Why an object cannot be its own enumerator

devblogs.microsoft.com/oldnewthing/20040322-00

March 22, 2004

Raymond Chen

I've seen people using the following cheat when forced to implement an enumerator:

```
class MyClass :
  public IDataObject, public IEnumFORMATETC, ...
{
  ...
  HRESULT EnumFormatEtc(DWORD dwDirection,
                IEnumFORMATETC** ppenumOut)
  {
    _dwDirection = dwDirection;
    Reset();
    *ppenumOut = this;
    AddRef();
    return S_OK;
  }
};
```

Why create a separate enumerator object when you can just be your own enumerator? It's so much easier.

And it's wrong.

Consider what happens if two people try to enumerate your formats at the same time: The two enumerators are really the same enumerator, so operations on one will interfere with the other. For example, consider this odd code fragment (error checking deleted for expository purposes) which looks to see if the data object exposes the same data under multiple aspects:

```
IDataObject *pdto = <MyClass instance>;
// Obtain two enumerators since we will run
// each one independently.
IEnumFORMATETC* penum1;
IEnumFORMATETC* penum2;
pdto->EnumFormatEtc(DATADIR_GET, &penum1);
pdto->EnumFormatEtc(DATADIR_GET, &penum2);
FORMATETC fe1, fe2;
while (penum1->Next(1, &fe1, NULL) == S_OK) {
  penum2->Reset(); // start a new pass
  while (penum2->Next(1, &fe2, NULL) == S_OK) {
    if (fe1.cfFormat == fe2.cfFormat &&
        cf1.dwAspect != cf2.dwAspect) {
        // found it!
    }
  }
}
penum1->Release();
penum2->Release();
```

When the code does a `penum2->Reset()`, this also inadvertently resets the first enumerator. The loop runs through penum2 (which therefore also runs through penum1), and when it's done, the enumerator is left at the end of the list.

Then we loop back and call `penum1->Next()`, which immediately returns failure since the inner loop ran it to completion.

Result: The loop fails to find anything because the second enumerator corrupted the first.

Raymond Chen

Follow