Two-Way Data Binding with WinJS

By Dan Shultz, Joel Cochran, and James Bender, authors of *Windows 8 Apps with HTML5 and JavaScript*

There are many cases where it would be very helpful to push data from the UI back to the JavaScript object. This is known as two-way binding, and it is not supported by WinJS out of the box. In this article, based on chapter 5 of Windows 8 Apps with HTML5 and JavaScript, the authors show you how you can add support for this yourself.

You may also be interested in…

Two-way data binding is not directly implemented in WinJS. All data binding occurs in a one-way fashion from the JavaScript observable object to the HTML elements in the View. Often, however, you want data to flow in both directions. It makes sense that, when a user updates the value in a textbox or any other control, that value should be pushed back into the JavaScript object. While it is highly likely that two-way binding will be supported in the future, the good news is that, thanks to the great community of developers out there, you don't have to wait for Microsoft. In this article, we'll be basing our work on code Phil Japikse posted on his blog at http://skimedic.com/blog. Phil is a Microsoft MVP and a friend of the authors, and we thank him for his ongoing contributions to the community.

> NOTE: Phil Japikse, a long time Microsoft developer, community speaker and organizer, and advocate of the Model-View-ViewModel (MVVM) pattern, developed the two-way data binding approach this article is based on. You can find more of Phil's work at http://skimedic.com/blog/. Our thanks to Phil for all of his great work!

As this is code that we may want to use in many places, let's create a JavaScript file named *BindingSupport.js* for these helpers and add a reference to the new file in *default.html*:

```
<link href="/css/default.css" rel="stylesheet" />
<script src="js/Person.js"></script>
<script src="js/BindingSupport.js"></script>                    #A
<script src="/js/default.js"></script>
```

#A Add a reference to the new BindingSupport.js file.

To add the missing two-way binding functionality, we're going to use another feature of WinJS known as a *binding initializer*, which is a function that is called when a binding is initialized. We're going to use this initializer to add an event handler to the HTML element (also known as the *destination* object) that will itself update the JavaScript object (also known as the *source* or *target* object).

Before we start down the road of creating our binding initializer, let's add some helper code, courtesy of Phil. The code will create a namespace called `BindingHelpers` with two functions, `getDestinationValue` and `getSourceObject`. The `getDestinationValue` function deconstructs the HTML element to extract the current value of the bound attribute. The `getSourceObject` function locates the reference to the bound JavaScript object. There is some pretty clever JavaScript involved, as you can see in listing 1, so give a big thanks to Phil for slogging through it for us in advance.

Listing 1 Binding helper code to retrieve destination and source data

```
(function () {
    "use strict";
```

```
WinJS.Namespace.define("BindingHelpers", {
  getDestinationValue: function (dest, destProps) {                    #A
      if (destProps.length === 1) {                                    #B
          return dest[destProps[0]];                                   #B
      }
      else {
          var element = eval("dest['" +
                destProps.slice(0, destProps.length - 1).join("'][") +
                "']");                                                 #C
          return element[destProps[destProps.length - 1]];             #C
      }
  },
  getSourceObject: function (source, sourceProps) {                    #D
      if (sourceProps.length === 1) {                                  #E
          return source;                                               #E
      }
      else {
          return eval("source['" + sourceProps                         #F
              .slice(0, sourceProps.length - 1)
              .join("'][") + "']");
      }
  }
});
})();
```

#A We'll use BindingHelpers.getDestinationValue to extract the new value entered in the textbox.
#B If only one attribute is bound, return its value.
#C If multiple attributes are bound, return the last value.
#D We'll use BindingHelpers.getSourceObject to get the bound object.
#E If only one object property is bound, return it.
#F If multiple object properties are bound, return the last one.

With our helper code in place, we can now declare a function as a binding initializer. We'll do that by passing the desired function to `WinJS.Binding.initializer` as seen in listing 2.

Listing 2 Declaring a Binding initializer

```
WinJS.Namespace.define("BindingDirection", {                           #A
  twoWay: WinJS.Binding.initializer(                                   #B
    function (source, sourceProps, dest, destProps) {                  #C
      dest.oninput = function () {                                     #D
        var newValue = BindingHelpers
[CA].getDestinationValue(dest, destProps);                             #E
        var targetObject = BindingHelpers
[CA].getSourceObject(source, sourceProps);                             #F
        var oldValue = targetObject[sourceProps[sourceProps.length - 1]];#F
        if (oldValue !== newValue) {                                   #G
          targetObject[sourceProps[sourceProps.length - 1]] = newValue;  #G
        }
      };
   })
});
```

#A Create a namespace for the initializer.
#B Define a function named `twoWay` and call `WinJS.Binding.initializer`.
#C Pass a function with the binding initializer signature.
#D Process the `oninput` event. This is different from Phil's original code that used `onchange`.
#E Using the helper we defined previously, get the new value from the textbox.
#F Using the helper we defined previously, get the current value from the bound object.
#G If the values are different, update the bound object.

Other than the namespace names, this example only has one difference from the original code posted by Phil. Instead of processing the `onchange` event, we are processing the `oninput` event. The reason is simply one of timing: `onchange` will only fire when the input element loses focus. That means that the binding evaluation will only occur when the user moves to another field. The `oninput` event will fire every time the input value changes, including if the user uses copy and paste to populate the field. This approach is akin to using `UpdateSourceTrigger=PropertyChanged` in XAML or `valueUpdate: 'aftekeydown'` in KnockoutJS. If you prefer to not have the binding event fire immediately, use `onchange` instead of `oninput`.

Now with the binding initializer defined, we just need to add it to the HTML element's `data-win-bind` instructions. And, since we don't need to process the update in JavaScript, we can remove the `id` element and the update button.

```
<input type="text" data-win-bind="value: salary BindingDirection.twoWay" />
```

Note that the `data-win-bind` now includes a reference to the binding initializer. It's worth noting that in yet another quirk, the reference to the binding initializer also does not follow the traditional JavaScript object pattern. Since we've removed the button don't forget to remove or comment out the button's event handler code in *default.js*. If you don't you'll get an exception at runtime.

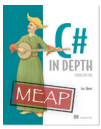
Summary

Thanks to HTML5's ability to handle custom data attributes, WinJS is able to provide a robust data binding system. Data binding will greatly ease your development efforts and open up a wealth of options to enhance your UI. Mastering these topics is crucial to crafting your Windows Store app. In this article, we gave you a tutorial on two-way binding, used when you want to push data from the UI back to the JavaScript object.

Here are some other Manning titles you might be interested in:

Dependency Injection in.NET
Mark Seemann

Windows Phone 7 in Action
Timothy Binkley-Jones, Massimo Perga and Michael Sync



C# in Depth, Third Edition
Jon Skeet

Last updated: 8/2/13