How do I provide data to the sharing pane from a Win32 desktop application?

devblogs.microsoft.com/oldnewthing/20170316-00

March 16, 2017

Raymond Chen

Last time, we were able to show the sharing pane from a Win32 desktop application, but we didn't provide any information to the sharing pane, so all it did was offer to share a screen shot. Today, let's provide some data.

This is a continuation of the interop pattern. Repeating the table we had from last time:

XxxStatics	XxxInterop
GetForCurrentView	GetForWindow
DoSomething (implied "for current view")	DoSomethingForWindow

Last time, we used the second case, converting `ShowSharingUI` to `ShowSharingUIFor-Window`. Today we're going to use the first case: Converting `GetForCurrentView` to `Get-ForWindow`.

Start with a blank scratch program and make these changes. (Remember, Little Programs do little to no error checking.)

```
#include <wrl/client.h>
#include <wrl/event.h>
#include <wrl/wrappers/corewrappers.h>
#include <windows.applicationmodel.datatransfer.h>
#include <shlobj.h> // IDataTransferManagerInterop
#include <tchar.h> // Huh? Why are you still using ANSI?
#include <roapi.h>

namespace WRL = Microsoft::WRL;
namespace awf = ABI::Windows::Foundation;
namespace dt = ABI::Windows::ApplicationModel::DataTransfer;

using Microsoft::WRL::Wrappers::HStringReference;

WRL::ComPtr<IDataTransferManagerInterop> g_dtmInterop;
WRL::ComPtr<DT::IDataTransferManager> g_dtm;
EventRegistrationToken g_dataRequestedToken;
```

Note that in real life, these global variables would be instance variables of some C++ class.

```
BOOL
OnCreate(HWND hwnd, LPCREATESTRUCT lpcs)
{
  RoGetActivationFactory(HStringReference(
    RuntimeClass_Windows_ApplicationModel_DataTransfer_DataTransferManager)
                         .Get(), IID_PPV_ARGS(&g_dtmInterop));

  g_dtmInterop->GetForWindow(hwnd, IID_PPV_ARGS(&g_dtm));

  auto callback = WRL::Callback<WF::ITypedEventHandler<
    DT::DataTransferManager*, DT::DataRequestedEventArgs*>>(
        [](auto&&, DT::IDataRequestedEventArgs* e)
    {
      WRL::ComPtr<DT::IDataRequest> request;
      e->get_Request(&request);

      WRL::ComPtr<DT::IDataPackage> data;
      request->get_Data(&data);

      WRL::ComPtr<DT::IDataPackagePropertySet> properties;
      data->get_Properties(&properties);

      // Title is mandatory
      properties->put_Title(HStringReference(L"Title from Win32").Get());

      // Description is optional
      properties->put_Description(HStringReference(L"This text came from a Win32
app").Get());

      data->SetText(HStringReference(L"Text from Win32 app!").Get());

      return S_OK;
    });

  g_dtm->add_DataRequested(callback.Get(), &g_dataRequestedToken);

  return TRUE;
}

void
OnDestroy(HWND hwnd)
{
    g_dtm->remove_DataRequested(g_dataRequestedToken);
    g_dtm.Reset();
    g_dtmInterop.Reset();
    PostQuitMessage(0);
}


void OnChar(HWND hwnd, TCHAR ch, int cRepeat)
{
  switch (ch) {
```

```
  case TEXT(' '):
    g_dtmInterop->ShowShareUIForWindow(hwnd);
    }
    break;
  }
}

HANDLE_MSG(hwnd, WM_CHAR, OnChar);
```

Okay, let's see what happened here.

When the window is created, we get the interop interface and save it in the global variable for later use. We then call `GetForWindow` to obtain the `DataTransferManager` for our window. In WinRT this would have been a call to `GetForCurrentView`.

That's all for the interop part of this exercise. Everything else is just operating on the WinRT objects at the ABI level instead of at the projection level.

Next we create a callback handler for the `DataRequested` event. We'll look at the body of the handler later.

We then register the handler for the event by calling `add_DataRequested` and save the registration token so we can unregister later.

Okay, now to look inside the callback: This is a direct translation of `DataTransferManager` from projection back into ABI. Reading a property becomes a call to the `get_PropertyName` method, and writing a property becomes a call to the `put_PropertyName` method. In our case, we take the `DataRequestedEventArgs` and get the `Request` property, which is an `IDataRequest`. From the `IDataRequest` we set the `Title` and `Description` properties, and use the `SetText` method to provide the text that we are sharing.

At destruction, we unregister the event and release the objects.

The final snippet of code is what we saw last time: When the user hits the space bar, open the share pane. But this time, the share pane actually shows something interesting, because our `DataRequested` event handler provides text to be shared.

Of course, in a real program, you would presumably offer text or other content that is based on the current state or selection rather than just spitting out hard-coded content, but this is just a Little Program.

Raymond Chen

Follow