Improving the performance of CF_HDROP by providing file attribute information

devblogs.microsoft.com/oldnewthing/20140609-00

June 9, 2014

Raymond Chen

The `CF_HDROP` clipboard format is still quite popular, despite its limitation of being limited to files. You can't use it to represent virtual content, for example.

For all of you still using `CF_HDROP`, you can improve the performance of drag/drop operations by adding a little more information to your data object.

Observe that the `CF_HDROP` clipboard format is just a list of paths. Some drop targets care about whether the paths refer to directories or to files, and since `CF_HDROP` does not provide this information, the drop targets are forced to access the disk to get the answer. (This can be expensive for network locations.)

To help this case, you can add a `CFSTR_FILE_ATTRIBUTES_ARRAY` to your data object. This contains the file attribute information for the items in your `CF_HDROP`, thereby saving the drop target the cost of having to go find them.

Take our tiny drag-drop sample and make the following changes:

```
class CTinyDataObject : public IDataObject
{
  ...
  enum {
    // ~~DATA_TEXT,~~
    DATA_HDROP,
    DATA_NUM,
    DATA_INVALID = -1,
  };
  ...
};
CTinyDataObject::CTinyDataObject() : m_cRef(1)
{
  SetFORMATETC(&m_rgfe[DATA_HDROP], CF_HDROP);
}
struct STATICDROPFILES
{
 DROPFILES df;
 TCHAR szFile[ARRAYSIZE(TEXT("C:\\Something.txt\0"))];
} const c_hdrop = {
  {
    FIELD_OFFSET(STATICDROPFILES, szFile),
    { 0, 0 },
    FALSE,
    sizeof(TCHAR) == sizeof(WCHAR), // fUnicode
  },
  TEXT("C:\\Something.txt\0"),
};
HRESULT CTinyDataObject::GetData(FORMATETC *pfe, STGMEDIUM *pmed)
{
  ZeroMemory(pmed, sizeof(*pmed));
  switch (GetDataIndex(pfe)) {
  case DATA_HDROP:
    pmed->tymed = TYMED_HGLOBAL;
    return CreateHGlobalFromBlob(&&c_hdrop, sizeof(c_hdrop),
                                 GMEM_MOVEABLE, &pmed->hGlobal);
  }
  return DV_E_FORMATETC;
}
```

Okay, let's look at what we did here.

First, we make our data object report a `CF_HDROP`. We then declare a static `DROPFILES` structure which we use for all of our drag-drop operations. (Of course, in real life, you would generate it dynamically, but this is just a Little Program.)

That's our basic program that drags a file.

Note that

> you are much better off letting the shell create the data object,

since that data object will contain much richer information (and this entire article would not be needed). Here's a sample program which uses the `GetUIObjectOfFile` function to do this in just a few lines. It's much shorter than having to cook up this `CTinyDataObject` class. I'm doing it this way on the assumption that your program is deeply invested in the less flexible `CF_HDROP` format, so changing from `CF_HDROP` to some other format would be impractical.

Okay, so that's the program we're starting from. Let's add support for precomputed attributes.

```
class CTinyDataObject : public IDataObject
{
  ...
  enum {
    DATA_HDROP,
    DATA_ATTRIBUTES,
    DATA_NUM,
    DATA_INVALID = -1,
  };
  ...
};
CTinyDataObject::CTinyDataObject() : m_cRef(1)
{
  SetFORMATETC(&m_rgfe[DATA_HDROP], CF_HDROP);
  SetFORMATETC(&m_rgfe[DATA_ATTRIBUTES],
               RegisterClipboardFormat(CFSTR_FILE_ATTRIBUTES_ARRAY));
}
FILE_ATTRIBUTES_ARRAY c_attr = {
 1, // cItems
 FILE_ATTRIBUTE_ARCHIVE, // OR of attributes
 FILE_ATTRIBUTE_ARCHIVE, // AND of attributes
 { FILE_ATTRIBUTE_ARCHIVE }, // the file attributes
};
HRESULT CTinyDataObject::GetData(FORMATETC *pfe, STGMEDIUM *pmed)
{
  ZeroMemory(pmed, sizeof(*pmed));
  switch (GetDataIndex(pfe)) {
  case DATA_HDROP:
    pmed->tymed = TYMED_HGLOBAL;
    return CreateHGlobalFromBlob(&amp;c_hdrop, sizeof(c_hdrop),
                                 GMEM_MOVEABLE, &pmed->hGlobal);
  case DATA_ATTRIBUTES:
    pmed->tymed = TYMED_HGLOBAL;
    return CreateHGlobalFromBlob(&c_attr1, sizeof(c_attr1),
                                 GMEM_MOVEABLE, &pmed->hGlobal);
  }
  return DV_E_FORMATETC;
}
```

Okay, let's look at what we did here.

We added a new data format, `CFSTR_FILE_ATTRIBUTES_ARRAY`, and we created a static copy of the `FILE_ATTRIBUTES_ARRAY` variable-length structure that contains the attributes of our one file. Of course, in a real program, you would generate the structure dynamically. Note that I use a sneaky trick here: Since the `FILE_ATTRIBUTES_ARRAY` ends with an array of length 1, and I happen to need exactly one item, I can just declare the structure as-is and fill in the one slot. (If I had more than one item, then I would have needed more typing.)

To make things easier for the consumers of the `FILE_ATTRIBUTES_ARRAY`, the structure also asks you to report the logical OR and logical AND of all the file attributes. This is to allow quick answers to questions like "Is everything in this `CF_DROP` a file?" or "Is anything in this `CF_DROP` write-protected?" Since we have only one file, the calculation of these OR and AND values is nearly trivial.

Okay, so there isn't much benefit to adding file attributes to a drag of a single file from the local hard drive, since the local hard drive is pretty fast, and the file attributes may very well be cached. But if you've placed thousands of files from a network drive onto the clipboard, this shortcut can save a lot of time. (That was in fact the customer problem that inspired this Little Program.)

Raymond Chen

Follow