How do I define a UWP XAML dependency property that is a collection type?

devblogs.microsoft.com/oldnewthing/20191004-00

October 4, 2019

Raymond Chen

In XAML, there are at least three ways to specify the contents of a collection-typed dependency property. For concreteness, I'm going to discuss UWP XAML, but the same principle apply to WPF XAML.

Let's say that we have a Doodad object with a Widgets property. The Widgets property is a collection of Widget objects.

First, we have the implicit collection:

```
<Doodad>
    <Doodad.Widgets>
        <Widget .../>
        <Widget .../>
        <Widget .../>
    </Doodad.Widgets>
</Doodad>
```

Second, we have the explicit collection:

```
<Doodad>
    <Doodad.Widgets>
        <MyWidgetCollection>
            <Widget .../>
            <Widget .../>
            <Widget .../>
        </MyWidgetCollection>
    </Doodad.Widgets>
</Doodad>
```

Third, we have binding:

```
<Doodad Widgets="{Binding MyWidgets}" .../>
<Doodad Widgets="{x:Bind MyWidgets}" .../>
```

Okay, let's tackle these in order.

The XAML compiler converts the implicit collection into C# code that is roughly equivalent to the below:

```
var e1 = new Doodad();
var widgets = e1.Widgets;
widgets.Add(new Widget(...));
widgets.Add(new Widget(...));
widgets.Add(new Widget(...));
```

(You can ask the XAML compiler to generate C++, but I'll use C# for notational convenience.)

In order for the implicit collection to work, the property must have an initial value that is an empty collection.

The explicit collection compiles to something like this:

```
var e1 = new Doodad();
var widgets = new MyWidgetCollection();
widgets.Add(new Widget(...));
widgets.Add(new Widget(...));
widgets.Add(new Widget(...));
e1.Widgets = widgets;
```

In order for explicit collections to work, the property must be settable.

Binding operates like this:

```
var e1 = FindTheDoodad();
e1.Widgets = this.MyWidgets;
```

This is equivalent to the explicit collection, so that same solution for explicit collections works for binding, too.

Okay, so how do we set up the dependency property so it satisfies all these requirements?

Turns out this is a special case of what we looked at last time: The dependency property whose initial value is a mutable object. In this case, the mutable object is itself a collection.

```
public class Doodad
{
  public static readonly DependencyProperty WidgetsProperty =
    DependencyProperty.Register(
      "Widgets",
      typeof(IList<Widget>),
      typeof(Doodad),
      new PropertyMetadata(null));

    public IList<Widget> Widgets
    {
      get => (IList<Widget>)GetValue(WidgetsProperty);
      set => SetValue(WidgetsProperty, value);
    }

    public Doodad()
    {
        this.InitializeComponent();
        Widgets = new List<Widgets>();
    }
  ..
}
```

Note that the type of the property is `IList<Widget>` instead of `List<Widget>`. That way, clients can assign or bind a custom collection.

Bonus chatter: There is documentation on how to create a WPF XAML dependency property that is a collection type, but it makes the mistake of presenting *incorrect* code first, without any immediate indication that the code is incorrect. I copied the initial code block, since it looked complete, but the result didn't work. (This is why, in my code samples, I'm careful to note that code in italics is wrong.)

Raymond Chen

Follow