# On the overloading of the address-of operator & in smart pointer classes

devblogs.microsoft.com/oldnewthing/20221010-00

October 10, 2022

Raymond Chen

Many smart pointer classes overload the address-of operator `&` to give you access to the inner raw pointer.

Unfortunately, they disagree on what happens to the object being managed by the smart pointer before you get its raw address.

| Library | Existing contents |
|---|---|
| _com_ptr_t | Released |
| ATL (CComPtr) | Must be empty<br>(will assert in Debug) |
| MFC (IPTR) | Released |
| WRL (ComPtr) | Released |
| wil (com_ptr) | Released |
| C++/WinRT (com_ptr) | N/A |

C++/WinRT avoids the confusion by simply not having an overloaded `operator&` at all! Not having an overloaded `operator&` also makes it easier to take the address of the smart pointer itself. The `put()` method Releases any managed COM pointer and then returns the address of the raw pointer.

So let's finish the table. Let's say that `sp` is the name of a variable of the corresponding smart pointer type.

| Library | Release | Don't release | Assumes empty |
|---|---|---|---|

| | | | |
|---|---|---|---|
| _com_ptr_t | `&sp` | `&sp.GetInterfacePtr()` | |
| ATL (CComPtr) | | `&sp.p` | `&sp` |
| MFC (IPTR) | `&sp` | | |
| WRL (ComPtr) | `&sp`<br>`p.ReleaseAndGetAddressOf()` | `p.GetAddressOf()` | |
| wil (com_ptr) | `&sp`<br>`sp.put()` | `sp.addressof()` | |
| C++/WinRT (com_ptr) | `sp.put()` | | |

**Bonus chatter**: The possibility of an overloaded `operator&` is one of those special cases you tend to forget about when writing template library code.[1] In general, it's not safe to use the `&` operator to get the address of an object of unknown type, because the operator might be overloaded. You have to use `std::addressof`.

[1] Hey, at least it's not an overloaded comma operator. That thing is *nasty*.

Raymond Chen

**Follow**