On exactly when XAML bindings are evaluated if an element is not yet loaded

devblogs.microsoft.com/oldnewthing/20250205-00

February 5, 2025

The `x:Load` attribute allows you to specify in your XAML markup that the corresponding XAML element should not be created when the markup is loaded, but rather be created based on your app’s decisions. (You can have it loaded on demand and unloaded explicitly, or you can have it loaded and unloaded at times controlled by a binding.)

If an element which is delay-loaded contains a binding, when is that binding evaluated?

The answer depends on what kind of binding you have.

The classic `{Binding}` binding follows the principle that the bindings are activated when the element loads and they are deactivated when the element unloads. By comparison, the newer `{x:Bind}` binding follows the principle that the bindings should behave as if all elements were loaded, virtualizing out the fact that the element may not actually be loaded.

For `{x:Bind}`, this means that when an element is not loaded, the bindings are still active, but instead of applying the result of the binding to the unloaded element, the binding results are merely saved in a holding pen, and when the element finally gets loaded, the held values are applied.

The virtualization of unloaded elements means that you can add or remove `x:Load` for an element, and the bindings will still be calculated at the same time.

```
<!-- {x:Bind} without x:Load -->
<Page>
    <Line x:Name="L"
        X1="{x:Bind A, Mode=OneTime}"
        Y1="{x:Bind B, Mode=OneWay}" />
</Page>

<!-- {x:Bind} with x:Load=False -->
<Page>
    <Line x:Name="L"
        x:Load="False"
        X1="{x:Bind A, Mode=OneTime}"
        Y1="{x:Bind B, Mode=OneWay}" />
</Page>

<!-- {Binding} without x:Load -->
<Page>
    <Line x:Name="L"
        X1="{Binding A, Mode=OneTime}"
        Y1="{Binding B, Mode=OneWay}" />
</Page>

<!-- {Binding} with x:Load=False -->
<Page>
    <Line x:Name="L"
        x:Load="False"
        X1="{Binding A, Mode=OneTime}"
        Y1="{Binding B, Mode=OneWay}" />
</Page>
```

Here's a timeline of what happens when these four markup fragments are loaded and when the `MessageTextBlock` is loaded (realized) and unloaded (unrealized). Suppose that the initial value of `A` and `B` are both 1.

	`{x:Bind}`		**`{Binding}`**	
	without **`x:Load`**	**with** **`x:Load=False`**	**without** **`x:Load`**	**with** **`x:Load=False`**
Extra variables		`Saved_X1` `Saved_Y1`		
Page loads	Create `L`		Create `L`	
	`L.X1 = A`	`Saved_X1 = A`	`L.X1 = A`	
	`L.Y1 = B`	`Saved_Y1 = B`	`L.Y1 = B`	
`A` changes to 2				

`B` changes to 2	`L.Y1 = B`	`Saved_Y1 = B`	`L.Y1 = B`	

`L` loads		Create `L`
		`L.X1 = Saved_X1`
		`L.Y1 = Saved_Y1`
				Create `L`
				`L.X1 = A`
				`L.Y1 = B`

`L` unloads		Destroy `L`		Destroy `L`
`A` changes to 3				
`B` changes to 3	`L.Y1 = B`	`Saved_Y1 = B`	`L.Y1 = B`	

`L` loads		Create `L`
		`L.X1 = Saved_X1`
		`L.Y1 = Saved_Y1`
				Create `L`
				`L.X1 = A`
				`L.Y1 = B`

`L` unloads		Destroy `L`		Destroy `L`

The first thing to note is that the behavior without `x:Load` is the same with either `{x:Bind}` or `{Binding}`. The one-time and one-way bindings are calculated when the `Line` is loaded, and the one-way bindings are also updated when `B` changes.

The next thing to note is that the points at which the code reads from `A` and `B` (circled) are the same with `{x:Bind}` regardless of whether `x:Load` is in effect. If the bindings were more complicated, like involving a function call, the function calls would execute at page load, even though the element `L` might not exist yet.

An important behavior change with `{Binding}` is that when the element `L` is created, it is the value of `A` *at the time `L` is created* that is used to set `L`'s properties, rather than the value of `A` at the time the page loaded. The binding is "one-time", but the time the value of `A` is captured is different.

Another change with `{Binding}` is that if an element is unloaded, and then reloaded, the bindings are recalculated *at the time of reload*. When `L` is reloaded, the `L.X1` gets the value 3 from `A`, which is different both from the value it would have had without `x:Load` at all (which would have been 1), and is also different from the value it had the first time `L` was loaded (which was 2). So a one-time binding is actually performed twice! ("For large values of 1.")

Both binding patterns seem reasonable, yet they are quite different. The `{x:Bind}` theory is to make `x:Load` transparent: All bindings are calculated as if `x:Load` were not in effect at all. This allows you to add `x:Load="False"` to an element and be more confident that you won't be introducing timing issues. The `{Binding}` theory is to say that bindings are evaluated when the element *actually loads*, not when the element *would have* loaded.

Bonus chatter: The `{x:Bind}` paradigm of “act as if the element were always loaded” means that you may run into trouble if you are using `x:Load` to avoid loading things because they don’t work.

```
<TextBlock x:Name="Name"
    x:Load="{x:Bind IsNameApiAvailable}"
    Text="{x:Bind Name}" />
```

Even if the Name API is not available, the `{x:Bind Name}` will still ask for the `Name` property, causing your code to ask for the Name from an API that doesn’t exist, and that may not be something your code was expecting.

Bonus chatter: There is a quirk in the `{Binding}` case when an element is unloaded: If you are using a GC language like C#, the `Windows.UI.Xaml.Data.Binding` object that manages the binding will continue to exist after the element is unloaded. If you have a `OneWay` or `TwoWay` binding, that means that the `Binding` object is still listening for changes to the binding source and recalculating the value to bind, only to discover at the last second that the thing it’s supposed to apply the value to doesn’t exist any more! If you load and unload an object *N* times, there will be *N* change listeners and *N* calculations of the bound value, at least until the `Binding` gets garbage-collected. Fortunately, frequently loading and unloading the same element is not a common scenario.