

After importing a TLB, how do I convert from one type of `_com_ptr_t` to another?

 devblogs.microsoft.com/oldnewthing/20221228-00

December 28, 2022



Raymond Chen

The Microsoft C++ compiler has [this weird feature called `#import`](#). You give it a type library (TLB) file, and it converts the type library into a C++ header file that exposes the types in the type library in a form of bunch of smart pointer types.

The documentation on how to use these autogenerated smart pointers is [kind of sparse](#). One thing it neglects to mention is how to convert one kind of `_com_ptr_t` to another. Suppose you have an `IWidgetPtr` (representing an `IWidget`) and you want to get the `IToggleSwitchPtr` interface from it.

Well, we see that [there is a `QueryInterface` method](#), but it appears to produce raw pointers, not fancy `_com_ptr_t` pointers.

Aha, but we can ask for the raw pointer version of a `_com_ptr_t`: [Use the `&` operator](#).

```
void FlipToggleSwitch(IWidgetPtr widget)
{
    IToggleSwitchPtr switch;
    HRESULT hr = widget.QueryInterface(__uuidof(switch), &switch);
    if (FAILED(hr)) _com_raise_error(hr);
    switch.Flip();
}
```

The parameters to the `QueryInterface` can be simplified (and made less error-prone) with the help of the `IID_PPV_ARGS` macro:

```
HRESULT hr = widget.QueryInterface(IID_PPV_ARGS(&switch));
```

There's another way, which is even easier, but also even more invisible: Use [the conversion constructor](#).

```
void FlipToggleSwitch(IWidgetPtr widget)
{
    IToggleSwitchPtr switch = widget; // conversion constructor!
    if (!switch) _com_raise_error(E_NOINTERFACE);
    switch.Flip();
}
```

The conversion constructor for `_com_ptr_t` lets you construct any type of `_com_ptr_t` from any other type of `_com_ptr_t`. It does so by using `QueryInterface` to get from one interface to the other. If the source is empty (wraps a `nullptr`) or if the query fails with `E_NOINTERFACE`, then the constructed object is empty. If the query fails with any other error, then a `_com_error` exception is thrown.¹

This conversion operator is not marked `explicit`, so it can be implicitly invoked. This makes it super-convenient, but also super-eager to stick its nose in places it might not belong.

```
void SetInvertedPolarity(IWidgetPtr widget);

void PrepareGadget(IGadgetPtr gadget)
{
    SetInvertedPolarity(gadget); // compiles!
}
```

We are passing the wrong kind of smart pointer to `SetInvertedPolarity`, but it compiles anyway! The compiler automatically converts the `IGadgetPtr` to an `IWidgetPtr` in order to pass it to `SetInvertedPolarity`. This is great if gadgets are deluxe versions of widgets, and you expect the conversion to succeed. This is not so great if you didn't mean to treat the gadget as a widget, and the attempt to call `SetInvertedPolarity` was really a mistake. Depending on how the `SetInvertedPolarity` function works, it may throw an exception or even crash on a null pointer if you give it a null widget. The super-eager conversion constructor led you into a trap.

The inner workings of the `_com_ptr_t` can be found in the header file `comip.h`.

¹ If you set a custom COM error handler, then the custom error handler is called.

Raymond Chen

Follow

