How can I break down a shell item in the same way as the breadcrumb bar?

devblogs.microsoft.com/oldnewthing/20210827-00

August 27, 2021

Raymond Chen

Suppose you have a shell item in the form of an `IShellItem`. How can you produce the folders that lead up to the shell item?

One idea is to start from the item you have in your hand and keep trying to get its parent item, until you run out of parents.

Let's try that.

My smart pointer library will be (rolls dice) C++/WinRT, with wil for string management.

```
#include <winrt/base.h>
#include <wil/resource.h>

winrt::com_ptr<IShellItem> GetShellItemFromUser()
{
    auto open = winrt::create_instance<IFileOpenDialog>(CLSID_FileOpenDialog);
    if (FAILED(open->Show(nullptr)))
    {
        return {};
    }

    winrt::com_ptr<IShellItem> item;
    winrt::check_hresult(open->GetResult(item.put()));
    return item;
}
```

We start with this helper function that produces an `IShellItem`. We'll do it by just letting the user pick one.

We can use this in our main program:

```
int main(int, char**)
{
    winrt::init_apartment(winrt::apartment_type::single_threaded);
    auto item = GetShellItemFromUser();

    while (item) {
        wil::unique_cotaskmem_string str;
        winrt::check_hresult(item->GetDisplayName(SIGDN_PARENTRELATIVEFORUI, &str));
        printf("[ %ls ] ", str.get());

        winrt::com_ptr<IShellItem> parent;
        item->GetParent(parent.put());
        item = parent;
    }

    return 0;
}
```

If the user picks an item, then we go into a loop that walks up toward the parent. At each step, we get its normal display name and print it, just to do something interesting. To go to the next step, we ask for the parent item, and loop back if we got one.

But since we are walking from the item to its parent, we generate the list bottom-up. For example:

```
[ Image ]
[ Scanned Documents ]
[ Documents ]
[ This PC ]
[ Desktop ]
```

Another way to generate the items is to start at the root and walk toward the item. In this case, we can take the ID list and use larger and larger prefixes.

```
winrt::com_ptr<IShellItem>
ItemFromPartialPidl(PCIDLIST_ABSOLUTE begin, PIDLIST_ABSOLUTE end)
{
    auto restore = wil::scope_exit([end, prev = std::exchange(end->mkid.cb, {})]
        { end->mkid.cb = prev; });
    return winrt::capture<IShellItem>(SHCreateItemFromIDList, begin);
}
```

This helper function takes a portion of an ID list (from `begin` up to but not including `end`) and creates a shell item from it. We do this by temporarily setting the `cb` to 0, causing to serve as a null terminator, and then asking `SHCreateItemFromIDList` to create an `IShellItem` from it. The call to `capture` expands to

```
winrt::com_ptr<IShellItem> item;
winrt::check_hresult(SHCreateItemFromIDList(begin, IID_PPV_ARGS(&item)));
return item;
```

Meanwhile, the `scope_exit` ensures that the original value is restored, even if the `SHCreateItemFromIDList` fails.

We can use this helper function to generate all the prefixes.

```
int main(int, char**)
{
    winrt::init_apartment(winrt::apartment_type::single_threaded);
    auto item = GetShellItemFromUser();

    if (item) {
        wil::unique_cotaskmem_ptr<ITEMIDLIST_ABSOLUTE> pidlFull;
        item.as<IPersistIDList>()->GetIDList(wil::out_param(pidlFull));
        PIDLIST_ABSOLUTE pidlNext = pidlFull.get();
        for (;;)
        {
            item = ItemFromPartialPidl(pidlFull.get(), pidlNext);
            wil::unique_cotaskmem_string str;
            winrt::check_hresult(item->GetDisplayName(SIGDN_PARENTRELATIVEFORUI,
&str));
            printf("[ %ls ]\n", str.get());
            if (ILIsEmpty(pidlNext)) break;
            pidlNext = ILNext(pidlNext);
        }
    }

    return 0;
}
```

From the item, we obtain the ID list by calling `IPersistIDList::GetIDList`. We then walk through the ID list one item at a time, producing the `IShellItem` for each partial prefix, stopping when we have nothing more to add.

This version generates the items starting from the root:

```
[ Desktop ]
[ This PC ]
[ Documents ]
[ Scanned Documents ]
[ Image ]
```

The last detail is that we used `SIGDN_PARENTRELATIVEFORUI`, which is a display name format that is suitable for use in the breadcrumb bar.

If you use this to generate your own breadcrumb bar, you can respond to a click on an element by navigating to the shell item that generated that element. You'll probably want to save the `IShellItem` as reference data associated with the element. That'll be easier than trying to regenerate it from scratch in response to the click.

Raymond Chen

Follow