If you want to sort a Windows Runtime collection, you may first want to capture it into something a bit easier to manipulate

devblogs.microsoft.com/oldnewthing/20230301-00

March 1, 2023

Raymond Chen

A customer wanted to sort a Windows Runtime collection. The obvious thing to try is

```
template<typename T, typename Compare>
void SortObservableVector(
    winrt::IObservableVector<T> const& observable,
    Compare comp)
{
    std::sort(begin(observable), end(observable),
        [&](auto&& a, auto&& b)
        { return comp(a, b); });
}
```

This works, but you can do better.

One problem with this version is that we are operating on an observable vector, which means that every vector operation raises a `VectorChanged` event, and the observers of this vector will get spammed with events as the sort operation proceeds.

Another problem is that every operation on the observable vector is a virtual method call. Depending on what you're sorting, this virtual method call overhead could be insignificant, or it could overwhelm the actual work you're trying to do.

One way to avoid both of the problem is to slurp the contents of the observable vector into a `std::vector`, sort the `std::vector`, and then shove the results back in.

```
template<typename T, typename Compare>
void SortObservableVector(
    winrt::IObservableVector<T> const& observable,
    Compare&& comp)
{
    // Slurp into a vector
    std::vector<T> v(observable.Size(),
        winrt_empty_value<T>());
    observable.GetMany(0, v);

    // Sort the vector
    std::sort(begin(v), end(v),
        std::forward<Compare>(comp));

    // Shove the results back in
    observable.ReplaceAll(v);
}
```

Bonus chatter: The Windows Implementation Library has a helper function `wil::to_vector` which wraps up the “slurp into a vector” while also addressing the race conditions I warned about in that earlier article.