How can my C++/WinRT component pass a std::vector back to the caller?

devblogs.microsoft.com/oldnewthing/20211130-00

November 30, 2021

Raymond Chen

The ReceiveArray pattern is the Windows Runtime pattern for how a function can return a C-style conformant array to its caller. In C++/WinRT, the projected version of the function is

```
// [out] parameter
void M(com_array<T>& value);

// return value
com_array<T> M();
```

A customer had a method that generated the result into a `std::vector<int>`. How do you return this to the caller? "There is no move constructor for `com_array<int>` that takes a `std::vector<int>`."

That's right, there is no constructor for `com_array<int>` that takes a `std::vector<int>&&`. If you think about it, there can't possibly be one.

The `com_array` is required to use the COM task allocator to allocate its memory, because the memory is going to be passed back to the calling component, and the calling component is responsible for freeing the memory. This means that the memory allocator must be something language-agnostic, since the caller could be written in C# or Visual Basic or JavaScript or Rust or whatever.

On the other hand, `std::vector` uses the C++ free store to allocate memory.[1] This is an allocator specific to the C++ language. Actually, it's even worse. It's an allocator that is specific to a particular *implementation* of the C++ language. Code that uses one version of the compiler and runtime library cannot interoperate with code that uses a different version of the compiler and runtime library.

The allocators don't agree, so you won't be able to transfer ownership: The memory was allocated from some version of the C++ free store, but putting it in a `com_array` will result in the memory being freed by `CoTaskMemFree`.

Okay, so you can't move it into a `com_array`, but can you copy it?

Yes, and the `com_array` even has a special constructor for copying from a `std::vector`.

```
std::vector<int32_t> m_indices;

com_array<int32_t> GetIndices()
{
    return com_array<int32_t>(m_indices);
}
```

Starting in C++/WinRT version 2.0.200601.2, there's a deduction guide that that deduces `com_array<T>` if you construct from a `std::vector<T>`, so you need only write

```
com_array<int32_t> GetIndices()
{
    return com_array(m_indices);
}
```

If you aren't wedded to `std::vector`, you could generate the results directly into a `winrt::com_array` and return it, thereby avoiding a copy.

[1] You can override this by providing a custom allocator, say, by a custom allocator that obtains memory via `CoTaskMemAlloc`. However, library types with custom allocators are going to create interop friction with other code that expects library types with the standard allocator.

Raymond Chen

Follow