Why does my attempt to index a collection with the x:Bind markup extension keep telling me I have an invalid binding path due to an unexpected array indexer?

devblogs.microsoft.com/oldnewthing/20190703-00

July 3, 2019

Raymond Chen

The documentation for the `x:Bind` markup extension notes that you can use a binding path to bind to a collection with an indexer. For example, `Teams[0].Players` and

```
<TextBlock Text="{x:Bind Players['John Smith']}" />
```

It notes that the prerequisites for being able to index into a collection from a binding path are that the model needs to implement `IList<T>` or `IVector<T>` for numeric indexing, or `IDictionary<string, T>` or `IMap<string, T>`.

So you try it out...

```
<TextBlock Text="{x:Bind Teams[0].Name}">
<TextBlock Text="{x:Bind Players['John Smith']}" />
```

... and you get errors.

```
WMC1110 Invalid binding path 'Teams[0]' : Unexpected array indexer
WMC1110 Invalid binding path 'Players['John Smith']' : Unexpected array indexer
XLS0522 Invalid index value ''John Smith''.
```

What's going on? What's wrong with the array indexer that made it unexpected? Why is `'John Smith'` an invalid index value?

Look more closely at the list of collection interfaces that support index notation. Notable by omission are the interfaces `IReadOnlyList<T>`, `IVectorView<T>`, `IReadOnly-Dictionary<string, T>` and `IMapView<string, T>`.

What's actually going on is that `Teams` and `Players` are not any of the types for which the `x:Bind` markup extension supports indexing. The error message was saying "I wasn't expecting an array indexer here, because the thing you are trying to apply it to is not something that can be indexed." (Writing error messages is hard.)

The `Teams` and `Players` properties are a read-only vector and read-only dictionary, and the `x:Bind` markup extension supports indexing only for read-write vectors and read-write dictionaries.

Nevermind that the binding was done in one-time mode and therefore will never attempt to write back to the vector or dictionary. The markup compiler requires that the vector and dictionary be read-write in order for you to be able to use index notation.

You can work around this by creating helper functions.

```
class MyPage
{
  public IReadOnlyList<Team> Teams { get; private set; }
  public IReadOnlyDictionary<string, string> Players { get; private set; }
  MyPage()
  {
    this.InitializeComponent();
  }

  string GetTeamName(int i) => Teams[i].Name;
  string GetPlayer(string s) => Players[s];
}

<!-- xaml -->
<TextBlock Text="{x:Bind GetTeamName(0)}">
<TextBlock Text="{x:Bind GetPlayer('John Smith')}" />
```

Unfortunately, you cannot chain beyond a function call, so you can't write

```
<!-- code in italics is wrong -->
<TextBlock Text="{x:Bind GetTeam(0).Name}">
```

Bonus chatter: If you are using one-way binding rather than one-time binding, then you need a way to signal that the result of the function call has changed. You can do that by raising a `PropertyChanged` event for the function name itself. This is analogous to how you raise `PropertyChanged` events when the value of a property changes.

Raymond Chen

Follow