What a drag: Dragging text

devblogs.microsoft.com/oldnewthing/20080311-00

March 11, 2008

Raymond Chen

This week's mini-series was almost titled "It's the least you could do" because I'm going to try to do the absolute least amount of work to accomplish something interesting with drag and drop. The real purpose today is to lay some groundwork, but just to have something to show for our effort, I'll show you how to drag text around.

We're going to need the `CDropSource` class from an earlier series on drag-and-drop. Also take the change from `CoInitialize` to `OleInitialize` (and similarly `CoUninitialize`), as well as the line

```
HANDLE_MSG(hwnd, WM_LBUTTONDOWN, OnLButtonDown);
```

Our mission for today is to create the tiniest data object possible.

```
#include <strsafe.h> // for StringCchCopy
#include <shlobj.h> // (will be needed in future articles)

/* note: apartment-threaded object */
class CTinyDataObject : public IDataObject
{
public:
  // IUnknown
  STDMETHODIMP QueryInterface(REFIID riid, void **ppvObj);
  STDMETHODIMP_(ULONG) AddRef();
  STDMETHODIMP_(ULONG) Release();

  // IDataObject
  STDMETHODIMP GetData(FORMATETC *pfe, STGMEDIUM *pmed);
  STDMETHODIMP GetDataHere(FORMATETC *pfe, STGMEDIUM *pmed);
  STDMETHODIMP QueryGetData(FORMATETC *pfe);
  STDMETHODIMP GetCanonicalFormatEtc(FORMATETC *pfeIn,
                                     FORMATETC *pfeOut);
  STDMETHODIMP SetData(FORMATETC *pfe, STGMEDIUM *pmed,
                       BOOL fRelease);
  STDMETHODIMP EnumFormatEtc(DWORD dwDirection,
                             LPENUMFORMATETC *ppefe);
  STDMETHODIMP DAdvise(FORMATETC *pfe, DWORD grfAdv,
                    IAdviseSink *pAdvSink, DWORD *pdwConnection);
  STDMETHODIMP DUnadvise(DWORD dwConnection);
  STDMETHODIMP EnumDAdvise(LPENUMSTATDATA *ppefe);

  CTinyDataObject();

private:
  enum {
    DATA_TEXT,
    DATA_NUM,
    DATA_INVALID = -1,
  };

  int GetDataIndex(const FORMATETC *pfe);

private:
  ULONG m_cRef;
  FORMATETC m_rgfe[DATA_NUM];
};
```

We'll learn more about those private members later. Let's start with the boring stuff: The `IUnknown` interface.

```
HRESULT CTinyDataObject::QueryInterface(REFIID riid, void **ppv)
{
  IUnknown *punk = NULL;
  if (riid == IID_IUnknown) {
    punk = static_cast<IUnknown*>(this);
  } else if (riid == IID_IDataObject) {
    punk = static_cast<IDataObject*>(this);
  }

  *ppv = punk;
  if (punk) {
    punk->AddRef();
    return S_OK;
  } else {
    return E_NOINTERFACE;
  }
}

ULONG CTinyDataObject::AddRef()
{
  return ++m_cRef;
}

ULONG CTinyDataObject::Release()
{
  ULONG cRef = --m_cRef;
  if (cRef == 0) delete this;
  return cRef;
}
```

Yawners. The constructor is interesting, though, because we use our constructor to build the array of supported `FORMATETC` s which other members will consult.

```
void SetFORMATETC(FORMATETC *pfe, UINT cf,
                  TYMED tymed = TYMED_HGLOBAL, LONG lindex = -1,
                  DWORD dwAspect = DVASPECT_CONTENT,
                  DVTARGETDEVICE *ptd = NULL)
{
  pfe->cfFormat = (CLIPFORMAT)cf;
  pfe->tymed    = tymed;
  pfe->lindex   = lindex;
  pfe->dwAspect = dwAspect;
  pfe->ptd      = ptd;
}

CTinyDataObject::CTinyDataObject() : m_cRef(1)
{
  SetFORMATETC(&m_rgfe[DATA_TEXT], CF_TEXT);
}
```

Our data object contains only thing: plain text. We set the clipboard format to `CF_TEXT`, indicating that that's the data we have. The type medium is `TYMED_HGLOBAL` because we are going to provide the text in the form of an `HGLOBAL`. The other fields are boilerplate that you will rarely have to change: The aspect is `DVASPECT_CONTENT` because we are going to provide the actual data content. The `DVTARGETDEVICE` is `NULL` because our content is device-independent. And the `lindex` is `-1` because we're going to provide all the data. I've created a helper function which uses the boilerplate values as default parameters.

The first member function that will use this helper array is one that we will use quite a bit to do the preliminary validation of incoming `FORMATETC` structures.

```
int CTinyDataObject::GetDataIndex(const FORMATETC *pfe)
{
  for (int i = 0; i < ARRAYSIZE(m_rgfe); i++) {
    if (pfe->cfFormat == m_rgfe[i].cfFormat &&
       (pfe->tymed    &  m_rgfe[i].tymed)   &&
        pfe->dwAspect == m_rgfe[i].dwAspect &&
        pfe->lindex   == m_rgfe[i].lindex) {
      return i;
    }
  }
  return DATA_INVALID;
}
```

The `GetDataIndex` method takes a candidate `FORMATETC` and looks to see whether it matches any of the ones in our table of supported formats, `m_rgfe`, returning its index or `DATA_INVALID` indicating that there was no match. Note that we consider it a match if any of the requested type media match the supported type media. For example, the caller might pass `TYMED_HGLOBAL | TYMED_STREAM`, indicating that the caller can handle receiving either an `HGLOBAL` or an `IStream`. If our format matches either one, then we'll call that a success.

Before we continue, here's a handy helper function when working with clipboard data: It takes a block of memory and turns it into a `HGLOBAL`.

```
HRESULT CreateHGlobalFromBlob(const void *pvData, SIZE_T cbData,
                              UINT uFlags, HGLOBAL *phglob)
{
  HGLOBAL hglob = GlobalAlloc(uFlags, cbData);
  if (hglob) {
    void *pvAlloc = GlobalLock(hglob);
    if (pvAlloc) {
      CopyMemory(pvAlloc, pvData, cbData);
      GlobalUnlock(hglob);
    } else {
      GlobalFree(hglob);
      hglob = NULL;
    }
  }
  *phglob = hglob;
  return hglob ? S_OK : E_OUTOFMEMORY;
}
```

The money in a data object lies in the `IDataObject::GetData` method, because this is where the data object client gets to see what all the excitement is about.

```
CHAR c_szURL[] = "http://www.microsoft.com/";

HRESULT CTinyDataObject::GetData(FORMATETC *pfe, STGMEDIUM *pmed)
{
  ZeroMemory(pmed, sizeof(*pmed));

  switch (GetDataIndex(pfe)) {
  case DATA_TEXT:
    pmed->tymed = TYMED_HGLOBAL;
    return CreateHGlobalFromBlob(c_szURL, sizeof(c_szURL),
                                 GMEM_MOVEABLE, &pmed->hGlobal);
  }

  return DV_E_FORMATETC;
}
```

Wow, that was deceptively simple. We ask `GetDataIndex` to look up the `FORMATETC`; if it's `DATA_TEXT`, we return the desired text in the form of an `HGLOBAL`. Otherwise, it's not supported, so we return an appropriate error code. Note that `CF_TEXT` is specifically ANSI text. For Unicode text, use `CF_UNICODE`.

Very closely related to `IDataObject::GetData` is `IDataObject::QueryGetData`, which is just like `GetData` except that it doesn't actually get the data. It just says whether the data object contains data in the specified format.

```
HRESULT CTinyDataObject::QueryGetData(FORMATETC *pfe)
{
  return GetDataIndex(pfe) == DATA_INVALID ? S_FALSE : S_OK;
}
```

The only other interesting method is `IDataObject::EnumFormatEtc`, which can be asked to return an enumerator that lists all the formats contained in the data object.

```
HRESULT CTinyDataObject::EnumFormatEtc(DWORD dwDirection,
                                       LPENUMFORMATETC *ppefe)
{
  if (dwDirection == DATADIR_GET) {
    return SHCreateStdEnumFmtEtc(ARRAYSIZE(m_rgfe), m_rgfe, ppefe);
  }
  *ppefe = NULL;
  return E_NOTIMPL;
}
```

If the caller is asking for the formats that it can "get", then we return an enumerator created from the shell stock format enumerator. Otherwise, we say that we don't have one.

The rest of the methods are just stubs.

```
HRESULT CTinyDataObject::GetDataHere(FORMATETC *pfe,
                                     STGMEDIUM *pmed)
{
    return E_NOTIMPL;
}

HRESULT CTinyDataObject::GetCanonicalFormatEtc(FORMATETC *pfeIn,
                                               FORMATETC *pfeOut)
{
  *pfeOut = *pfeIn;
  pfeOut->ptd = NULL;
  return DATA_S_SAMEFORMATETC;
}

HRESULT CTinyDataObject::SetData(FORMATETC *pfe, STGMEDIUM *pmed,
                                                 BOOL fRelease)
{
    return E_NOTIMPL;
}

HRESULT CTinyDataObject::DAdvise(FORMATETC *pfe, DWORD grfAdv,
                     IAdviseSink *pAdvSink, DWORD *pdwConnection)
{
    return OLE_E_ADVISENOTSUPPORTED;
}

HRESULT CTinyDataObject::DUnadvise(DWORD dwConnection)
{
    return OLE_E_ADVISENOTSUPPORTED;
}

HRESULT CTinyDataObject::EnumDAdvise(LPENUMSTATDATA *ppefe)
{
    return OLE_E_ADVISENOTSUPPORTED;
}
```

And we're done. Let's take it for a spin.

```
void OnLButtonDown(HWND hwnd, BOOL fDoubleClick,
                   int x, int y, UINT keyFlags)
{
  IDataObject *pdto = new CTinyDataObject();
  if (pdto) {
    IDropSource *pds = new CDropSource();
    if (pds) {
      DWORD dwEffect;
      DoDragDrop(pdto, pds, DROPEFFECT_COPY, &dwEffect);
      pds->Release();
    }
    pdto->Release();
  }
}
```

Fire up Wordpad and then click in the client area of our scratch program and drag and drop the invisible text over to the Wordpad window. Ta-da, the text is inserted.

This even works with Firefox to drag a URL into a Firefox window. But it doesn't work for Internet Explorer. We'll see why next time.

Exercise: Why didn't we also have to set `CF_UNICODE` text?

Pre-emptive Igor Levicki comment: "Windows Vista should be dragged and dropped to the trash can."

Raymond Chen

Follow