# How do I declare an operator overload for my Windows Runtime class?

devblogs.microsoft.com/oldnewthing/20241106-00

November 6, 2024


A customer tried to add an operator overload to their Windows Runtime class:

```
// C++/WinRT

namespace winrt::Contoso::implementation
{
    struct Length : LengthT<Length&;
    {
        Length(double meters) : m_value(meters) {}

        Double Meters() { return m_value; }

        bool operator<(Length const& other)
        {
            reutrn m_value < other.m_value;
        }
    private:
        double m_value;
    };
}
```

But they found that it didn't work:

```
// C++/WinRT
winrt::Contoso::Length part1(1.0);
winrt::Contoso::Length part2(2.0);
if (part1 < part2) { // error

// C#
var part1 = new Contoso.Length(1.0);
var part2 = new Contoso.Length(2.0);
if (part1 < part2) { // error
```

It doesn't work because you added them to your implementation class, which is an implementation detail that is not visible to clients. Other parts of your implementation could use it when manipulating `Length` objects, but nobody outside the implementation can.

```
winrt::Contoso::Length part1(1.0);
winrt::Contoso::Length part2(2.0);
if (*winrt::get_self(part) < *winrt::get_self(part2)) {
```

People outside the implementation can see only what is exposed in the metadata, which is typically generated from the IDL file. And there is currently no way to express operator overloading in MIDL3.

```
namespace Contoso
{
    runtimeclass Length
    {
        Length(Double meters);

        Boolean operator<(Length other); // error
    }
}
```

I wasn't around for the initial decision, but I suspect that one reason is that not all languages support operator overloading. (For example, JavaScript doesn't.) And even among those which do, not all languages support the same operators. (For example, C++ and Python support the spaceship operator, but C# doesn't.)

But really, I think the reason is that operator overloading is not essential to the original mission of the Windows Runtime which is to provide a way for one language to consume objects produced by another language. Operator overloading would require a formalization of how to express it, including a fallback for languages that don't support it. This is a lot of work for something that isn't part of the core value proposition, and it could be interpreted as an expansion of scope.

You can fall back to using names.

```
namespace Contoso
{
    runtimeclass Length
    {
        Length(Double meters);

        Boolean IsLess(Length other);
        // or even
        Int32 Compare(Length other); // returns negative, zero, or positive
    }
}
```