ISSUE VOLUME
02/21
JULY– AUG
2015

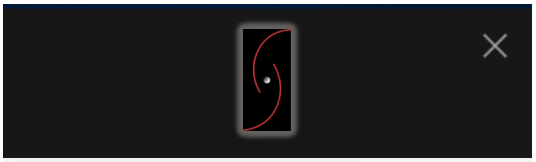
The NT Insider

A publication of OSR Open Systems Resources, Inc.

Hey Cortana.
What's Peter Pontificating about now?

what's peter pan pacific about now

Hey OSR.
What's Peter Pontificating about now?



windows 10 is ruining my life

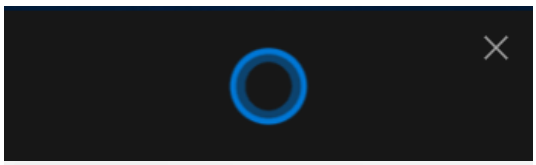
Hey Cortana.
What are the new rules for driver signing?

what are the new rules for java sunny

Hey OSR.
What are the new rules for driver signing?

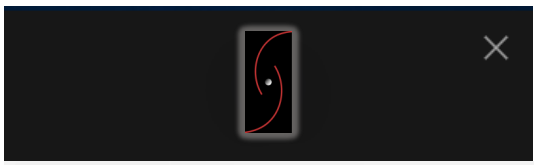
driver signing and windows 10

Hey Cortana.
How do I set up network debugging?



how do I set up a network toboggans

Hey OSR.
How do I set up network debugging?

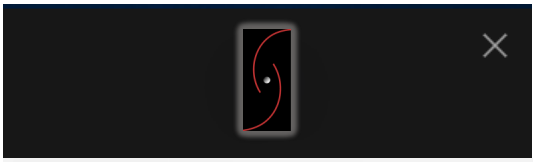


kdnet debugging

Hey Cortana.
What's new with the WDK for Windows 10?

what's new with wtkk for this tent

Hey OSR.
What's new with the WDK for Windows 10?



windows 10 wdk and visual studio 2015

Hey Cortana.
Can you explain valid data length to me?

can you explain validator links to me?

Hey OSR.
Can you explain valid data length to me?

more on maintaining valid data length

OSR Announces Monthly KMDF and Debug Seminars

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

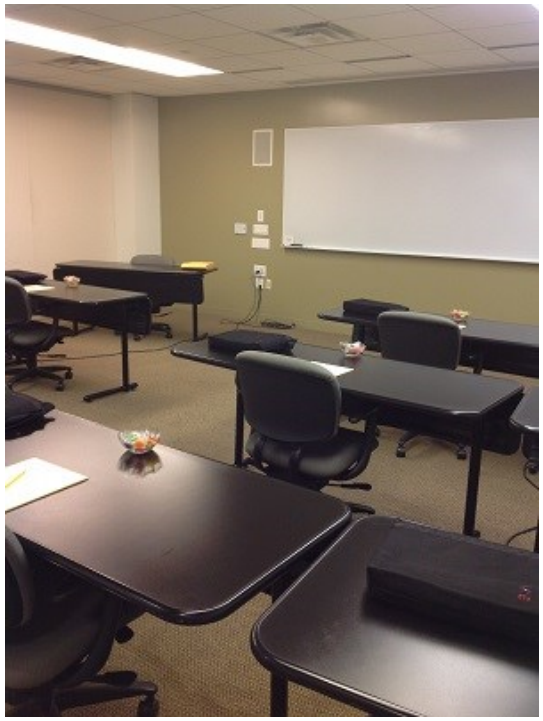
Contributing Editors
Scott J. Noone
OSR Associate Staff

Send Stuff To Us:
NtInsider@osr.com

Single Issue Price: $15.00

OSR's New Training Venue
Now with Flexibility to Host Monthly Seminars

While OSR will continue to offer it's seminar presentations across the US and internationally, we now have the flexibility to schedule seminars more often at our brand new training space adjacent to OSR headquarters.



What this means for you is:

- Flexibility in attendance at an OSR seminar that is timely—no more waiting several months for a seminar when you need the training now.
- A low instructor-student ratio—allowing ample opportunity for assistance in labs and securing answers to your questions.
- A comfortable learning environment that we control—we're not subject to the policies of some large hotel/conference center. We make our own rules to serve the needs of our attendees best.
- And...with the OSR engineering team just down the hall, you never know who will swing by to answer questions, drill down into an advanced topic, or offer an impromptu guest presentation.

Want to know more? Contact one of our helpful seminar consultants to find out about the next, convenient OSR seminar for you: seminars@osr.com

OSRHINTS: 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

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.

Windows 10 WDK + Visual Studio 2015 Issues
More to come, but some quick feedback for those of you early adopters...
https://www.osr.com/blog/2015/08/12/windows-10-wdk-visual-studio-2015-issues/

Questions and Answers: Windows 10 Driver Signing
OSR's Peter Viscarola interviews Microsoft PM James Murray on some of the new and important changes to driver signing on Windows 10.
http://www.osr.com/blog/2015/07/24/questions-answers-windows-10-driver-signing/

Newbie Corner: Breaking Windows
Failures, repeated failures, are finger posts on the road to achievement. One fails forward toward success—C.S. Lewis
http://www.osr.com/blog/2015/07/09/newbie-corner-breaking-windows/

Newbie Corner: The Newbie Gets a Taste of Architecture
...and learns the value of instruction from an expert as opposed to just reading through documentation.
http://www.osr.com/blog/2015/06/22/newbie-corner-newbie-gets-taste-architecture/

Newbie Corner: There's a type for that! The Unexpected World of Windows Kernel Types
OSR's Chris Barr tackles newbie driver dev topics in this first in a series post.
http://www.osr.com/blog/2015/05/27/newbie-corner-theres-type-unexpected-world-windows-kernel-types/

Binary Literals in VS2015—I Can't Wait!
Our take on new features coming in Visual Studio 2015...
http://www.osr.com/blog/2015/05/22/binary-literals-vs2015-cant-wait/

THE NT INSIDER - Hey...Get Your Own!

Just send a blank email to join-ntinsider@lists.osr.com — and you'll get an email whenever we release a new issue of The NT Insider.

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.

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

Peter Pontificates:
Windows 10 is Ruining My Life

Why did Microsoft have to make the free upgrade period to Windows 10 a year long? Why couldn't it have been, say, a day or two? Because, if the past month is any indication, I don't think I'm going to make it through an entire year of answering Windows 10 upgrade questions from dough-heads. Well-meaning dummies, mostly, even some goofs who also happen to be friends and neighbors. And relatives. But, you know, dough-heads nonetheless.

And my continued encounters with these unfortunates have put me in the uncomfortable position, dear reader, of having to ask for your assistance.

Am I the only one who's forced into these conversations in random locations? I'll be at the market / car dealer / dog groomer / Asian grocery / mail box / local gun shop / overpriced, yet shockingly mediocre, suburban restaurant / gas station… and I'll be accosted by somebody I either know well, or only know in passing. It doesn't matter. The questions always follow a predictable pattern:

"Peter! Just the man I've been wanting to see! How's the wife how are the dogs how 'bout them Red Sox do you think Brady cheated?" Even here in New England it's de rigueur to begin with the social pleasantries, even if you don't really give a shit and don't listen to the answers.

"Fine, good, they suck, probably," I reply.

"Nice, so… What do you think about this Win 10 thing," they continue.

At this point, there's no avoiding it. But I can perhaps forestall the inevitable for just a **bit** longer. So I innocently ask, "This Win10 Thing?"

"You're a computer guy, you travel to Microsoft all the time, shit you probably know Bill Gates, *you know all about this stuff*. Should I install Windows 10?"

Over the course of time, my replies have evolved. Initially, I took the approach of trying to be maximally and earnestly helpful. I made the mistake of trying to thoughtfully answer the questions.

"Well, it depends. What version of Windows are you running now?" I asked.

The answer was always some variant of "The one with Outlook" or "It's on my Dell" or "Not the one that requires a touch screen." So, scratching my head and looking around to see if there was anything that I might urgently claim to have to do, I would reply "Well, if it's working for you now, do you have any specific reason you **need** to upgrade to a new version of Windows?"

The answer to this question always, every single time, unfailingly got the answer "I don't know!" followed by a blank stare. Or a big smile. Or a worried look. This was most often followed by vague statements about the one year free upgrade period and having something in their task bar telling them to upgrade.

So, that was getting us both, you know, exactly nowhere, and usually lead to the suggestion, "If you have a few minutes, maybe you could drop by my place and take a look at my computer. *You know all about this stuff*. I just cooked up a new batch of cookies / meth / beer / organic weed killer / pasta."

Like there was something useful that I would glean from laying eyes on the long-suffering Dell lodged in their kitchen ("You ever think to clean the fucking FAN on this thing? No? Fan? F. A. N. Down here? At the bottom of the case? Never mind."). And like their cooked products would be an actual incentive.

(CONTINUED ON PAGE 5)

Peter Pontificates... (Cont.)

(CONTINUED FROM PAGE 4)

After a more than a few of these events, all with the same outcome, I figured it was time to get smart. Whenever I got The Question:

"You're a computer guy, you travel to Microsoft all the time, shit you probably know Bill Gates, *you know all about this stuff*. Should I install Windows 10?"

I gave them The Answer: "Absolutely. It's the best Windows yet. And it's free. So, you know, why not, right?"

I knew, I just **knew**... well, OK, I at my core at least hoped and prayed, that this would stop the conversation and I could go about my business unmolested.

But, to my surprise, while that one liner satisfied one or two of the less inquisitive types, it almost always resulted in one of the following additional responses. You can play along while you read this, and help me write my column, by assembling these responses yourself. Just choose one (or more!) from **Column A** and another one (or more!) from **Column B**:

Their reply: **My...** (item from Column A) **... said ...** (item from Column B)**:**

(CONTINUED ON PAGE 19)

Column A (pick one)	**Column B** (pick one)
Brother	Anybody who knows anything always waits for the first service pack to be released before installing a MSFT operating system.
Auntie	It's actually pretty good. Finally Microsoft got it right.
Spouse	Windows 10 will hijack your documents, ship them to the cloud, and after the one year "free" period force you to pay every month to access them.
Bernese Mountain Dog	He/she/it tried it but the upgrade failed and he/she/it couldn't use their computer anymore.
Daughter	It's better than the one that's only touch based and had those big blobs on the screen when you started your computer.
Boss	Can I eat it?! I'm hungry! You should take me for a walk!!
Gardener	It seems to be faster than Windows 7.
Dentist	It's full of spyware and Microsoft will spy on you and read all your documents and settings, put them in the cloud where the CIA can get them, and Obama the Socialist who was born in Kenya will share them with all the other radical Muslims.

Signed, Sealed, Delivered
Driver Signing and Windows 10

In case you've been busy worrying about writing code for existing projects and stuff, let me call your attention to some **big changes in the realm of driver signing that start with Windows 10**.

There's been an ongoing thread on NTDEV about this topic for more than a month now that has led to some very good information being collected and discussed by the Driver Development Community.

For all the details, you'll need to read my blog post from the end of July entitled Questions and Answers: Windows 10 Driver Signing. In that blog post, Microsoft program manager James Murray answers questions from the community on how the Windows 10 driver signing process will work.

Let me try to provide you a quick summary of what's changing, because you definitely will care: The big news is that, aside from a short-term exception, a Microsoft signature is required by Windows 10 to load kernel-mode drivers. To get that signature, you have to sign a submission using an Extended Validation (EV) Code Signing Certificate and upload your driver package to the Microsoft SysDev portal. **You do not need to run or pass any Microsoft certification, logo, or compatibility tests**. You just need to sign your driver package appropriately, agree to some conditions, and submit your package to Microsoft via SysDev for signature. This procedure is called “attestation signing” because when you upload you declare (that is “attest”) that you’ve tested the driver, will monitor sysdev for driver problems, and will fix any issues that are reported.

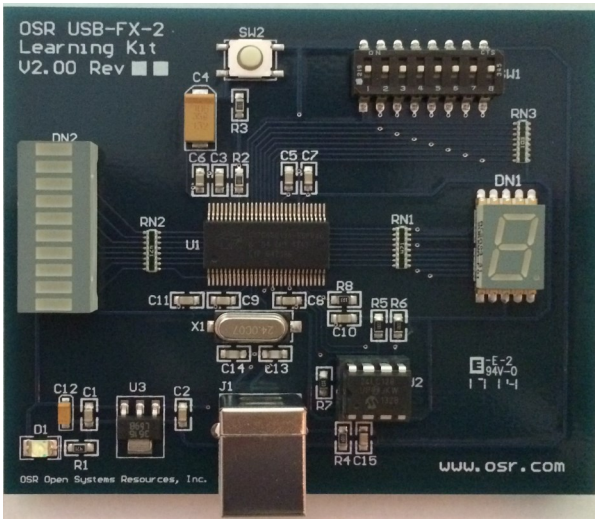
So, aside from the exception case, drivers for Windows 10 must have a Microsoft signature obtained through the SysDev portal. What's the exception case? The exception is that Windows 10 Client (not server) systems **will load** drivers that have been properly signed and cross-signed using the pre-Windows 10 KMCS procedure **if the certificate used to sign those drivers was issued prior to the release of Windows 10**. This gives driver developers some "breathing room" to adjust to the new policy.

One really important thing to note is, according to my discussion with Mr. Murray, **Windows Server vNext will only load drivers that have passed HLK testing** (formerly known as Certification testing or WHQL testing) and have obtained the appropriate signature from the SysDev portal.

Based on the information we learned from Mr. Murray's Q&A, the community devised the following seven rules about Windows 10 driver signing:

- A driver signed with the standard SHA-1 certificate issued prior to the 29th of July 2015 and correctly cross-signed according to the pre-Windows 10 KMCS procedures, will work on all platforms Vista through to 10. This is, however, subject to configuration an enterprise-defined code integrity policy that is part of Device Guard (available on Windows 10 Enterprise edition only). This enterprise-defined policy may be configured to require at least an attestation-signed driver.

(CONTINUED ON PAGE 7)



Driver Signing and Windows 10 (Cont.)

(CONTINUED FROM PAGE 6)

- A driver signed with a SHA-2 certificate (including an EV certificate) issued prior to the 29th of July, and cross-signed according to the pre-Windows 10 KMCS procedure, will work on Windows 8 and above, and will work on Windows 7 / Server 2008R2 if the patch issued through Windows Update earlier this year has been applied. It won't work on Windows Vista / Server 2008 though.

- A driver signed with any certificate issued after the 29th of July won't work on Windows 10, unless the driver is signed with a Microsoft signature available through the SysDev portal.

- A driver signed with any certificate that expires after the 29th of July will work on Windows 10, assuming that the signature was timestamped at the time of signing. If the signature was not timestamped, the driver will not work after the certificate expires.

- A portal-signed driver using attestation signing (which requires an EV certificate) will only work on Windows 10, unless **also** signed with an additional valid certificate and cross-signed according to the pre-Windows 10 KMCS procedure.

- A portal-signed driver that passes HLK tests will work on Windows 7 through Windows 10. Submitting a package for HLK certification requires the use of an EV certificate.

- Windows Server vNext will only load portal-signed drivers that have successfully passed the HLK tests.

Thanks to Community members (in order of the date of their first post to the thread) Tim Roberts, Daniel Terhell, Tom McDermott, David Cattley, Mike Fontana, Vikram Parthasarathy, Christiaan Ghijselinck, Alan Adams, Jeff Pages, Jan Bottorff, Gabe Jones, Don Burn, Chris Read, Anton Bassov, Gregory Dyess, James Bellinger, Matt McIntire, Chris Aseltine, Volodymyr M. Shcherbyna, Prokash Sinha, Maxim Shatskih, Phil Barila, Menachem Shapira, and a few others who participated via pseudonym. Your participation is what got these questions answered for the community. Bravo!

One more time: Don't rely on my restatements of the policy in this article. Please read the definitive Q&A on this topic entitled Questions and Answers: Windows 10 Driver Signing.

Follow us!

There's Gotta Be a Better Way
KDNET Debugging

I love using virtual machines for test systems. Particularly during ramp up at the start of a development project, where I'm constantly rebooting (and, ahem, corrupting…) systems with my first attempts at getting my code working. They also make great test specimens for my various experiments. Just today I was wondering if a particular bug found on Windows 8.1 is also present on Windows 7 and Windows 10. A few clicks later and I had my answer (yes, it is).

For all that I love debugging with VMs, there is one thing that I absolutely hate: KD debugging with virtual COM ports. Specifically in the case of VMware Workstation, which is what I use for my day to day virtualization needs, it's painfully slow. Even worse, the virtual COM port doesn't **quite** mimic the behavior that WinDbg expects from a real COM port. This leads to more than the occasional WinDbg hang or crash.

Thankfully, starting in Windows 8 there is a new way to solve this problem: KD debugging over Ethernet! Instead of piping a virtual COM port over a named pipe to the host, we can use a network connection between the guest and the host to shuttle our KD traffic. This feels much more responsive than the virtual COM port route, and overall provides a much more pleasant debugging experience (though sadly not as good as using a 1394b card on a physical machine).

Unfortunately, the following only works if you're debugging target VMs running Windows 8 and later. For targets running earlier versions of Windows, your best bet is still VirtualKD. Also note that the directions specified here are specific to VMware Workstation, YMMV with other virtualization solutions.

Step 1: Establish a network connection

The guest is going to need access to the host via Ethernet. You can configure this any way you like, but in our case we're going to debug over a private network with the host. *Figure 1* shows the connection settings that we used.

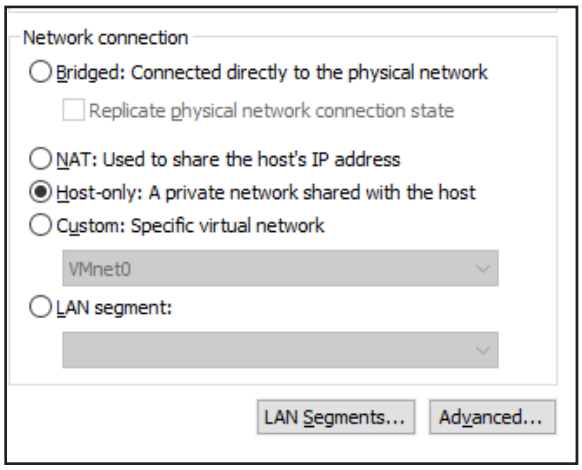


Figure 1— Host-only Network

Step 2: Locate host IP address

We'll need to specify the host IP address when configuring our debug session. By default, if you're using a VMware host only network the host IP is at address **.1** on the guest's subnet.

Using **ipconfig** (not shown), we determine that the guest IP is **192.168.154.129**. This means that our host IP address will be **192.168.154.1**.

Step 3: Use kdnet.exe to configure debug settings

The Windows 10 release of the Debugging Tools for Windows package contains **kdnet.exe**, which is an awesome utility that makes it dead simple to

(CONTINUED ON PAGE 9)

KDNET Debugging (Cont.)

(CONTINUED FROM PAGE 8)

configure network debugging. Included with this utility is the file **VerifiedNICList.xml**, which contains a list of all NICs known to work with KD network debugging.

Running **kdnet.exe** by itself to verifies that the virtualized NIC being used is on the verified list. In *Figure 2*, we can see that we are running with a supported NIC.

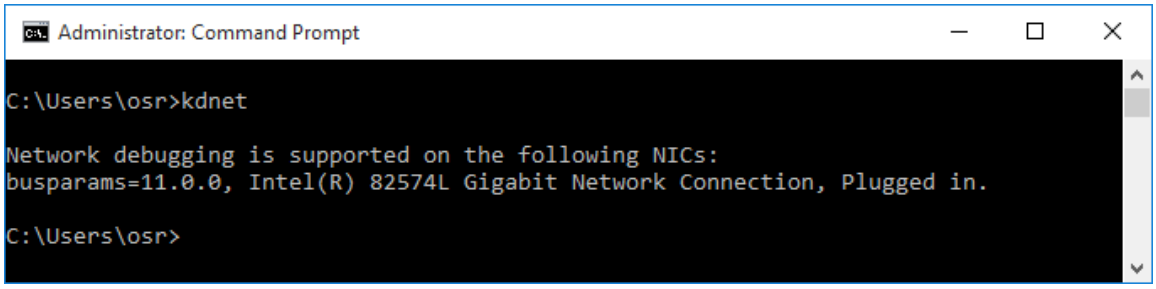


Figure 2—NIC is supported!

Now, all that's left is to enable debugging. **kdnet.exe** takes as arguments the IP address of the host and the port number to use for debugging. In this case, we'll specify **192.168.154.1** as our host and **50000** as our port (which happens to be the default). The result of this command will be an encryption key that we'll need to specify in our host connection settings, as seen in *Figure 3*.

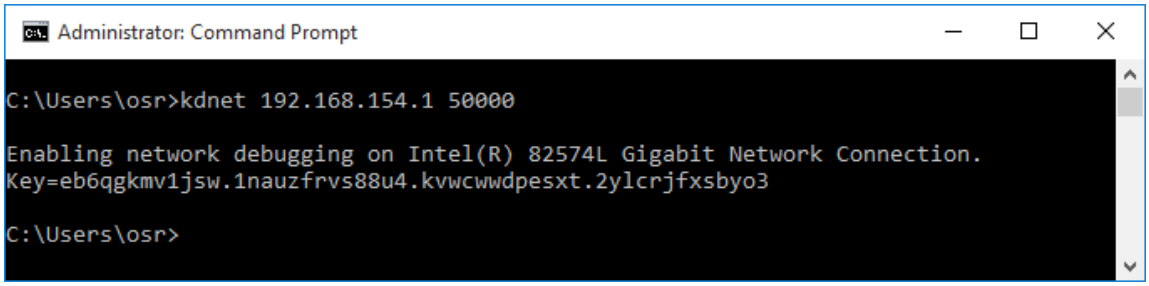


Figure 3—Success! And a secret key...

Now reboot the machine, you're done configuring the guest!

Step 4: Configure WinDbg

On your host machine, launch WinDbg and bring up the Kernel Debugging connection dialog (Ctrl+K). Under the NET tab, enter in the port number used as the debugging port (50000 in this case). Under Key, specify the key that was generated by the **kdnet.exe** command. You can see our settings in *Figure 4*.

The final thing to do is click OK and connect to the guest!

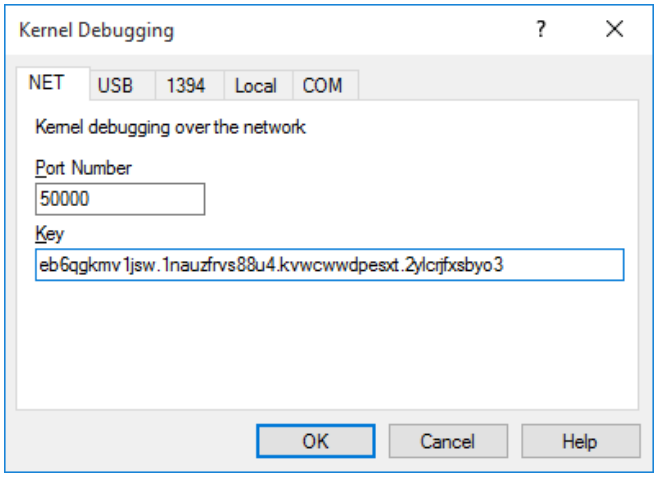


Figure 4—Specify the Port and Key

(CONTINUED ON PAGE 18)

The Same, Yet Different
Windows 10 WDK and Visual Studio 2015

A new version of Visual Studio brings along with it a new version of the Windows Driver Kit (WDK). And so it is for Windows 10. As Visual Studio (VS) and Windows itself continue to evolve, so does the WDK. With at least one notable exception (that we dearly hope is fixed soon) the new versions of VS and the WDK are changes for the better.

Probably the biggest change is that WDK is now fully integrated. You can use it to build drivers for any desktop/server OS from Windows 7 onward. The WDK also includes built-in support for targeting 32-bit and 64-bit ARM processors, as well as Windows Mobile platforms. So, there's a lot more in the kit than was there by default previously.

The Most Important Change

Let's get one absolutely critical point out of the way immediately: They changed the menu bar titles back to upper and lower case by default. As far as I'm concerned, VS 2015 has me right there. But, in case you find yourself pining away for those menu items THAT SHOUT AT YOU, you can change back to all uppercase easily. Just go to Tools… Options… Environment… General… and UNcheck "Apply title case styling to menu bar" and you'll be good to go.

OK? You happy? Now we can go on and deal with one or two other details.

Coexistence

You'll be happy to know that VS 2015 and the Windows 10 WDK and VS 2013 and the Windows 8.1 Update WDK can live happily side by side on your dev system. After installing the new versions of VS and the WDK, here at OSR we were able to build drivers successfully using either tool set. In fact, if we were careful and got our settings right, we were even able to switch back and forth between tool sets for projects that were initially created in VS 2013. In other words, we were able to open a (KMDF) driver project created in VS 2013 with the Win 8.1 Update WDK in VS 2015 with the Windows 10 WDK and build the Solution without any problems. Our Configurations (Win 7 Debug, for example) showed up and could be selected just as you would in VS 2013. Even more interesting, after doing this we were able to open that same solution in VS 2013 and build it successfully using the Windows 8.1 Update WDK.

The key to making this work is paying attention to the "Platform Toolset" setting in the Configuration Properties (General section) of each project (shown in *Figure 1*).

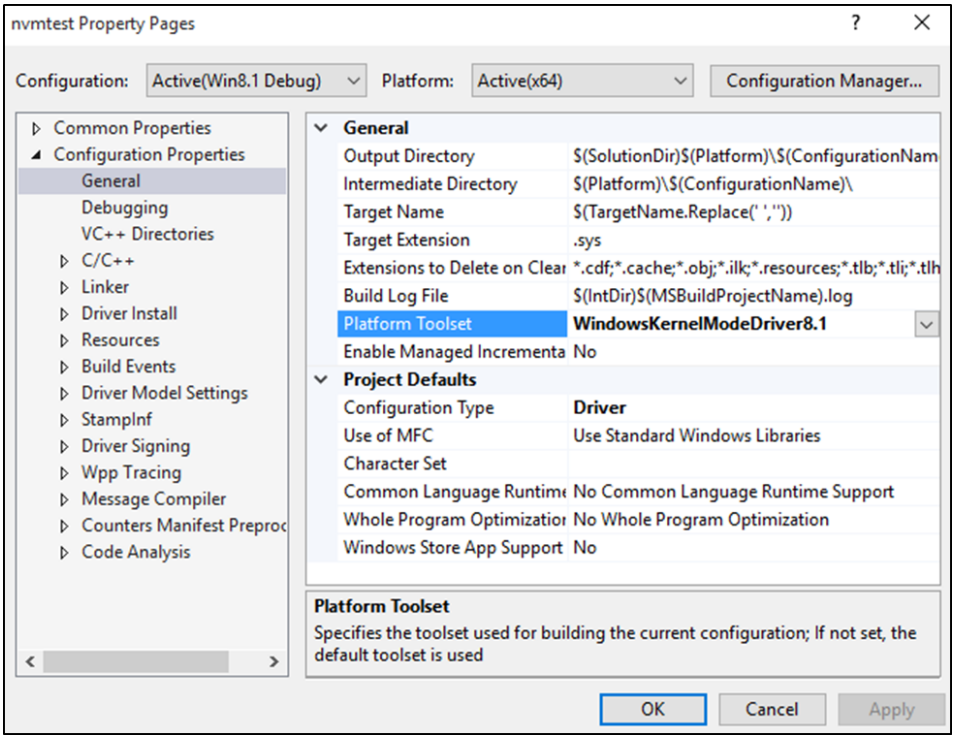


Figure 1—Using VS 2015…but building drivers with the VS 2013 tools

When you first open a Windows 8.1 WDK project with VS 2015, it asks you if you want to convert the projects to the new toolset. If you agree, the Platform Toolset will be changed to appropriate Win10 versions. In this case, when you build your driver you'll be using the new project configurations, the new compiler and the new linker.

If you **do not** let VS convert the projects to VS 2015, it will use the toolset that was previously defined for each project when you build your code. That means even if you choose to do your editing in VS 2015, you can choose which toolset VS will use for compiling and linking.

In summary, the power is in your hands to choose the tools you want to use to build your driver. And the IDE that you use for editing does not necessarily dictate which tools you'll use. Cool, right?

(CONTINUED ON PAGE 11)

Windows 10 WDK and VS 2015 (Cont.)

(CONTINUED FROM PAGE 10)

Where Are My Build Environments?!

The biggest change you'll see when working with the Windows 10 WDK (and the appropriate default Windows 10 toolset) is that the only Solution Configurations defined by default are "Release" and "Debug". No more "Win8.1 Debug", "Win8.1 Release", "Win8 Debug", "Win8 Release", "Win7 Debug", "Win7 Release". At first, we found this quite shocking. After all, we've been selecting the "build environment" to use for many years. To all of a sudden have these gone was, well, a bit unsettling. But after working with the new tools for a while, I **totally** get why they've made this change, and I've come to love it.

Why? Well, for one thing, most of us don't really **care** about the myriad target OS versions. Maybe we care about one or two specific OS versions that we're targeting, but do we really need to look at Release and Debug alternatives for every OS that's supported? It makes for a mess, especially when shipping drivers to customers. You either have to tell them which build environments are supported ("Remember… you only want to build the Windows 7 or the Windows 8.1 versions of the driver), or you have to **delete** build environments from your project. Which is always a nerve-wracking thing to do.

Another reason I've come to like the "roll your own" settings approach is, given that you'll almost certainly want to create your own Solution Configuration name(s) in any case, you now get to name them whatever you want. This is particularly nice for us, because we can name the configuration in a provided solution for a specific client, like we could create a "FredCorp Debug" Solution Configuration and a "FredCorp Release" Solution Configuration. Nice!

Finally, not having the long list of pre-made Solution Configurations just isn't practical anymore starting in Win10. Not only are OS versions back to Win 7 still supported (so there are four different Target OS Versions: Win7, Win8, Win8.1, and "Win10 or higher"), but there are now multiple Target Platforms supported. These are: Universal, Desktop, and Mobile.

(CONTINUED ON PAGE 14)

This Far and No Further
More on Maintaining Valid Data Length

By Malcolm Smith

As has been covered in a previous issue (see here), file systems including NTFS record three sizes per stream: allocation size, file size, and valid data length. This article explores in more detail the third of these. Documentation on this topic is limited, and the best available sample for developers is the FastFat sample which doesn't implement valid data length fully, and also does not have the same behavior as NTFS or ReFS.

Valid data length (VDL) sounds deceptively straightforward. Conceptually, data prior to this location in the file is considered 'valid', meaning it may contain user data. Data after this point in the file is 'invalid', meaning it should be zero. The complexity with this stems from the interactions between valid data at different layers in the system. Consider that data may be valid in the cache which is not yet valid on disk, and data may be valid due to a memory mapped write that the cache or file system have no awareness of.

So rather than considering valid data length as one piece of data, it can be thought of as three separate values:

- Valid data on disk;
- Valid data in cache, which is greater than or equal to valid data on disk;
- Valid data in a memory mapping, which is greater than or equal to valid data in cache.

Basic Writes – File System and Cache Manager Cooperating

Obviously the file system must update its on-disk value in response to user non-cached writes, and the cache value in response to user cached writes with CcSetFileSizes. For paging writes however, the process can be more complicated.

The Cache Manager aims to ensure writes are as sequential as possible and are of an appropriate individual size and priority. As part of doing this, the Cache Manager **can** also track valid data and update the on disk value in an aggregated way. For this support to be available, the file system must notify the cache of any regions being implicitly converted from invalid to valid via CcZeroData, and having done so, the file system will be notified (via FileEndOfFileInformation call with its IO_STACK_LOCATION's Parameters.SetFile.AdvanceOnly member set to TRUE) that a series of paging writes have completed, which allows the on-disk notion of valid data to advance.

For example, if a 3MB file currently has 1MB of valid data, and the user writes 1MB at offset 2MB into the cache, the file system must call CcZeroData on the range between 1MB and 2MB, then CcCopyWrite at 2MB to 3MB, then CcSetFileSizes. Since the cache now has a sequential data to write, it can write the data as it chooses and later indicate to the file system to advance on disk valid data to 3MB.

The consequence of using this support is that the file system must consider the origin of paging writes. If a paging write occurs as part of the Cache Manager's lazy writer, valid data does not need to be updated as part of the write but will instead be updated in the AdvanceOnly callback. Conversely, if the write originates from the Memory Manager's mapped page writer or a cache flush, no such callback will be made, and the file system must perform any required zeroing and update both on disk and in cache valid data at the time of the write. These can be distinguished by their preacquire callbacks; by convention file systems should call IoSetTopLevelIrp with FSRTL_CACHE_TOP_LEVEL_IRP for lazy write operations, and having done so, IoGetTopLevelIrp can be used to test for this condition.

Read and Write Interactions due to the Cache Manager

The situation with reads is also complicated somewhat by this optimization. Naively, an attempt to read from disk data beyond valid data on disk should result in zeroes. But this optimization introduces a race condition, because pages can be written to disk, become clean, and leave memory before valid data on disk has been updated. In normal operation, data between valid data on disk and valid data in cache would always be dirty in the cache and the file system would not need to consider this case; but due to this race condition, reads for this region can occur and must therefore be treated as valid. So, counter intuitively, a paging read from disk should be compared to valid data in the cache to determine how to zero it.

(CONTINUED ON PAGE 13)

Valid Data Length (Cont.)

(CONTINUED FROM PAGE 12)

The reason for updating cached valid data at the time of a paging write is due to the third type of valid data: writes made via memory mapped views. In this case a page can be modified outside of the file system or Cache Manager's control or awareness, resulting in the otherwise paradoxical valid data beyond valid data length. It is not valid in the presence of memory mapping to assume that pages beyond valid data length must be zero due to this condition. For the same reason, a call to CcZeroData is not guaranteed to result in zero data, because pages may have been modified that the file system does not yet know about, and those pages must be considered valid. CcZeroData will silently decline to zero pages in that case.

Although dirty data is tracked at the page level, valid data is a byte level concept. A mapped view may also modify data on the same page as a cached write. When this occurs, the Cache Manager will attempt to write the page, but is not aware that it contains data which is beyond the cache's current valid data length, and would not be included in any AdvanceOnly callback. It is important that the file system allow the cache to write the page (as the cache will not allow the mapped page writer to write it), but it follows that the file system must be willing to update cached VDL via CcSetFileSizes during the paging write to ensure a subsequent AdvanceOnly callback covers the entire range that was written. Because valid data is a high water mark, this issue is only problematic for the final page of valid data, where valid data length falls within that page.

Cache Manager Cooperation, or Not

The Cache Manager optimization is used by NTFS but is optional for other file systems. A file system may indicate that the Cache Manager should not attempt to track valid data by using the special value of MAXLONGLONG as the valid data length in a CcSetFileSizes call. Having done so, the file system must take care to update valid data on disk in response to all paging writes. This is the approach that ReFS takes as of Windows 8.1 and Server 2012 R2, leaving all data potentially valid in the cache and zeroing to the disk as needed when paging writes occur.

Regardless of the approach chosen, some form of synchronization is required when implicitly zeroing to ensure that user data is not destroyed inadvertently by racing writes to different ranges. The change between NTFS behavior and ReFS behavior in 8.1 is where this synchronization is performed. In NTFS, because data is zeroed through the cache with CcZeroData, synchronization must be provided with extending cached writes. In ReFS, where data is zeroed on disk and not through the cache, synchronization is used at noncached or paging write time, eliminating the need to serialize cache extensions.

This difference surfaces via differing implementations of FSCTL_QUERY_FILE_REGIONS. Because NTFS only remembers and enforces against cached VDL, it can only answer queries for cached VDL (FILE_REGION_USAGE_VALID_CACHED_DATA). Because ReFS only remembers and enforces against on disk VDL, it can only answer queries for noncached VDL (FILE_REGION_USAGE_VALID_NONCACHED_DATA). In both cases, note this API has limited applicability because there is no guarantee that data outside of a valid range contains zeroes due to the memory mapped writing case discussed above, and in the case of NTFS, it is also not guaranteed that ranges within the valid range are necessarily stored on disk.

The definition of FSCTL_QUERY_FILE_REGIONS allows for a file system to implement validity tracking on a per-range basis as opposed to a high water mark, which is what classic valid data length describes. Future versions of ReFS exercise this capability by recording on a much finer granularity which regions contain valid data on disk, eliminating the need for implicit zeroing and the synchronization required to implement it.

Malcolm has been working on File Systems and Filters at Microsoft for over a decade. Despite eschewing UI, he occasionally contributes to fsutil, and is the author of the hugely unpopular sdir tool at sdir.codeplex.com. He can be found loitering on ntfsd.

Windows 10 WDK and VS 2015 (Cont.)

(CONTINUED FROM PAGE 11)

You define the Target OS Version and the Target Platform (Universal, Desktop, or Mobile) in the new "Driver Settings" section of your driver project's Properties. See *Figure 2*.

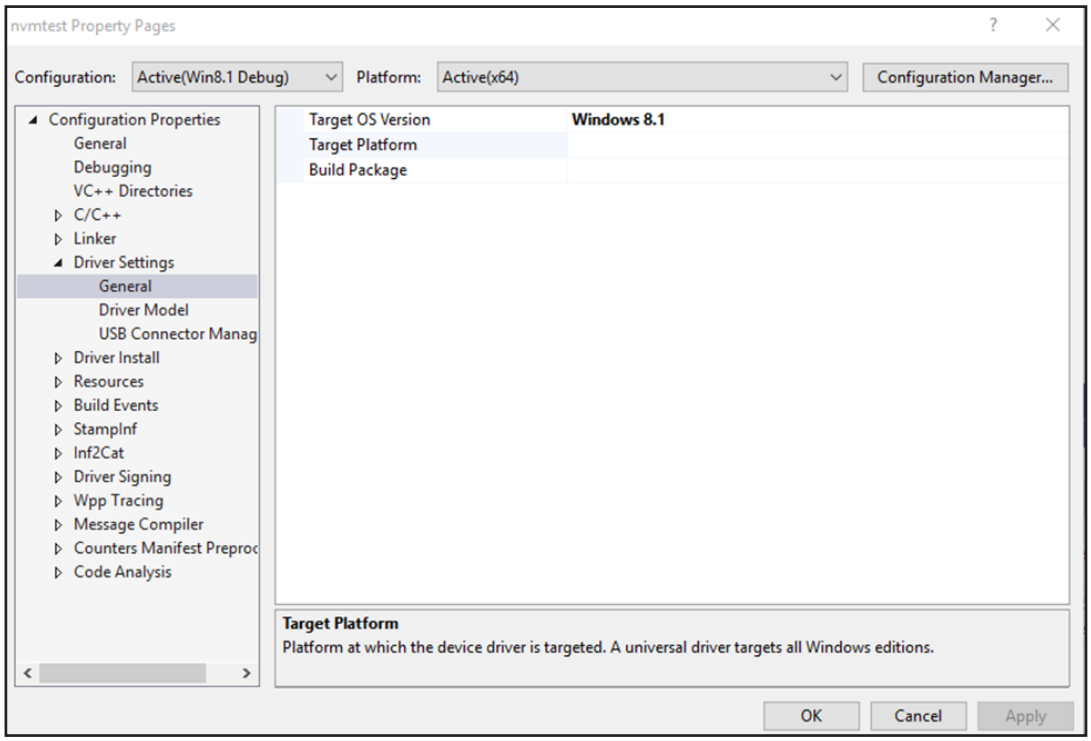


Figure 2—Windows 10 WDK Driver Properties

Ah! You might be wondering about what the "Universal" platform is all about. In short, it's the setting that will allow your drivers to properly support **any** Windows platform – Desktop, Server, or Mobile. For a quick rundown on Universal Drivers, see the sidebar entitled, *A Universal Driver? Do I Care?* (P. 16)

So how do you create your own Solution Configurations? It's easy, really. Within VS, open Configuration Manager. There are several ways to do this, but perhaps the easiest is to click on the "Configuration Manager…" button on the top right of the Project Properties dialog box… see *Figure 2*). Within Configuration Manager, pull down the "Active solution configurations" drop-down, and select "<New…>" as shown in *Figure 3*.

Choose a name for your new configuration, and choose to copy the settings from an existing Solution Configuration to make your life easier. You can see in *Figure 3* I've already created a configuration for myself named "Universal Debug."

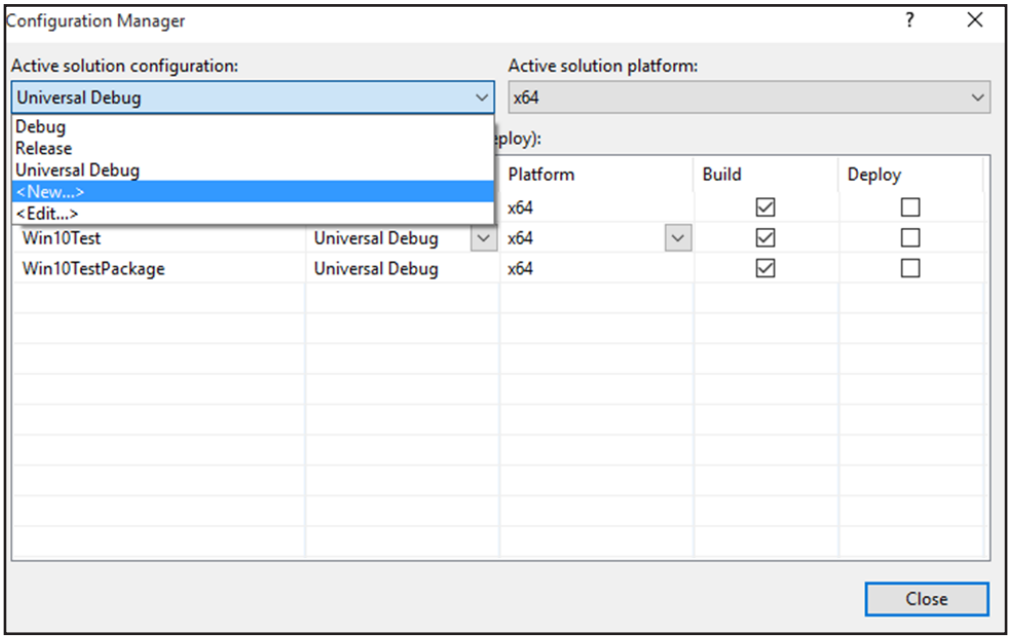


Figure 3—Creating a new Solution Configuration

Once the new Solution Configuration has been created, ensure it's selected as the Active Solution Configuration and select the Project Settings you want to use in that Configuration. These settings can obviously include Target OS Version and Target Platform. But they can also include any other setting you may want to vary among your Solution Configurations. Like, for example, signing and StampInf processing. You could easily create a Solution Configuration named "Project Release" that builds the Release version of your components, selects Release Signing, and tells StampInf to stamp a specific version into your INF's DriverVer.

(CONTINUED ON PAGE 15)

Windows 10 WDK and VS 2015 (Cont.)

(CONTINUED FROM PAGE 14)

The key point to keep in mind here is that as the WDK becomes more integrated with VS, we get to leverage increasing amounts of Visual Studio's capabilities to make our driver development jobs easier.

Where's My Code Analysis Window!?

It's dead, Jim. No, not code analysis. Certainly not Code Analysis! But the separate window that appears with code analysis warnings after your driver builds is gone. Code analysis warnings now appear along with all the other errors and warnings about your code, in the "Error List" window that's traditionally located at the bottom of VS. You can see this in *Figure 4*. We'll miss the dedicated Code Analysis Window. Sigh!

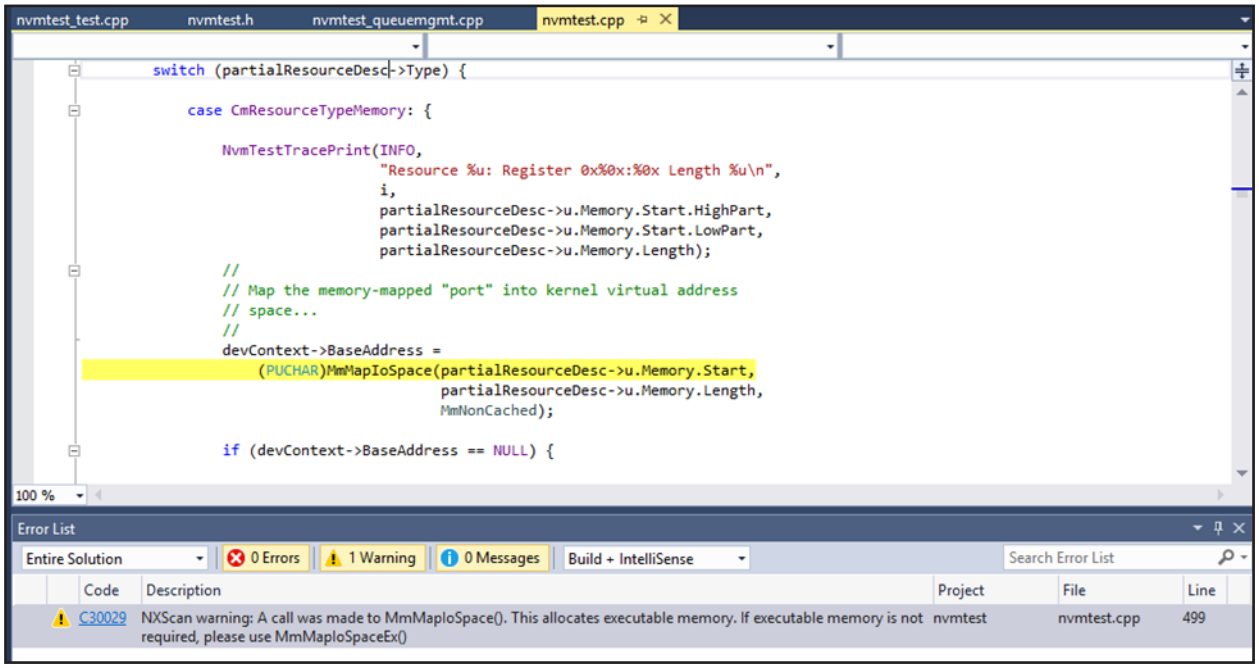


Figure 4—Farewell to the Dedicated Code Analysis Window

Anything Else?

Actually, there's a lot. But we don't have the space to get into much more right now. Static Driver Verifier works almost identically to the way it works in the Windows 8.1 Update WDK. Driver testing (including automated deployment) is rumored to actually work and be far more reliable than it has been in the past, but we haven't had time to try this yet. And, no… "Calculate Code

(CONTINUED ON PAGE 16)

Windows 10 WDK and VS 2015 (Cont.)

(CONTINUED FROM PAGE 15)
Metrics" still only works on Managed Code (why they can't provide me the LOC, Cyclomatic Complexity, and other metrics for my driver is frankly beyond me… it would actually be helpful).

In Summary
There are a lot of changes in VS 2015 and a lot of cool things in the Windows 10 WDK. If you've been waiting to install VS 2015, to be sure it won't break your existing development environment, our tests indicate that you won't encounter any problems. Dive in, and let us hear what new features you find, and what old features you miss!

Follow us!


A Universal Driver? Do You Care?

Starting with the Windows 10 WDK, the concept of Target Platform was introduced. Therefore, in addition to Target OS Version, and Solution Platform (the "Solution Platform" is Visual Studio's way of indicating the target processor architecture), there's now a Target Platform that indicates the environment to which you're targeting your driver.

There are presently three possible Target Platforms:

- Desktop
- Mobile
- Universal

The **Desktop** Target Platform is used to exclusively target "traditional" Windows client and server systems. The **Mobile** Target Platform indicates that your driver is exclusively targeting Windows 10 Mobile.

Without a doubt, the most interesting alternative is the **Universal** Target Platform. When you select Universal and successfully build your driver, your driver will be able to run on Windows systems running on Desktop (including Server), Mobile, and IoT Core platforms. Note that the Universal Target Platform can only be used if the Target OS Version for your Configuration is Windows 10 or later.

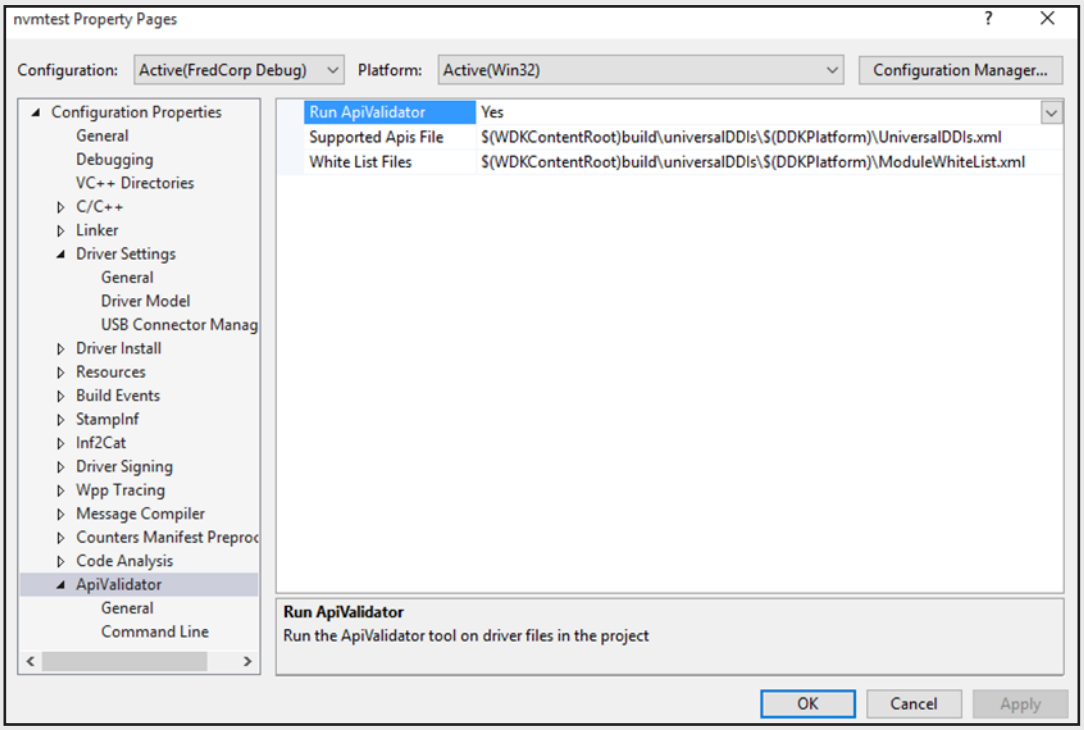
The concept of a Universal Target Platform is simple: You agree to only use a specific subset of APIs and/or DDIs in your driver, in return for the guarantee that your driver will successfully load on any Windows platform. How do you know which APIs/DDIs you can use in your driver? The documentation pages on MSDN have been (mostly) modified to indicate the Target Platform supported by that API/DDI. If the DDI says "Universal" you can use that function in a driver built for the Universal Target Platform. How limited is the list of APIs/DDIs that you can call? For kernel mode drivers, not very restrictive at all. In fact, we'd be surprised if any KMDF driver is calling any DDIs (WDF or WDM) that aren't supported on the Universal Target Platform. Certainly, all of the OSR-written kernel-mode drivers that we've tried here have all met the requirements. For user-mode drivers, things are a bit more limited. The more your user-mode driver does that is outside the realm of what a "traditional" driver supports, the more likely you are to be calling an API that's not supported by the Universal Target Platform.

You can get a quick view of whether or not you're using any prohibited APIs by selecting the Universal Target Platform, and then enabling ApiValidator (see *next page*).

(CONTINUED ON PAGE 17)

Windows 10 WDK and VS 2015 (Cont.)

(CONTINUED FROM PAGE 16)



Use ApiValidator to Ensure Your Driver is Universal Platform Compliant

Perhaps the best question about the Universal Target Platform is "Do you care?" If your driver is meant for a specific environment, whether that's the traditional desktop environment or a Windows Phone, why should you care if it's calling any DDIs or APIs that aren't part of the Universal Target Platform? The answer to that is easy: Because it makes your driver more future-proof. We all know that most code, especially driver code, lives on long after its initial intended use. By building to the Universal Target Platform you can be sure that your driver is using the subset of DDIs/APIs that are most appropriate for general driver use, while also ensuring that your driver can run on another platform… perhaps Windows IoT Core… at some point in the future.

KDNET Debugging (Cont.)

(CONTINUED FROM PAGE 9)

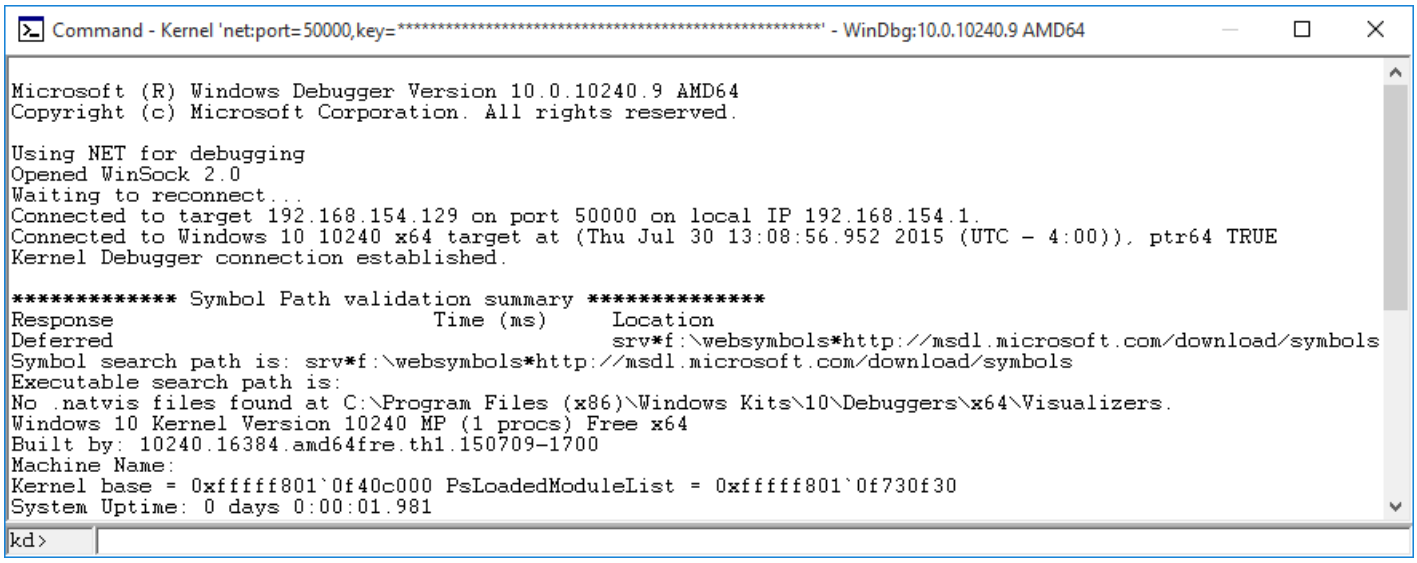


Figure 5—Success!

Step 5: Success!

If you've configured everything properly up to this point, you should be greeted with your familiar connection message from WinDbg (*Figure 5*).

But What About Boot Debug?

In the above steps, we've only enabled normal kernel debugging. Sometimes it's necessary to enable boot debugging, for example, to allow you to load an unsigned boot start driver. Unfortunately, *network boot debugging requires an EFI BIOS* and is not supported with a traditional PC-AT BIOS. While it should be possible to make this work in VMware Workstation (you can boot your VM with an EFI BIOS), we were not able to make it work at the time of writing this article. If we figure out the magic steps to make it work we'll be sure to update.

Peter Pontificates... (Cont.)

(CONTINUED FROM PAGE 5)

You can see that these are all the sorts of well thought-out replies presage a thoughtful discussion about the release of a new operating system. So, given that we were getting, you know, exactly nowhere, the above nonsense was usually followed by "If you have a few minutes, maybe you could drop by my place and take a look at my computer. *You know all about this stuff*. I just cooked up a new batch of cookies / meth / beer / organic weed killer / pasta."

Hmmmm. This was not progress. As the questions about upgrading to Windows 10 have continued unabated, I've defaulted to saying "Oh, oh, oh, you've got to excuse me, I urgently need to find a toilet!" But I can't use THAT one forever. Especially not after having already used it two or three times on the same person.

So, that brings us to my request for help. How can I fend off the inquiries I receive on a daily basis about Windows 10 from people (a) who don't have a clue what they're talking about, (b) want to entice me to lay hands on their machine with the promise of something home-cooked, and (c) won't really understand whatever answer I give them in any case. Surely somebody has a strategy to share?

Follow us!

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.

OSR Seminar Schedule

Seminar	Dates	Location
Internals & Software Drivers	21-25 September	Amherst, NH
WDF Drivers: Core Concepts	5-9 October	Amherst, NH
Developing File Systems	20-23 October	Seattle, WA
WDF Drivers: Core Concepts	2-6 November	Palo Alto, CA
WDF Drivers: Core Concepts	2-6 November	Amherst, NH
Kernel Debugging & Crash Analysis	9-13 November	Amherst, NH
WDF Drivers: Advanced	8-11 December	Amherst, NH

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.

THE NT INSIDER - Hey...Get Your Own!

Just send a blank email to join-ntinsider@lists.osr.com — and you'll get an email whenever we release a new issue of The NT Insider.

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 then 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.