I created an overloaded operator for my C++/WinRT class, but it's not working

devblogs.microsoft.com/oldnewthing/20231012-00

October 12, 2023

Raymond Chen

A customer added an operator overload to their C++/WinRT class:

```
namespace winrt::Contoso::implementation
{
    struct StringList : StringListT<StringList>
    {
        int32_t Size() { return size(); }

        hstring operator[](int32_t index) {
            return string_at(index);
        }
    };
}

namespace winrt::Contoso::factory_implementation
{
    struct StringList : StringListT<StringList, implementation::StringList>
    {
    };
}
```

The customer wants their `StringList` to act like an array of strings, except that the values are generated on the fly. To carry out the simulation, they added an overloaded `[]` operator so that it can be indexed like an array.

But they found that even though they overloaded the `[]` operator, they weren't able to use the `[]` syntax from their C# app that consumed this class.

```
StringList list = new StringList();
for (int i = 0; i < list.Size; i++) {
    System.Console.WriteLine(list[i]); // doesn't compile
}
```

Why doesn't this work?

The overloaded `[]` operator was defined on the implementation class, which is not exposed to outsiders. All outsiders see is the projection, which is the class defined in the IDL, and the IDL file doesn't have an overload of `[]`:

```
namespace Contoso
{
    runtimeclass StringList
    {
        StringList();
        Int32 Size { get; };
        // no operator[] defined here
    }
}
```

The customer didn't share their IDL file, but I'm pretty confident it doesn't have an overload of `operator[]`. The reason is that Windows Runtime IDL doesn't support operator overloading; there is no syntax for it.

The Windows Runtime doesn't support operator overloading because not all languages support it, and the Windows Runtime tries to be usable by a wide range of programming languages.

However, all is not lost.

This particular class is modeling a read-only array of strings, so we can have it implement `IVectorView<String>`. Many language projections expose this interface as an array-like object in the language itself. For example, C# projects the Windows Runtime `IVectorView<T>` as an `IList<T>`, and JavaScript projects it as an Array-like object. Now, `IVectorView<T>` comes with additional methods, so you'll have to implement those too, as well as `IIterable<T>` which is an interface required by `IVectorView<T>`.

Similarly, if you want to model your class as an associative array, you can implement `IMap<K, V>` or `IMapView<K, V>`.