How well does wil com_ptr support class template argument deduction (CTAD)?

devblogs.microsoft.com/oldnewthing/20240318-30

March 18, 2024

Raymond Chen

Continuing our investigation of which C++ COM wrappers support class template argument deduction (CTAD), next up is the Windows Implementation Library's `wil::com_ptr`.

The wil `com_ptr` fails CTAD through no fault of its own.

```
template<typename T, typename err_policy = err_exception_policy>
class com_ptr_t
{
public:
    using pointer = T*;

    com_ptr_t(pointer ptr) WI_NOEXCEPT; // this constructor

    [[ other stuff ]]
};

template<typename T>
using com_ptr = com_ptr_t<T, err_exception_policy>;
```

The problem is that `wil::com_ptr` is an alias template, and in C++17, alias templates could not participate in CTAD. This hole was filled in by C++20, so at least in C++20, you can say

```
Widget* p;
auto smart = wil::com_ptr(p); // requires C++20
```

If you are stuck with C++17, then deduction guides won't help, since C++17 disallows them on alias templates. (Seeing as alias templates couldn't participate in CTAD in the first place, there's no point letting you specify a deduction guide for something that canot be deduced in the first place.)

Therefore, if you can't use C++20, you'll have to make do with a maker function.

```
template<typename T>
wil::com_ptr<T> make_com_ptr(T* p)
{
    return p;
}

template<typename T>
wil::com_ptr_nothrow<T> make_com_ptr_nothrow(T* p)
{
    return p;
}
template<typename T>
wil::com_ptr_failfast<T> make_com_ptr_failfast(T* p)
{
    return p;
}
```

Fortunately, wil comes with these maker functions already, so everything is ready for you.

Next time, we'll wrap up this mini-series on C++ COM wrappers and CTAD by looking at C++/WinRT.