I declared my Windows Runtime method as accepting an array by reference, but it always arrives empty

devblogs.microsoft.com/oldnewthing/20210903-00

September 3, 2021

Raymond Chen

A customer was writing a Windows Runtime component that took as one of its parameters an array of strings.

```
namespace Contoso
{
    runtimeclass Widget
    {
        Widget();
        void SetMessages(ref String[] messages);
    }
}
```

They tried to pass an array of strings from C#:

```
    Widget widget = new Widget();
    string[] messages = new string[] { "testing", "is this thing on?" };
    widget.SetMessages(messages);
```

The class implementation was written in C++/WinRT, but they found that when their implementation received the array, it consisted of null strings!

```
void Widget::SetMessages(array_view<hstring> messages)
{
    // Debugger says that messages.size() == 2, as expected
    // but all the elements are nullptr, not the original strings
}
```

The customer chased this all the way back to the ABI boundary. With a breakpoint at the C++/WinRT ABI boundary glue code, they found that the array was empty as soon as it arrived from C#:

```
    int32_t __stdcall SetMessages(uint32_t __messagesSize, void** messages)
    noexcept final try
    {
```

```
    zero_abi<hstring>(messages, __messagesSize);
    typename D::abi_guard guard(this->shim());
    this->shim().SetMessages(array_view<hstring>(
        reinterpret_cast<hstring*>(messages),
        reinterpret_cast<hstring*>(messages) + __messagesSize));
    return 0;
  }
  catch (...) { return to_hresult(); }
```

At the breakpoint, which is right at the ABI boundary, the `__messagesSize` has the expected value of 2, but the `messages` array holds two null pointers rather than the desired strings.

What's going on here? What's the correct way to pass an array from C# to a Windows Runtime component written in C++/WinRT? Am I missing something in my declaration of the array parameter?

The problem isn't that you're missing something. The problem is that you have *too much*.

Take a closer look at the C++/WinRT auto-generated code. The first thing that happens at the ABI boundary is `zero_abi(messages, __messageSize)` : It's setting the incoming array to zeros! So even if C# passed us an array filled with good strings, the first thing we do is throw them away.

Something is seriously messed up here.

The problem is the `ref` keyword.

In MIDL3, the `ref` keyword on an array means that the array is allocated by the caller, but filled by the method. It is for passing data *out* of a method, not *into* it.

Let's take another look at our table of the various ways of passing C-style arrays across the Windows Runtime ABI boundary, specifically the line for IDL:

	PassArray	FillArray	ReceiveArray	
			Parameter	Return value
IDL	`void M(T[] value);`	`void M(ref T[] value);`	`void M(out T[] value);`	`T[] M();`

PassArray is the case where the caller passes an array to the method. The method can read the array but cannot modify it. Use this when the caller wants to provide information to the method.

FillArray is the case where the caller passes an array to the method. The method is expected to produce exactly enough elements to fill the array, and when the method returns, the caller uses those values. Use this when the caller wants to receive information of a known size.

ReceiveArray is the case where the method creates the array and returns it to the caller. Use this when the caller wants to receive an array, but only the method knows how big the array is going to be.

In the customer's case, they want a *PassArray*, so the solution is to remove the `ref` keyword and declare it as a plain array:

```
namespace Contoso
{
    runtimeclass Widget
    {
        Widget();
        void SetMessages(String[] messages);
    }
}
```

Bonus chatter: It's rather unfortunate that the `ref` keyword doesn't mean quite the same thing in MIDL as it does in C#. In MIDL, `ref` means "Write to the caller-provided array." But in C#, `ref` means "The caller created an array for you, and you can choose to read from it, write to it, or even replace it with another array object entirely."

I suspect the choice of `ref` was constrained by the list of existing MIDL keywords, to avoid breaking existing IDL files. Maybe they could have used `set`? Would this have made sense?

```
        void SetMessages(set String[] messages);
```

Or maybe it would have been just as confusing. I don't know.

No point second-guessing it now. What's done is done.

Raymond Chen

Follow