On the various ways of constructing a C++/WinRT com_array

devblogs.microsoft.com/oldnewthing/20200522-00

May 22, 2020

Raymond Chen

The C++/WinRT `com_array<T>` represents a C-style conformant array of data where the underlying buffer is allocated via the COM task allocator. It is typically used to represent a C-style conformant array which is allocated by one component and freed by another.

You will probably need to make one of these things when you are returning a projected array to the caller, either as the return value or through an output parameter. Here are your choices of constructor, with names that I made up.

`com_array();`	(1)
`com_array(uint32_t count);`	(2)
`com_array(uint32_t count, T const& value);`	(3)

`template<typename InIt>`
`com_array(InIt first, InIt last)`
	(4)

`com_array(std::vector<T> const& value)`	(5)

`template<size_t N>`
`com_array(std::array<T, N> const& value)`
	(6)

`template<size_t N>`
`com_array(T const(&value)[N])`
	(7)

`com_array(std::initializer_list<T> value)`	(8)

`com_array(void* ptr, uint32_t count,`
`  take_ownership_from_abi_t);`
	(9)

`com_array(com_array&& other)`	(10)

1) Default constructor: Creates an empty buffer.

2) Capacity constructor (default value): Creates a buffer of `count` elements, all containing copies of a default-constructed `T`.

3) Capacity constructor (explicit value): Creates a buffer of `count` elements, each of which is a copy of the provided value.

4) Range constructor: Creates a buffer that is a copy of the range `[first, last)`.

5) Vector constructor: Creates a buffer that is a copy of the contents of the vector.

6) Array constructor: Creates a buffer that is a copy of the contents of the array.

7) C-style array constructor: Creates a buffer that is a copy of the contents of the C-style array.

8) Initializer-list constructor: Creates a buffer that is a copy of the contents of the initializer list.

9) ABI constructor: Takes ownership of a buffer of specified length.

10) Move constructor: Moves the resources from another `com_array` of the same type, leaving the original empty.

Remarks for capacity constructor with default value (2)

Constructor (2) is almost but not quite the same as creating a buffer of `count` elements each of which is a default-constructed `T`. Consider:

```
auto players = com_array<MediaPlayer>(50);
```

The `MediaPlayer` object's default constructor creates a reference to a new media player object, and its copy constructor copies the reference. Therefore, the above line of code creates an array of 50 references to the same media player object, not an array of 50 different media player objects.

Bonus weirdness: If you pass a `count` of zero, the `com_array` will still default-contruct a `T`, even though it doesn't use it for anything.

Remarks for capacity constructor with explicit value (3)

`com_array(2, 42)` is interpreted as an attempt to use the range constructor (4), which fails because `2` and `42` are not iterators. To get this to be interpreted as a capacity constructor with explicit `int32_t` value, use an explicitly unsigned integer as the first parameter: `com_array(2u, 42)`.

Remarks for range constructor (4)

Sadly, there is (as of this writing)[1] no deduction guide for the range constructor (4), so you will have to state the underlying type `T` explicitly:

```
auto a = com_array<T>(source.begin(), source.end());
```

Bonus trick: If you want to move the range rather than copy it, use the `std:: move_ iterator` iterator adaptor:

```
auto a = com_array<T>(std::move_iterator(source.begin()),
                      std::move_iterator(source.end()));
```

Remarks for vector (5), array (6), and C-style array (7) constructors

For constructors (5) through (7), the contents of the container are copied. You can use the range constructor (4) with the `move_ iterator` iterator adaptor to move the contents into the `com_ array` instead of copying.

Remarks for ABI constructor (9)

The ABI constructor (9) is the lowest-level constructor. Use it when you have a block of memory already allocated via `CoTaskMemAlloc` and you want the `com_array` to assume responsibility for it. To emphasize the special requirements for this constructor, the final parameter must be `take_ ownership_ from_ abi`.

[1] Hint hint. Add a deduction guide and create a PR. While you're at it, fix the range constructor so it doesn't inadvertently trigger for `com_array(2, 42)`.

Raymond Chen

Follow