

Re: Accessing Ndis miniport from user mode application

Source:

<http://www.tech-archive.net/Archive/Development/microsoft.public.development.device.drivers/2007-06/msg00674.html>

- *From:* Anton Bassov <AntonBassov@xxxxxxxxxxxxxxxxxxxxxxxxxxxxx>
 - *Date:* Fri, 22 Jun 2007 18:33:00 -0700
-

Well, everything depends on your objectives, but keep in mind that, by sticking to WMI, you confine yourself to strictly defined model – there are well-defined OID

that you can pass to your driver and well-defined status values that your driver can indicate. Unlike WMI, IOCTL model allows you do define a proprietary model of communication between your app and driver. According to your original post, you want to be able to send "non-standard" commands to your driver and to get notified about events. In such case, WMI is of no help to you – as I told you already, you should go for IOCTL model.

Another point to consider is that WMI may be used by .NET languages. As a general rule, everything that works for high-level languages like VB is a pain in the back (actually, few inches below it) for C programmers, because all "ugly details" are handled by high-level language's runtime behind the scenes and C programmers have to do everything manually. The way you put it, you want to write as little code as possible. Therefore, if your app is written in C, you should go for IOCTL model –this is the standard way of dealing with drivers in C.

Anton Bassov

"miki" wrote:

On Jun 21, 9:44 pm, Anton Bassov
<AntonBas...@xxxxxxxxxxxxxxxxxxxxxxxxxxxxx> wrote:

Steve,

You make good points and from the limited description I provided, you would appear to be correct. However, within the DNE framework, we have the ability to uniquely identify the application as part of a DNE filter so we know that

Re: Accessing Ndis miniport from user mode application

the port/application combination is valid.

This is why I said "at least the way you have described it" – it was obvious to me that you had to do some other work, although, as it turns out, it involved much more than I could possibly suspect. This changes quite a lot – apparently, you just took this approach because all additional work had to be done anyway, regardless of the way IM and app communicate with one another. In such case, your approach seems to be justified – as long as it does not involve any additional work, communication between IM and its client app via the socket seems to be quite convenient.

Anton Bassov

"Steve Jackowski" wrote:

Anton,

You make good points and from the limited description I provided, you would appear to be correct. However, within the DNE framework, we have the ability to uniquely identify the application as part of a DNE filter so we know that the port/application combination is valid. Also, we/our customers typically implement a proprietary mechanism for exchanging data which wouldn't be visible to arbitrary applications. I note that this approach is particularly useful where large amounts of data need to flow directly between the driver and a user-space application. Among the applications using this approach are WAN and Application Accelerators, Identity management, and Content filtering applications. There are, indeed situations where it is more complex to keep an IOCTL outstanding to receive enough data...

Steve

Re: Accessing Ndis miniport from user mode application

"Anton Bassov" wrote:

Steve,

The approach that you have mentioned leaves the potential security hole and is quite fragile in itself, at least if implemented the way you have described it.

If you want to make sure that only users with Admin privileges are able to send control requests to your driver (and I believe one normally has a good reason to want things work this way), your task is just straight-forward if you rely upon NdisMRegisterDevice() – all you have to control is an access to adapter's standalone device object, i.e. you have to decide who is able to call CreateFile() on it successfully. This task can be achieved by build-in access control mechanisms, which makes your driver compliant with the standard NT security model.

However, this is not the case with sockets – you just cannot tell the system "Please make sure that UDP port XYZ is available only to privileged users", can you? You cannot make your IM responsible for this check either – by the time a packet reaches it, context of the sending process may be already lost. Furthermore, even if you are not concerned about the security issues, what happens if UDP port that you want to use is already being used by another app???

Re: Accessing Ndis miniport from user mode application

Therefore, you have to introduce the additional TDI-level filter driver in order to make sure that no caller, apart from your app, is able to specify the given UDP port number upon its call to bind(). Please note that, in order to get into a conflict with your app, another process does not even have to explicitly specify your reserved port number upon its call to bind() – TCPIP may choose the target port accidentally if it specifies INADDR_ANY. Therefore, your TDI filter has to foresee this possibility and be able to deal with it.

To summarize, in most cases you are about to do absolutely unnecessary additional work if you take this approach – the only type of project when it seems to be appropriate is a firewall.

Anton Bassov

"Steve Jackowski" wrote:

Hi Miki,

While I agree with Anton and the vast majority of our DNE customers use IOCTLs and we use them ourselves, there are a few (and we've done this as part of some of our products) who bypass this mechanism by using sockets. In this case, the application binds to a port and then sends/receives UDP on

Re: Accessing Ndis miniport from user mode application

that port. The IM (or in our case, DNE plugin) captures UDP traffic on that port and can inject incoming to send back to the application. If the messages are small with no fragmentation, this is pretty straight forward and provides a bi-directionally indicated message rather than forcing the application to keep IOCTLS ready to get information from the IM. This is not for all IMs/application interaction, but there are certainly applications that can take advantage of it.

Steve

"miki" wrote:

Hi

The NDIS miniport driver my company write should support some custom features. Those features can be set and queried by user mode applications. Also the driver

Re: Accessing Ndis miniport from user mode application

should be
able to
signal
the
application
on certain
events.

In <http://www.ndis.com/faq/QA10290101.htm> they
describe
several
ways
to do that:

1. Using a
device
control
object that
the miniport
creates and
the
application
IOCTL it
2.
Accessing
custom
OIDs
through
IOCTL_NDIS_QUERY_GLOBAL_STATS
(support
only OID
query – so
its not good
for me)
3.
Accessing
custom
OIDS
through
Protocol
driver
4.
Accessing
custom
OIDS
through
WMI.

Re: Accessing Ndis miniport from user mode application

Can anyone
tell me what
is the
preferred/recommended
method (I
tend to
think that
accessing
the OIDs
through
WMI, but I
would like
to hear
what the
pros think)

I am writing
both NDIS
5 and NDIS
6 miniport
drivers so I
prefer a
solution
which
applies to
both

Thanks
Miki– Hide
quoted text
–

– Show quoted text –

Thanks Anton and Steve,

I did not clarify several things:

1. The amount of data that is exchange between application and user is not big.
2. I want to write as little as possibe code

Re: Accessing Ndis miniport from user mode application

Those are the things I don't like about the IOCTL:

1. It looks to me like a backdoor and not consistent with the NDIS model which is base on OIDs
2. The driver is at the mercy of the application – if the application opens an handle to the control device the driver can't unload until application closes the device
3. There is a a single control device channel for all devices the driver manages so you have to implement a multiplexing
4. In order for the driver to send data to the the application it has to signal the application to send him IOCTL or to use a queue of pending IOCTL requests

On the other hand I cant see the disadvantages of using WMI the miniport can send its own custom events through NdisMIndicatStatus those events are translated to WMI events which the application can read

What am I missing?

Miki