You already know the answer since you do it yourself

devblogs.microsoft.com/oldnewthing/20120413-00

April 13, 2012

Raymond Chen

A customer was writing a program that performed virtual drag/drop. They were using the `IStream` technique but found that many applications don't support drag/drop of virtual content. They support only `CF_HDROP`. What's more, often these applications query for `CF_HDROP` on `DragEnter` not because they want to access the file, but just because they want to get the file names (for example, because they want to put up the no-entry cursor if the file types are not ones the application supports).

> Given that we want to be able to drop content onto applications which do not support drag/drop of virtual content, we have the problem of knowing exactly when to generate the content into a temporary file. If we generate the content too soon, then we may end up going to a lot of effort of creating a file that won't actually be used; if we generate it too late, then the application will try to open the file and find that it isn't there. When is the correct moment to generate the content? Is there some set of rules by which applications which do not support virtual drag/drop indicate whether they are obtaining the `CF_HDROP` just to see the names of the files, and whether they are obtaining it because they want to go open the files?

If you think about it, you already know the answer to this question because you already do it yourself when you write code that operates the client side of the drag/drop contract. When you write your program that accepts `CF_HDROP` content, do you use any special signal to tell the data object, "Hey, I'm asking for `CF_HDROP` just because I want the file names, but I promise not to try to open the files yet"? No, you don't, so why would you expect any other application to? Even if there were rules surrounding this signaling protocol, the fact that they are widely ignored (because *even you yourself ignore them*) means that you can't rely on the client performing them anyway. The rule for `CF_HDROP` is that at the moment you offer `CF_HDROP` content, the files must already exist. The `CF_HDROP` clipboard format was created by File Manager in Windows 3.1, and File Manager did not support virtual content. The only thing it knew how to drag and drop was files, and the only thing it could drag was files that already existed. In a sense, the rules around `CF_HDROP` were not so much codified by rule as codified by circumstance. Since the only things you could drag were files that already existed, those became the *de facto* rules for `CF_HDROP`. Sorry. Note that there is a little you can do: If an application calls `QueryGetData` for `CF_HDROP`, that does not force you to create the file content yet, because `QueryGetData` is just a yes/no query as to

whether you could produce `CF_HDROP` if asked. You're not being asked to do so, so you can just say "Why yes, I have files" even though you don't yet. It's only at the `GetData` that you have to return a list of file names and therefore must create the files. Most applications are kind enough to use `QueryGetData` when they are not interested in the files yet but only in the potential for files, but there are some which use `GetData` on the `DragEnter`, and those applications will force you to commit to creating the content even if the user ultimately abandons the operation without dropping. (You could try deferring the creation until your `IDropSource::QueryContinueDrag` returns `DRAGDROP_S_DROP`, but programs which sniff the file contents during `DragEnter` will not find the file there, and they probably won't like that. It also doesn't work if the transfer is done via copy/paste rather than drag/drop—and you need copy/paste support for accessibility—since your drop source is not active during a copy/paste.)

Exercise: How do you know when it's safe to delete the temporary files?

Raymond Chen

Follow