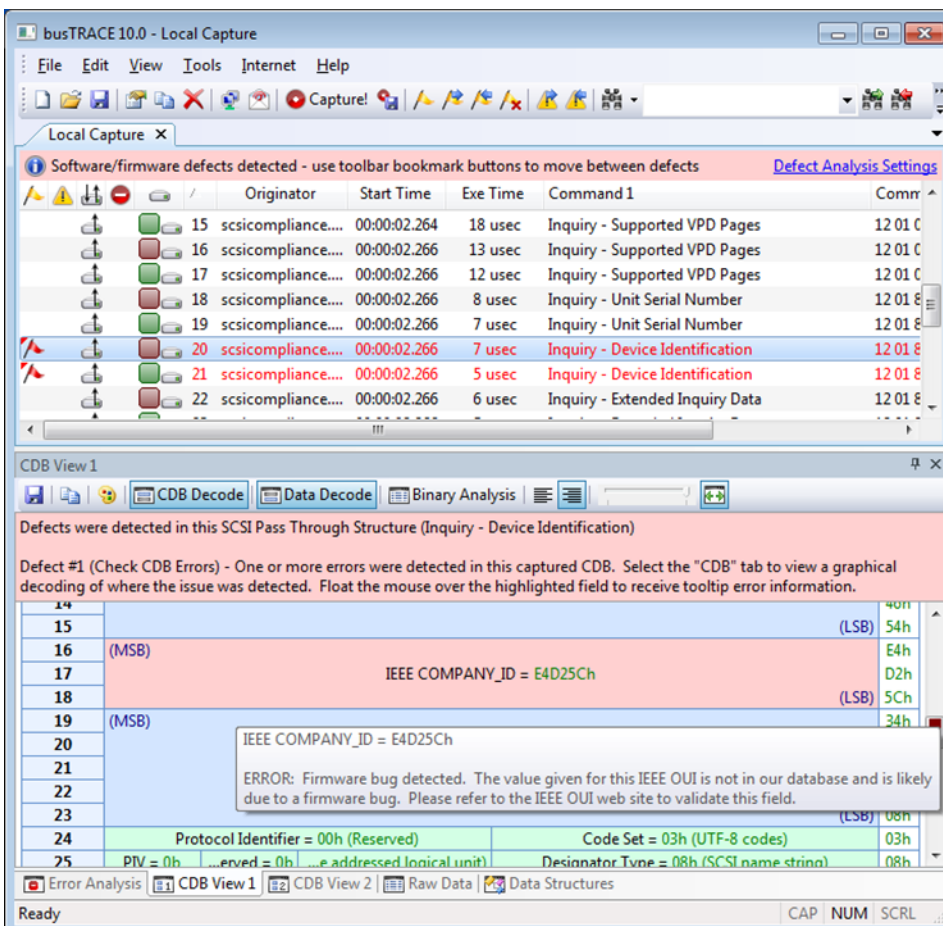
With the tracing software decoding the transactions for us, we're also able to see the entire seven step sequence in one compact view. We're even able to see an MSI-X interrupt being generated via a write from the device into host memory. As a software developer you just have to admit how cool it is to get this level of detail and how valuable that ability might be during development.

Software Only Analyzers: An Alternative?

Software analyzers are a whole different beast than hardware analyzers. Of course, software-based analyzers are undeniably less expensive and far less complicated than hardware analyzers. But for most software analyzers this is where their advantages end. Hardware-based analyzers show you an objective "device side" view of what's going on. The vast majority of software analyzers offer you little more than a pretty-printed version of the data that your driver formatted and sent. Heck, most of the time I could do that with DbgPrint, if I wanted to take the time.

Nowhere is this more evident than with some of the software USB "analyzers" that are available. These decoders are often nothing more than filter drivers that intercept USB Request Blocks and provide an interpretive display of their contents. While this can sometimes be helpful, you have to wonder if there's really enough added value to justify your time and the purchase of such a tool. This is especially true when a really great hardware analyzer for USB costs between \$400 and \$800, and most commercial software-based USB analyzers costs in the realm of \$200.

This is not to say that there are no software analyzers that are worthy of consideration. In the hardware analyzer example using NVMe that we discussed previously, we showed how helpful a PCIe bus analyzer can be. But every shop doesn't have \$85K to spend for this type of hardware. If your work is limited to a specific area, such as Windows storage controller drivers in general on NVMe in particular, are there any software-based analyzers that can provide real assistance?



The screenshot displays the busTRACE 10.0 interface. The top pane shows a list of SCSI compliance inquiries. The bottom pane shows a detailed CDB View 1 for an Inquiry - Device Identification command. The CDB view highlights a defect in the IEEE COMPANY_ID field (E4D25Ch) with an error message: "ERROR: Firmware bug detected. The value given for this IEEE OUI is not in our database and is likely due to a firmware bug. Please refer to the IEEE OUI web site to validate this field."

Originator	Start Time	Exe Time	Command 1	Commr
15	00:00:02.264	18 usec	Inquiry - Supported VPD Pages	12 01 C
16	00:00:02.266	13 usec	Inquiry - Supported VPD Pages	12 01 C
17	00:00:02.266	12 usec	Inquiry - Supported VPD Pages	12 01 C
18	00:00:02.266	8 usec	Inquiry - Unit Serial Number	12 01 E
19	00:00:02.266	7 usec	Inquiry - Unit Serial Number	12 01 E
20	00:00:02.266	7 usec	Inquiry - Device Identification	12 01 E
21	00:00:02.266	5 usec	Inquiry - Device Identification	12 01 E
22	00:00:02.266	6 usec	Inquiry - Extended Inquiry Data	12 01 E

Field	Value
15	(LSB) 54h
16 (MSB)	E4h
17	02h
18	(LSB) 5Ch
19 (MSB)	34h
20	IEEE COMPANY_ID = E4D25Ch
21	IEEE COMPANY_ID = E4D25Ch
22	ERROR: Firmware bug detected. The value given for this IEEE OUI is not in our database and is likely due to a firmware bug. Please refer to the IEEE OUI web site to validate this field.
23	(LSB) 08h
24	Protocol Identifier = 00h (Reserved) Code Set = 03h (UTF-8 codes) 03h
25	PIV = 0h reserved = 0b e addressed logical unit Designator Type = 08h (SCSI name string) 08h

Figure 3 – Failing Inquiry Validation (click to enlarge)

With a Windows storage controller driver, software analyzers can actually play quite an interesting role. At their device edge, controller drivers communicate with their hardware using a protocol defined by the hardware. With NVMe, the controller driver sends NVMe commands over PCIe. Clearly a PCIe analyzer with NVMe decoding capabilities is the best choice for analyzing this portion of the driver.

However, at their host edge, controller drivers in Windows communicate using SCSI and SCSI Request Blocks. It is the job of the controller driver to translate between their native device edge and the host SCSI interface. A software analyzer in the host can be used to trace the SCSI edge of the driver interface, which can be used to analyze and validate the translations being performed by the driver.

Of course, the utility of the software analyzer hinges much upon the quality of the software. While working on our NVMe controller driver, we've found the busTRACE 10.0 from busTRACE Technologies (<http://bustrace.com>) does [\(CONTINUED ON PAGE 15\)](#)

Hardware vs. Software Bus... (Cont.)

(CONTINUED FROM PAGE 14)

an excellent job of providing insight into the host side of our translations. Not only does it actually show us a trace of the operations being performed, but it adds a significant level of validation to each of the responses returned by the driver.

A good, real example of this from our development experience comes from running the WHCK SCSI Compliance Test 2.0 as part of certifying our solution. Our driver passed all of the SCSI Inquiry related tests with no errors or warnings, but received a validation failure in the associated busTRACE capture. In **Figure 3** (Page 14), you can see that busTRACE does not believe that the IEEE Company ID returned in the Device Identification Page to be correct.

This was surprising, as we simply take the data from the NVMe Identify structure and place it into the corresponding SCSI Inquiry structure. Looking at the data returned from the device and the failing data from busTRACE, we realized the error: SCSI devices returned this data in the reverse order and thus the host is expecting the bytes to be reversed. A quick code change and we passed both the WHCK SCSI Compliance Test **and** the additional busTRACE validations (**Figure 4**).

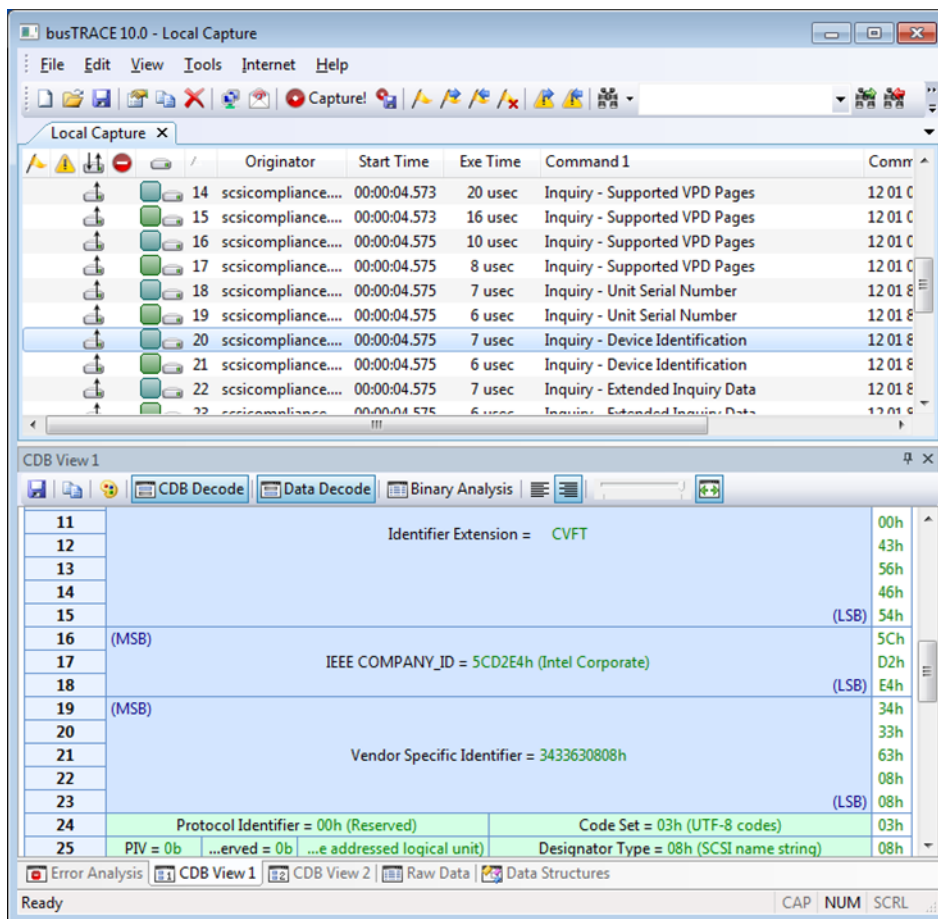


Figure 4– Fixed! (click to enlarge)

(CONTINUED ON PAGE 16)

KERNEL DEBUGGING & CRASH ANALYSIS SEMINAR

I Tried !analyze-v...Now What?

You've seen our articles where we delve into analyses of various crash dumps or system hangs to determine root cause. Want to learn the tools and techniques yourself? Consider attendance at OSR's [Kernel Debugging & Crash Analysis](#) seminar.

Next presentation:

Palo Alto, CA
6-10 April

For more information, visit www.osr.com/seminars/kernel-debugging/, or contact an OSR seminar coordinator at seminars@osr.com

Hardware vs. Software Bus... (Cont.)

(CONTINUED FROM PAGE 15)

Aside from just the translation help, busTRACE also does a nice job of enforcing some other best practices, such as ensuring that the host is informed that no data was transferred when an error occurs. It also provides insight into all the other, non-SCSI based things going on in the host, such as PnP and Device Controls IRPs. busTRACE also does a nice job of value adding with features above and beyond just logging, such as providing I/O stress and data validation utilities. Some of the additional features can be seen in the busTRACE Start Menu, as seen in **Figure 5**.

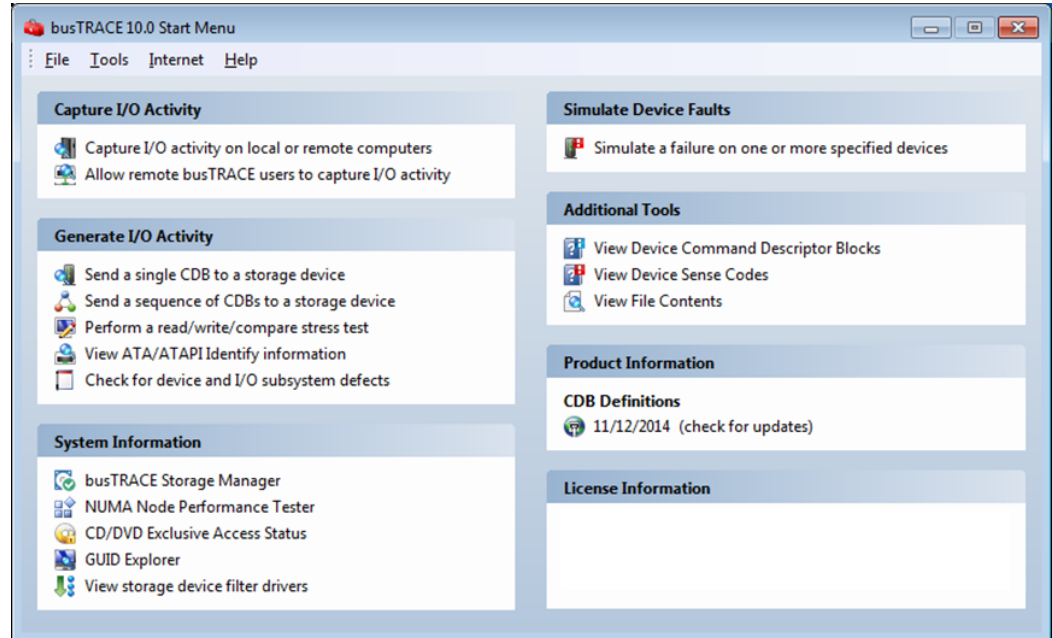


Figure 5 – Start Menu (click to enlarge)

Hardware, Software, or Both?

As you can probably tell, we're pretty high on all the analyzers that we've discussed, both hardware and software. This is definitely not where our opinion on the matter started, we had always stood firm on the fact that hardware analyzers were the only ones worth having. However, after seeing some of what busTRACE had to offer we have definitely seen the light a bit. Now we wouldn't start any project without having **both** a hardware and software analyzer at the ready for whatever problems might come.



Follow us!



WINDOWS FILE SYSTEM DEVELOPMENT

This seminar filled a very specific need for me, and was a perfect fit. The specific material, plus the instructor's expertise and engaging teaching style made it an extremely worthwhile experience for me. I was able to take this course without any other pre-req seminars from OSR, based on years of experience with OS internals and file systems on platforms other than Windows. Now that I am working with Windows file systems, I needed to learn as much as I could, and this was the right choice. I have a stack of books, but a classroom experience was much more direct and efficient way to learn the material. I have already felt the value of this seminar in my day-to-day work.

- Feedback from an attendee of THIS seminar

Next Presentation:

Boston/Waltham, MA
12-15 May

Our Journey to Writing... (Cont.)

[\(CONTINUED FROM PAGE 11\)](#)

What's been most gratifying is that our careful design resulted in the highest performance and lowest CPU utilization of any NVMe driver we've tested to date in our lab. We handily beat both the inbox Windows 8.1 driver and the OFA Windows driver in our tests.

So, what's supported? At Beta release, the driver will support all the basic functionality in NVMe 1.1b and the NVMe SCSI Translation Reference. The driver will pass all Windows Hardware Certification Kit tests for the driver (that is, the tests that test functionality of the driver itself and not the actual NVMe hardware device being tested), including SCSI compatibility tests. The driver also supports MSI-X, and an almost entirely lockless model using a dedicated pair of submission and completion queues per processor.

The distribution contains both built (and signed) executable images, as well as full source code. The driver builds using the latest Windows 8.1 WDK.

By the Final V1.0 release of our driver, we currently are committed to support at least the following:

- Windows 7 (and Server 2008 R2) through Windows 10 Technology Preview
- All mandatory features specified in NVMe V1.1e
- All mandatory features in the NVMe SCSI Translation Reference
- Comprehensive NVMe pass-through capability, including namespace formatting via pass through.
- MSI and MSI-X. Where MSI-X is available, support for dedicated submission and completion queue pairs per CPU.
- Windows Hardware Certification Kit tests (relevant to drivers, not to an NVMe hardware device) all pass, including SCSI compatibility tests.

We're taking a unique approach to development between the Beta and Final releases: Clients who are members of the Beta program will have the opportunity to influence the standard NVMe features that will be included in the 1.0 release, over and above those listed. We're open to considering the addition of features described in any currently published version of the NVMe specification (even NVMe V1.2).

We're also willing to customize the driver for a particular client's needs. These customization can include specific interfaces, specialized hardware handling, or even inclusion of advanced NVMe features that we would not otherwise have time to include in our standard 1.0 release.

NVMe hardware vendors might choose a customized driver to ensure specific, advanced, features are supported that are advantageous to their device. Similarly, they may choose to customize algorithms, such as by specifying algorithms for read/write optimizations.

Data centers deploying NVMe might want us to customize the driver to provide features that are important in their environment (such as namespace hot-plug), or to ensure that diverse vendor hardware can be used via a uniform, standardized, interface.

It's been a long and interesting journey for us, as we decided to undertake this project and started to make it happen. We're excited to finally be working with clients as part of our beta program, and are looking forward to releasing our work to the world.

For more on OSR's NVMe Solution Kit and the Beta program, visit www.osr.com/nvme-driver/ or contact our sales team (sales@osr.com) to describe your specific NVMe needs or to get your questions answered.



Follow us!



Win7 vs. Win8 StorPort MiniPorts... (Cont.)

[\(CONTINUED FROM PAGE 9\)](#)

Due to the extensibility and overall fanciness of the new structure, retrieving fields of the structure is often not as simple as dereferencing a field by name. Several macros are provided for parsing these structures and retrieving individual fields. For example, **SrbGetScsiData** retrieves SCSI related information from a **STORAGE_REQUEST_BLOCK** that describes a SCSI operation (CDB, Sense Buffer, etc.). As an aside, the new structure also makes it difficult to extract this information while debugging. See *Sidebar – Dumping SRBs in WinDbg: How Hard Can It Be?, P. 19*.

As you can imagine, the code to handle both the old and new structures in a single driver quickly becomes spaghetti. For **every single field** that the driver wants to retrieve, the driver must first determine which type of structure it is and then either directly dereference the field or call a macro to do it.

Thankfully, a header file is provided that does most of the work for you: *srbhelper.h*. This header contains many inline functions to retrieve fields from either type of structure - just call with the SRB pointer and the inline function does the rest. For example, check out the implementation of **SrbGetCdbLength**:

```
FORCEINLINE UCHAR
SrbGetCdbLength(
    _In_ PVOID Srb
)
{
    PSTORAGE_REQUEST_BLOCK srb = (PSTORAGE_REQUEST_BLOCK)Srb;
    UCHAR CdbLength = 0;

    if (srb->Function == SRB_FUNCTION_STORAGE_REQUEST_BLOCK)
    {
        SrbGetScsiData(srb,
            &CdbLength,    // CdbLength8
            NULL,        // CdbLength32
            NULL,        // ScsiStatus
            NULL,        // SenseInfoBuffer
            NULL);       // SenseInfoBufferLength
    }
    else
    {
        CdbLength = ((PSCSI_REQUEST_BLOCK)srb)->CdbLength;
    }
    return CdbLength;
}
```

Pretty neat, right? Now you just need to discipline yourself to actually **call** this function every time you need the length of the CDB and you're done! Well, not quite...These inline functions only exist when building for Windows 8 and later, so you can't call them for your Windows 7 builds! The pesky backwards compatibility story kills it every time.

In the interest of keeping our code clean, we chose to layer our own wrappers over the wrappers. Given that the **STORAGE_REQUEST_BLOCK** structure doesn't exist on Windows 7, we can assume that when built for Windows 7 we're dealing with **SCSI_REQUEST_BLOCKS** and can let the *srbhelper.h* functions figure it out on Windows 8 and later. For example, see our implementation of **OsrSrbGetCdbLength**:

```
FORCEINLINE
UCHAR
OsrSrbGetCdbLength(
    _In_ PVOID Srb
) {
    #if (NTDDI_VERSION >= NTDDI_WIN8)
        return SrbGetCdbLength(Srb);
    #else
        return ((PSCSI_REQUEST_BLOCK)Srb)->CdbLength;
    #endif
}
```

[\(CONTINUED ON PAGE 19\)](#)

Win7 vs. Win8 StorPort MiniPorts... (Cont.)

[\(CONTINUED FROM PAGE 18\)](#)

osrhelper.h turned into a very long module that was not very fun to write. However, our mainline paths are now clean and free from distinguishing between the two different request formats. To enforce the usage of these functions, the drivers immediately cast all SRB pointers to PVOID and treat them as opaque throughout the driver.

On the Right Track?

We are clearly at a pivotal moment for the storage driver model in Windows. Many changes are being made, including ones that are not simply reactionary but are attempting to plan for the future. This is creating some pain now for those of us still worried about the past, but hopefully we're trading that for some flexibility in the future.



Follow us!



Dumping SRBs in WinDbg: How Hard Can It Be?

Given that StorPort abstracts the operating system from the driver writer, when something goes wrong it's often quite difficult to get a clear picture of exactly what's happened. This is precisely why other abstractions provide a set of custom debugger extensions for use by driver writers. For example, WDF provides the excellent WDFKD.dll with commands for displaying structures and logs. NDIS does the same with NDISKD.dll, which is the absolute gold standard for debugger extensions. Seriously, even if you're never going to write an NDIS based driver you should try playing with this extension. It will make you wish you had a network driver problem to debug.

StorPort, on the other hand...Not so much. There are in fact **no** debugger commands for interpreting StorPort state or structures. Not even a *!srb* command to interpret a **SCSI_REQUEST_BLOCK** or a **STORAGE_REQUEST_BLOCK**. This is particularly painful with respect to the new **STORAGE_REQUEST_BLOCK**, good luck trying to manually walk that structure in the debugger and extract something useful.

What's frustrating is that this need isn't anything new or surprising. Even SCSIPort had two extension DLLs, SCSIKD and MinipKD, that, while not as useful when compared to modern standards, at least gave you some support. There even was a *!srb*. Unfortunately that command appears to have rotted, so all you get when running it now is this:

```
0: kd> !minipkd.srb fffffe001`73af95f8
Could not read SRB @ fffffe00173af95f8
```

Hopefully in time we'll see some new or updated commands in the standard tools. In the meantime, we're stuck manually interpreting these structures or writing our own debugger commands.

Introduction to NVMe... (Cont.)

[\(CONTINUED FROM PAGE 7\)](#)

But, wait, you say. If an MSI-X will be generated for each completion operation, won't that create a lot of overhead? Well, yes, it will... **IF** the controller generates one interrupt for each Completion Queue entry it processes. But the NVMe specification includes native support for interrupt coalescing. Using this feature, the driver can programmatically establish interrupt count and timing thresholds. The result? Interrupts are automatically combined by the NVMe device, within driver defined limits.

Figure 2 (page 19) illustrates a host that's communicating with a controller via a pair of Admin queues that is used for all controller management operations, and Submission Queues that are setup for I/O processing per core. (This diagram has appeared in numerous NVMe presentations without citation and its origin is unknown. It is thus assumed to be in the public domain).

Another interesting option is that NVMe allows the driver to create multiple Submission Queues per core and establish different priorities among those queues. While Submission Queues are ordinarily serviced in a round robin manner, NVMe optionally supports a weighted round robin scheme that allows certain queues to be serviced more frequently than others. **Figure 2** shows two Submission Queues configured for Core 1. Presumably, these queues would differ in terms of their priority. Of course, having a single additional Submission Queue on one specific core as shown in **Figure 2** probably isn't very realistic configuration, but it illustrates the point. And it sure looks great in the diagram.

The format of the queues themselves is designed for maximum speed and efficiency. Submission Queues comprise fixed size, 64 byte, entries. Each entry describes one operation to be performed by the device such as a Read, Write, or Flush. The 64 byte command queue entry itself has enough information in it to describe a Read or Write operation up to 8K in length. Transfers up to 2MB can be accommodated by setting-up a single, additional 4K table entry. Completion Queues are similarly efficient. Completion Queue entries are also fixed size, but only 16 bytes each. Each Completion Queue entry includes completion status information and has a reference to the Submission Queue entry for the operation that was completed. Thus, it's relatively straight-forward to correlate a given Completion Queue entry with its original command.

Why It's Cool – The Commands Themselves

If you've ever worked with SCSI, or even SATA, you know what a morass those command sets have become. There are a zillion different commands, many of them required, many of them with different options. NVMe specifically sought to simplify this situation. In fact, in NVMe there are only three mandatory I/O commands: Read, Write, and Flush. That's it! There are an additional nine optional I/O commands, including "Vendor Specific." All in all, that's a pretty lean command set for performing I/O operations.

[\(CONTINUED ON PAGE 21\)](#)

OSR CUSTOM SOFTWARE DEVELOPMENT

I Dunno...These Other Guys are Cheaper...Why Don't We Use Them?

Why? We'll tell you why. Because you can't afford to hire an inexperienced consultant or contract programming house, that's why. The money you think you'll save in hiring inexpensive help by-the-hour will disappear once you realize this trial and error method of development has turned your time and materials project into a lengthy "mopping up" exercise...long after your "inexpensive" programming team is gone. Seriously, just a short time ago, we heard from a Turkish entity looking for help to implement a solution that a team it previously hired in Asia spent *two years* on trying to get right. Who knows how much money they spent—losing two years in market opportunity and still ending up without a solution is just plain lousy.

You deserve (and should demand) definitive expertise. You shouldn't pay for inexperienced devs to attempt to develop your solution. What you need is fixed-price solutions with guaranteed results. Contact the OSR Sales team at sales@osr.com to discuss your next project.

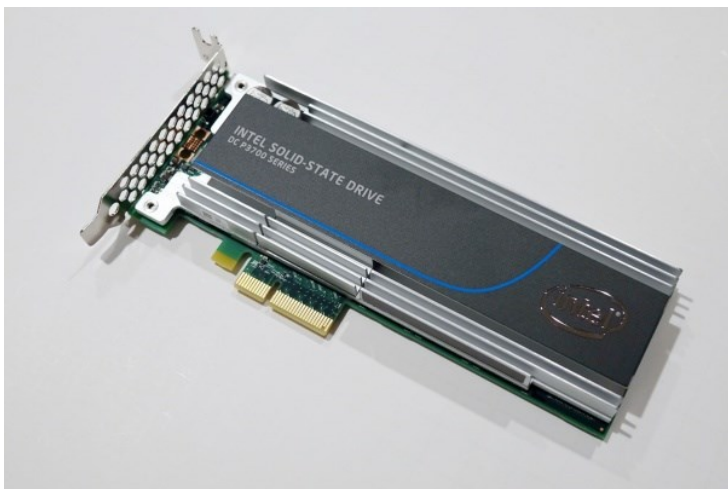
Introduction to NVMe... (Cont.)

[\(CONTINUED FROM PAGE 20\)](#)

In terms of admin commands, which are used for configuration and management, there are only ten required and six optional commands (again, including "Vendor Specific"). The required admin commands include Create I/O Submission Queue, Delete I/O Submission Queue, Create I/O Completion Queue, and Delete I/O Completion Queue. There are also Identify, Abort, Get Log Page, Set Features, Get Features, and Async Event Request. That's the whole list.

Needless to say, a small well-defined command set makes command processing pretty streamlined in the driver.

But there's also a lot of power that can be optionally implemented using these few commands. For example, there are provisions for end-to-end data protection (what SCSI calls DIF and DIX). This feature allows (among other options) a CRC value to be stored with each logical block that's written and compared to a computed value when the block is read.



Intel's DC P3700 PCIe SSD Controller

A more forward looking feature is the Dataset Management (DSM) support that can optionally accompany Read and Write commands. DSM allows the data being written to be tagged with various attributes, including expected access frequency (one time write, frequent writes but infrequent reads, infrequent writes but frequent reads, etc.). Other information that can be specified includes whether the data is incompressible, or whether a read or write is expected to be part of a larger, sequential operation. While it might be a while before Windows (and even NVMe drives) supports all these features, it's awesome to know that they're there for future use.

[\(CONTINUED ON PAGE 22\)](#)

ARE YOU PASSIONATE ABOUT WINDOWS INTERNALS, DRIVER DEVELOPMENT AND KERNEL DEBUGGING?

OSR is Hiring!

OSR is hiring one or more [Software Development Engineers](#) to implement, test and debug Windows kernel mode software.

We're looking for a very talented individual (or two) to grow into valued contributors to the OSR engineering team, our clients, and the community.

Do you need to be a Windows internals guru? No—we'll help you with that—but you DO have to LOVE operating system internals. It's what we live and breathe here at OSR.

We've found such folks to be a rare breed, so if this is YOU or someone you know, get in touch with us and tell us why we can't afford NOT to hire you. See www.osr.com/careers for more detail.

Introduction to NVMe... (Cont.)

[\(CONTINUED FROM PAGE 21\)](#)

And It's Here Now

You might not realize it, but Windows began shipping with a driver that supports NVMe devices with Windows 8.1. NVMe devices themselves are also beginning to become available. Intel's DC P3700 series drive is the first NVMe device to ship in volume to end users, and it's widely available for sale.

But why should you care? Because NVMe isn't just **designed** to be fast, it really **is** FAST, that's why. According to our own testing (which confirms [what Tom's Hardware found](#)), the Intel P3700 NVMe drives positively **fly**: Doing 4K random reads, we got 460,000 I/O Operations per Second (IOPs). Max sequential read throughput is about 2.8Gbps. These aren't theoretical numbers, they're real measurements. Now, granted that is only for read operations. Write operations are significantly slower, and depend a lot on the state of the drive when it's being written. But still, there's no denying NVMe is **fast**.

Considering the performance, the Intel drives aren't impossibly expensive, either. You can get the 400GB version on Amazon for about \$1200. Of course, as more NVMe drives come to market, we can expect the price to come way down. And, remember, this **is** an SSD device.



I was just telling Dan that I need at least two of these: One 800GB drive as my boot volume and another one for my paging file. Dan's response? Well, I'm still waiting for him to answer my email.

Follow us!



THE NT INSIDER Hey...Get Your Own!

If a colleague three cubes down with less than stellar hygiene forwarded this on to you and you fear that this act of kindness may be interpreted as the start of a budding relationship...

Just [send a blank email to join-ntinsider@lists.osr.com](mailto:send_a_blank_email_to_join_ntinsider@lists.osr.com) — You'll get an email whenever we release a new issue of The NT Insider.

OSR USB FX2 LEARNING KIT

Don't forget, the popular OSR USB FX2 Learning Kit is available in the Store at www.osronline.com.

The board design is based on the well-known Cypress Semiconductor USB FX2 chipset and is ideal for learning how to write Windows drivers in general (and USB specifically of course!). Even better, grab the sample WDF driver for this board, available in the Windows Driver Kit.

OSR'S CORPORATE, ON-SITE TRAINING

Save Money, Travel Hassles; Gain Customized Expert Instruction

We can:

- Prepare and present a one-off, private, on-site seminar for your team to address a specific area of deficiency or to prepare them for an upcoming project.
- Design and deliver a series of offerings with a roadmap catered to a new group of recent hires or within an existing group.
- Work with your internal training organization/HR department to offer monthly or quarterly seminars to your division or engineering departments company-wide.

To take advantage of our expertise in Windows internals, and in instructional design, contact an OSR seminar consultant at +1.603.595.6500 or by email at seminars@osr.com

Peter Pontificates... (Cont.)

[\(CONTINUED FROM PAGE 5\)](#)

(24 degrees with scatter clouds and a 51% probability of precipitation)... because I, once again, had Internet coverage!

After several hours of cavorting on the Information Super Highway, I decided I had been absent from home long enough. I slipped and slid my way to my car, and started to head home.

On my way home from the office, my wife called: "I just wanted you to know that the power's back on! And the landline works again. Cooper says hello. We went for a long walk in the snow, it was really beautiful. Oh, and I think the Internet is probably back too." You **think!**? You called to tell me the power's back and my Bernese mountain dog says hello, but you only **think** the Internet is back! How can you not **know!**? That would be the first thing that I checked!! My wife hung up the phone.

When I got home, the Internet was indeed back. And life was back to normal. Any fact I wanted to know was, once again, available at the click of a button. And I was thankful. I guess it was my own little Thanksgiving Day.

Funny thing though: I couldn't resist programming the TiVo to start recording all the new episodes of Tattoo Nightmares.



Peter Pontificates is a regular column by OSR Consulting Partner, Peter Viscarola. Peter doesn't care if you agree or disagree with him, but there's always the chance that your comments or rebuttal could find its way into a future issue. Send your own comments, rants or distortions of fact to: PeterPont@osr.com.

Follow us!



ALREADY KNOW WDF? BOOST YOUR KNOWLEDGE

Read What Our Students Have to Say About
Writing WDF Drivers: Advanced Implementation Techniques

It was great how the class focused on real problems and applications. I learned things that are applicable to my company's drivers that I never would have been able to piece together with just WDK documentation.

A very dense and invaluable way for getting introduced to advanced windows driver development. A must take class for anyone designing solutions!

The seminar is what I was looking for in a training class. Covering in depth topics about WDF driver development and verify that I am writing WDF drivers the right way. The experience of the instructors was very valuable.

Next Presentation:

Palo Alto, CA 10-13 March

OSR Seminar Schedule

Seminar	Dates	Location
WDF Drivers: Core Concepts	12-16 January	Palo Alto, CA
Internals & Software Drivers	16-20 February	Seattle, WA
WDF Drivers: Advanced Implementation Techniques	10-13 March	Palo Alto, CA
Kernel Debugging & Crash Analysis	6-10 April	Palo Alto, CA
Developing File Systems	12-15 May	Boston/Waltham, MA

OSR Seminars

We “Practice What We Teach” For a Reason

When we say “we practice what we teach”, this mantra directly translates into the value we bring to our seminars. But don’t take our word for it...below are some results from recent surveys of attendees of OSR seminars:

- I've learned everything that I wanted to learn and quite a bit that I did not know I needed.
- I think Scott nicely catered to a diverse audience, some of whom had only taken intro to OS and some who already had experience developing drivers.
- Scott was fantastic. He was very helpful at trying to meet each student’s needs. **I will highly recommend him and OSR to my associates.**
- “Peter’s **style of teaching is excellent** with the kind of humor and use of objects to give you a visual representation of how things work that was simply amazing.
- “I was very nervous coming in to the seminar as I wasn’t sure I had enough hands on experience and background knowledge on Windows internals. The class put my mind at ease and **I was able to quickly grasp and understand the concepts.**”
- “I was pleased with the experience. As someone new to driver development, it gave me a better understanding of the framework and **made me a lot more comfortable working with our driver code.**”
- “This is a great seminar. I will say this **is the best seminar** so far I have attended. **I learned a lot** in the class and the lab.”
- “Peter was kind enough to answer technical questions other than topics listed. Both **Scott and Peter were excellent** in teaching this seminar.”
- “A very dense and invaluable way for getting introduced to advanced windows driver development. A **must take** class for anyone designing solutions.”
- “Scott is a gifted teacher. He truly made the concepts **easy to understand and learn**”.

Private Training

A private, on-site seminar format allows you to:

- **Get project specific questions answered.** OSR instructors have the expertise to help your group solve your toughest roadblocks.
- **Customize your seminar.** We know Windows drivers and file systems; take advantage of it. Customize your seminar to fit your group's specific needs.
- **Focus on specific topics.** Spend extra time on topics you really need and less time on topics you already know.
- **Provide an ideal experience.** For groups working on a project or looking to increase their knowledge of a particular topic, OSR's customized on-site seminars are ideal.
- **Save money.** The quote you receive from OSR includes everything you need. There are never additional charges for materials, shipping, or instructor travel.
- **Save more money.** Bringing OSR on-site to teach a seminar costs much less than sending several people to a public class. And you're not paying for your valuable developers to travel.
- **Save time.** Less time out of the office for developers is a good thing.
- **Save hassles.** If you don't have space or lab equipment available, no worries. An OSR seminar consultant can help make arrangements for you.

