How do I save a C++/WinRT array_view as a com_array?

devblogs.microsoft.com/oldnewthing/20201106-00

November 6, 2020

Raymond Chen

If you have a Windows Runtime class with a property whose Windows Runtime type is an array, then the C++/WinRT projection expresses the property setter and getter as follows:

```
// Set array
void IntArray(winrt::array_view<int32_t const> const& value);

// Get array
winrt::com_array<int32_t> IntArray();
```

We saw earlier that these correspond to the `PassArray` and `ReceiveArray` patterns, respectively.

How would you implement the backing store for this property?

Well, the first thing to note is that the backing store should *not* be an `array_view`, because an `array_view` is a non-owning view into somebody else's data. If your setter saved just the `array_view`, then it would be left with a dangling pointer, because the `array_view` parameter is valid only for the duration of the call.

You realize that what you want to do is save a copy of the contents of an `array_view`. One option is to save it in a `com_array`, but there is no obvious way to create a `com_array` that is a copy of the contents of an `array_view`, seeing as there's no constructor that takes an `array_view`.

You need to use the two-iterator constructor for `com_array`. This creates a copy of the provided range of data and saves it in a `com_array`.

Similarly, to return the `com_array`, you need to construct the return value in the same way. The `com_array` is not copyable, so you'll have to use the two-iterator constructor.

```
struct Class : ClassT<Class>
{
private:
  winrt::com_array<int32_t> int_array_;

public:
  void IntArray(winrt::array_view<int32_t const> const& value)
  {
    int_array_ = { value.begin(), value.end() };
  }

  auto IntArray()
  {
    return winrt::com_array
      { int_array_.begin(), int_array_.end() };
  }
};
```

We take advantage of class template argument deduction (CTAD) to avoid having to repeat the type `int32_t` when constructing the `com_array`. The full version would have been

```
    return winrt::com_array<int32_t>
      { int_array_.begin(), int_array_.end() };
```

The `com_array` is not copyable, but it *is* movable, so if you want to just give it away, you can `std::move` it. You don't want to do that for a property backing store, but maybe it'll come in handy in other cases.

If you intend to do something with the backing store beyond simply using it to hold data, you might want to use a more versatile data structure like a `std::vector`. Fortunately, a `com_array` can construct from a vector, so you can do this:

```
struct Class : ClassT<Class>
{
private:
  std::vector<int32_t> int_vector_;

public:
  void IntArray(winrt::array_view<int32_t const> const& value)
  {
    int_vector_ = { value.begin(), value.end() };
  }

  auto IntArray()
  {
    return winrt::com_array{ int_vector_ };
  }
};
```

Here, we take advantage of a deduction guide (introduced in version 2.0.200601.2) to avoid having to specialize the `winrt::com_array`.

Raymond Chen

Follow