Helper functions to make shell bind contexts slightly more manageable

devblogs.microsoft.com/oldnewthing/20150123-00

January 23, 2015

Raymond Chen

Last time, we learned about the wonderful world of shell bind context strings, and I promised some helper functions to make this slightly more manageable.

Here are some helper functions which supplement the `CreateBindCtxWithOpts` function we created some time ago.

```
#include <propsys.h>

HRESULT EnsureBindCtxPropertyBag(
    IBindCtx *pbc, REFIID riid, void **ppv)
{
 *ppv = nullptr;
 CComPtr<IUnknown> spunk;
 HRESULT hr = pbc->GetObjectParam(STR_PROPERTYBAG_PARAM, &spunk);
 if (FAILED(hr)) {
  hr = PSCreateMemoryPropertyStore(IID_PPV_ARGS(&spunk));
  if (SUCCEEDED(hr)) {
   hr = pbc->RegisterObjectParam(STR_PROPERTYBAG_PARAM, spunk);
  }
 }
 if (SUCCEEDED(hr)) {
  hr = spunk->QueryInterface(riid, ppv);
 }
  return hr;
}

HRESULT AddBindCtxDWORD(
    IBindCtx *pbc, LPCWSTR pszName, DWORD dwValue)
{
 CComPtr<IPropertyBag> sppb;
 HRESULT hr = EnsureBindCtxPropertyBag(pbc, IID_PPV_ARGS(&sppb));
 if (SUCCEEDED(hr)) {
  hr = PSPropertyBag_WriteDWORD(sppb, pszName, dwValue);
 }
 return hr;
}

HRESULT AddBindCtxString(
    IBindCtx *pbc, LPCWSTR pszName, LPCWSTR pszValue)
{
 CComPtr<IPropertyBag> sppb;
 HRESULT hr = EnsureBindCtxPropertyBag(pbc, IID_PPV_ARGS(&sppb));
 if (SUCCEEDED(hr)) {
  hr = PSPropertyBag_WriteStr(sppb, pszName, pszValue);
 }
 return hr;
}

HRESULT CreateDwordBindCtx(
    LPCWSTR pszName, DWORD dwValue, IBindCtx **ppbc)
{
 CComPtr<IBindCtx> spbc;
 HRESULT hr = CreateBindCtx(0, &spbc);
 if (SUCCEEDED(hr)) {
  hr = AddBindCtxDWORD(spbc, pszName, dwValue);
 }
 *ppbc = SUCCEEDED(hr) ? spbc.Detach() : nullptr;
 return hr;
```

```
}

HRESULT CreateStringBindCtx(
    LPCWSTR pszName, LPCWSTR pszValue, IBindCtx **ppbc)
{
 CComPtr<IBindCtx> spbc;
 HRESULT hr = CreateBindCtx(0, &spbc);
 if (SUCCEEDED(hr)) {
  hr = AddBindCtxString(spbc, pszName, pszValue);
 }
 *ppbc = SUCCEEDED(hr) ? spbc.Detach() : nullptr;
 return hr;
}
```

The `EnsureBindCtxPropertyBag` function puts a property bag in the bind context if there isn't one already.

The `AddBindCtxDWORD` function adds a `DWORD` to that associated property bag. If you wanted to add multiple `DWORD` s to a bind context, you would call this function multiple times. You can also use the `AddBindCtxString` if the thing you want to add is a string.

The `CreateDwordBindCtx` function handles the simple case where you want to create a bind context that contains a single `DWORD` . Similarly, `CreateStringBindCtx` .

But now things are starting to get kind of unwieldy. What if you want a bind context with a string and a `DWORD` ? Let's go for something a bit more fluent.

But first, some scaffolding.

```
class CStaticUnknown : public IUnknown
{
public:
 // *** IUnknown ***
 IFACEMETHODIMP QueryInterface(
  _In_ REFIID riid, _Outptr_ void **ppv)
 {
  *ppv = nullptr;
  HRESULT hr = E_NOINTERFACE;
  if (riid == IID_IUnknown) {
   *ppv = static_cast<IUnknown *>(this);
   AddRef();
   hr = S_OK;
  }
  return hr;
 }

 IFACEMETHODIMP_(ULONG) AddRef()
 {
  return 2;
 }

 IFACEMETHODIMP_(ULONG) Release()
 {
  return 1;
 }

};

CStaticUnknown s_unkStatic;
```

This static implementation of `IUnknown` is one we'll use for the bind context strings whose mere presence indicates that a flag is set.

```
class CBindCtxBuilder
{
public:
 CBindCtxBuilder()
 {
  m_hrCumulative = CreateBindCtx(0, &m_spbc);
 }

 CBindCtxBuilder& SetMode(DWORD grfMode);
 CBindCtxBuilder& SetFindData(const WIN32_FIND_DATA *pfd);
 CBindCtxBuilder& SetFlag(PCWSTR pszName);
 CBindCtxBuilder& SetVariantDword(PCWSTR pszName, DWORD dwValue);
 CBindCtxBuilder& SetVariantString(PCWSTR pszName, PCWSTR pszValue);

 HRESULT Result() const { return m_hrCumulative; }

 IBindCtx *GetBindCtx() const
 { return SUCCEEDED(m_hrCumulative) ? m_spbc : nullptr; }
private:
 HRESULT EnsurePropertyBag();

private:
 CComPtr<IBindCtx> m_spbc;
 CComPtr<IPropertyBag> m_sppb;
 HRESULT m_hrCumulative;
};
```

The bind context builder class is a helper class that creates a bind context, and then fills it with stuff. For now, we let you set the following:

- The mode to use for opening the target of the bind. The default is `STGM_READWRITE`.
- The find data to use, if creating a simple pidl.
- An arbitrary flag, associated with a dummy `IUnknown`.
- A `DWORD` in the property bag.
- A string in the property bag.

After you build up the bind context, you can check the `Result()` to see if it was built successfully, and use `GetBindCtx` to extract the result.

Here's the implementation. It's really not that exciting. We accumulate any error in `m_hrCumulative`, and once an error occurs, all future methods do nothing aside from preserving the error. To make the object fluent, the methods return a reference to themselves.

There is a special bind context method for setting the mode:

```
CBindCtxBuilder&
CBindCtxBuilder::SetMode(DWORD grfMode)
{
 if (SUCCEEDED(m_hrCumulative)) {
  BIND_OPTS bo = { sizeof(bo), 0, grfMode, 0 };
  m_hrCumulative = m_spbc->SetBindOptions(&bo);
 }
 return *this;
}
```

Find data is set as a direct object on the bind context, as we saw some time ago:

```
CBindCtxBuilder&
CBindCtxBuilder::SetFindData(const WIN32_FIND_DATA *pfd)
{
 if (SUCCEEDED(m_hrCumulative)) {
  m_hrCumulative = AddFileSysBindCtx(m_spbc, pfd);
 }
 return *this;
}
```

Flags are set by there mere presence, so we associate them with a dummy `IUnknown` that does nothing:

```
CBindCtxBuilder&
CBindCtxBuilder::SetFlag(PCWSTR pszName)
{
 if (SUCCEEDED(m_hrCumulative)) {
  m_hrCumulative = m_spbc->RegisterObjectParam(
    const_cast<PWSTR>(pszName), &s_unkStatic);
 }
 return *this;
}
```

If a property is set in the property bag, we need to proceed in two steps. First, we create the property bag if we don't have one already. Second, we put the value into the property bag:

```
CBindCtxBuilder&
CBindCtxBuilder::SetVariantDword(
    PCWSTR pszName, DWORD dwValue)
{
 if (SUCCEEDED(m_hrCumulative)) {
  m_hrCumulative = EnsurePropertyBag();
 }
 if (SUCCEEDED(m_hrCumulative)) {
  m_hrCumulative =  PSPropertyBag_WriteDWORD(
    m_sppb, pszName, dwValue);
 }
 return *this;
}

CBindCtxBuilder&
CBindCtxBuilder::SetVariantString(
    PCWSTR pszName, PCWSTR pszValue)
{
 if (SUCCEEDED(m_hrCumulative)) {
  m_hrCumulative = EnsurePropertyBag();
 }
 if (SUCCEEDED(m_hrCumulative)) {
  m_hrCumulative =  PSPropertyBag_WriteStr(
    m_sppb, pszName, pszValue);
 }
 return *this;
}
```

And finally, the helper function that creates a property bag if we don't have one already.

```
HRESULT CBindCtxBuilder::EnsurePropertyBag()
{
 HRESULT hr = S_OK;
 if (!m_sppb) {
  hr = PSCreateMemoryPropertyStore(
    IID_PPV_ARGS(&m_sppb));
  if (SUCCEEDED(hr)) {
   hr = m_spbc->RegisterObjectParam(
    STR_PROPERTYBAG_PARAM, m_sppb);
  }
 }
 return hr;
}
```

The idea here is that the class is used like this:

```
CBindCtxBuilder builder;
builder.SetMode(STGM_CREATE)
       .SetFindData(&wfd)
       .SetFlag(STR_FILE_SYS_BIND_DATA_WIN7_FORMAT)
       .SetFlag(STR_BIND_FOLDERS_READ_ONLY);
hr = builder.Result();
if (SUCCEEDED(hr)) {
 hr = psf->ParseDisplayName(hwnd, builder.GetBindCtx(),
   pszName, &cchEaten, &pidl, &dwAttributes);
}
```

You create the bind context builder, then use the various `SetXxx` methods to fill the bind context with goodies, and then you check if it all worked okay. If so, then you use `GetBind-Ctx` to get the resulting bind context and proceed on your way.

Raymond Chen

Follow