# Reading a contract from the other side: Simulating a drop

devblogs.microsoft.com/oldnewthing/20080724-00

July 24, 2008

Raymond Chen

Most people, when they think of the `IDropTarget` interface, think only of implementing a drop target. But you can read the contract from the other side, because the description of how a drag source interacts with a drop target tells you how to be a drag source.

To summarize, the sequence of drop target operations go like this:

- `IDropTarget::DragEnter` is called to indicate that an object has been dragged into the drop target. If the drop target returns a failure code, then the drop operation ends immediately.
- Otherwise, `IDropTarget::DragOver` calls are made to advise the drop target as to the object's location.
- If the user completes the drop operation, then call `IDropTarget::Drop`. Otherwise call `IDropTarget::Leave`. A drop operation can fail to complete because the user hit the Escape key, for example, or dragged the mouse out of the drop target.

Let's write a simple program that drops one file onto another.

```
#include <windows.h>
#include <shlobj.h>
#include <shellapi.h>
#include <tchar.h>


… Insert the function GetUIObjectOfFile here …


int __cdecl wmain(int argc, WCHAR **argv)
{
 if (argc == 3 && SUCCEEDED(CoInitialize(NULL))) {
  IDataObject *pdto;
  if (SUCCEEDED(GetUIObjectOfFile(NULL, argv[1],
                        IID_IDataObject, (void**)&pdto))) {
   IDropTarget *pdt;
   if (SUCCEEDED(GetUIObjectOfFile(NULL, argv[2],
                         IID_IDropTarget, (void**)&pdt))) {
    POINTL pt = { 0, 0 };
    DWORD dwEffect = DROPEFFECT_COPY | DROPEFFECT_LINK;
    if (SUCCEEDED(pdt->DragEnter(pdto, MK_LBUTTON,
                                 pt, &dwEffect))) {
     dwEffect &= DROPEFFECT_COPY | DROPEFFECT_LINK;
     if (dwEffect) {
      pdt->Drop(pdto, MK_LBUTTON, pt, &dwEffect);
     } else {
      pdt->DragLeave();
     }
    }
    pdt->Release();
   }
   pdto->Release();
  }
  CoUninitialize();
 }
 return 0;
}
```

This is a pretty straightforward implementation of the host side of the drag/drop protocol. Run this program with the *full paths* to two files, the first being the file to drop, and the second being the file you want to drop it onto. (Modifying this program to accept relative paths is left as an exercise for the reader.) For example, you might try

```
fakedrop c:\autoexec.bat c:\windows\notepad.exe
```

Now, sure, dropping a file on a program is nothing exciting. You could've just run the program with the file as the command line argument, after all. But that's looking at it too narrowly; you are simulating a drop operation, after all. For example, you can drop a file onto a shortcut to a printer, and the file will print; or you can drop a file onto a folder and it will be copied there (since we specified `DROPEFFECT_COPY | DROPEFFECT_LINK`, but folders prefer

copy to link if the Ctrl+Shift keys are not held down); or you can drop a file onto the `Mail Recipient.MAPIMail` shortcut in your "Send To" folder to create a mail message with the file as an attachment.

Oh wait, that last example with `Mail Recipient.MAPIMail` doesn't work. We'll look at why next time, although I suspect you already know the reason.

Raymond Chen

**Follow**