

# Re: Multicast Directshow Graph Bridging, COW Rustling, & the Use of Portable Holes in Cartoon Physics.

---

*Source:*

<http://www.tech-archive.net/Archive/Development/microsoft.public.win32.programmer.directx.video/2008-05/msg00>

---

- *From:* Jamie Faye Fenton <[jamiefaye@xxxxxxxxxx](mailto:jamiefaye@xxxxxxxxxx)>
  - *Date:* Fri, 30 May 2008 04:45:31 -0700 (PDT)
- 

On May 18, 4:34 pm, Jamie Faye Fenton <[jamief...@xxxxxxxxxx](mailto:jamief...@xxxxxxxxxx)> wrote:

Hello Boys and Girls. I am a mad scientist on the loose who wants to hack DirectShow and its progeny in new and even more unusual ways! I have already done a number of strange things, most recently a "Buffered-data switching matrix w/ plug in media processors." Beyond its potential as an acronym, this DS Filter collection uses mapped-memory to share buffer pools across processes and works pretty well. You get two basic Direct Show filters, a \$ink and a \$ource. (or Sink and Source if you are likely to hate it when people substitute \$ for S). There are also specializations that act more like renderers or capture filters, including one that combines 4 channels into a 2x2 quad.

A sink or source can be commanded to "tune into" a virtual channel group and either copy buffers into or out of the ring so selected. You can have any number of channels and any number of groups, the mapping can be changed in a few microseconds, and video formats and sizes are either renegotiated or converted on the spot to what each receiver requires. Fan-out is not limited. Buffering can be set to whatever, typically 4-6 frames works out well for the relatively low bandwidth stuff web-cam applications demand.

But wait, there is more! Each source filter can be sent an XML description of a synthesizer patch to run during the time it transfers the buffer into the destinations world. Each output stream has its own video-oriented virtual machine with a frame stack and a list of plug ins to call in sequence (I use the Freeframe open standard for this, which gives us 300 or so effects, and will likely add the Japaneze EffectTV package and whatever else I can scrounge).

So we now have a Swiss Army Direct Show filter that you can put into your old, inflexible, DS Filtergraph of your choice and have it re-emerge in other places and in other roles, with add-on eye candy. You can share web cams indiscriminately, glue video applications together,

Re: Multicast Directshow Graph Bridging, COW Rustling, & the Use of Portable Holes in Cartoon Physics.

and do a wild variety of other things people always ask about on this newsgroup but get frustrated doing.

I may even ship the darned thing. I am hesitant, mostly because doing it the first time gave me insight on how to do it better. What will likely emerge is a scheme based on what I already have with a transitional road-map to get where I want it to go.

So what do I want to do that I can't do now? A big one is to copy frames less. While I am pretty careful about that topic, there are a few places where I can squeeze better performance out of things. I also want to keep the semantic noise down – by doing DirectShow stuff using DirectShow compatible APIs, etc., to the maximal extent without buying into some of the policy choices made way-back-then. I also want to add YUV formats, audio, and add a lot more to the DirectX side of things.

So here a few research directions.

Hacking the GMFBridge Toolkit – In particular I want to add "multicast eavesdropping." This means we have the GMFBridge work like it already does, except that we graft on our own allocator that does some trickery. The sink and source filters in GMFBridge work as they always did, but it will also be possible to "Eavesdrop" on the signal flow, much the same way my MoneyShot source filters peer into a ring buffer. An ideal scheme for this would have the grafted-on allocator issue each GMF-source a virtual memory alias with COW (Copy on Write) semantics. Unlike the infinite Tee, each GMF-Source would get its buffer right-away, and only the prime-recipient (or non-eavesdropper) would be able to influence the quality-control loop, etc.

Since the way COW works in Windows is that each view gets its own version of whatever it changes, the original buffer is unmolested. (We can allow the non-sneaky sink to be an exception if that matters).

Aside from the fact that a lot of MediaSamples already do shared-memory stuff (which may make some of the sneaky stuff I have in mind easier), are there any show-stoppers that any of you know about? (I expect to handle the renderer./ video memory problem by either copying or mapping when I can. DRM I figure will go away pretty soon, or the angry mobs will burn their villages to the ground.

Thinking about virtual memory hacking has made me consider this "Willey Coyote" scenario: I know there is a buffer somewhere, perhaps in another process, that has data I want to have a copy of. I want to put a portable hole under that roadrunner. What I want to do is in a "close to atomic" operation declare myself to be one of the viewers of that buffer as a shared-memory or memory mapped file, and let the other guy view it the same way. I write to that zone, I go COW, he writes to his, he goes COW. What I got was a temporal snap shot that I can work with at my leisure. Where this gets tricky is when the

address range is already mapped to some other shared-memory scheme and we don't want to trouble our victim in any way...

Another wacko idea is to hang out where he derefs his Media Samples and returns them to his allocator pool. Then I cut a deal with the allocator where he gives me non-zero pages full of sample-stuff and I give him nice fresh zeros in return. (Note this isn't a hack for electronic espionage, I just want to copy my samples a few times less than necessary if I can get away with it). Here the big issue is that everybody links to a different version of the base classes, so placing intercepts could be a nightmare.

I already do tight windowing on my video buffers, and hence create plenty of page-faults. It would be nice to have fewer faults per frame. Perhaps kernel-mode driver hacking will be my next-big-thing.

As for Money\$hot, I will hopefully be going beta pretty soon and I will announce that when its ready to go. I am planning to make it an open-source/free ware kind of thing. I also have to get it done and find some work that pays, since Money\$hot is a good description of my financial affairs. So if anyone out there needs a Direct Show expert from the lunatic fringe, I am available.

As for the technical issues and implications of some of the ideas mentioned before, your insights are vital. It is so easy to blunder around in the dark with this API and I have multiple bruises and scars. Still I think the creation of a flexible and elegant multimedia API is possible and know that our group here knows how.

I finally found the right place to go for information about this problem area. Its a group called microsoft public development device drivers. People regularly ask over there questions about sharing web cameras and other capture devices, and how to map kernel memory into user mode. I read the last six months of their traffic, and it is in many ways like the chatter here – a few super-helpful (and a little gruff) MVPs that disabuse your illusions about what Windows will let you get away with. To summarize what I learned:

Do not set up any sort of ongoing memory share bridging between kernel and user space – instead employ an "inverted callback" where the user side sets up the buffer and makes an I/O call which is then held pending. When the provider has the buffer ready, it transfers the data, either by direct memory copying or perhaps by a virtual memory remapping (although it is unclear if that saves much).

.