Binding a Floating Point Property to a `TextBox` With a `StringFormat` in WPF

WPF bindings are a fine piece of technology. They are flexible and powerful, allowing changes to your objects' properties to ripple through the rest of your application with a minimal amount of coding effort. (On your part. I'm sure Microsoft put a lot of work into the implementation.)

However, there is one small but very common situation where bindings behave poorly: binding a `TextBox` control to a floating point property (either `Single` or `Double`). When you bind these without specifying a `StringFormat` parameter, you cannot enter a decimal point so you cannot actually enter a floating point number. When you do specify a `StringFormat`, you cannot enter numbers naturally; you have to edit the string to look like the number you want.

Some examples of the unnatural editing style when using a `StringFormat` parameter (here, `F2`) are:

- Replacing the number does not work. If you highlight the number that is there and type "1.25", you end up with "1.25.00".

- Deleting the decimal point does not *re*move it, but moves it. If you start at the end of the number "12.88" and backspace three times, you get the number "1200.00".

When we ran across this problem on a recent project, I did what any programmer does these days for non-domain-related problems—I fired up Google to see who had already fixed this problem for me. Darn! Although a lot of people had complained about the problem, no one had a solution.

The only potential solution offered was to set the `KeepTextBoxDisplaySynchronizedWithTextProperty` property on the `FrameworkCompatibilityPreferences` class to `false`. Doing so allows you to enter a decimal point when you do not specify a `StringFormat` parameter on your floating point bindings. But it does not allow you to enforce a format, and therefore a precision, on your numbers. This may work for your project, but was not sufficient for ours.

A quick survey of third-party libraries showed no help from them, either. So if this was going to be fixed, we would have to fix it ourselves.

When you analyze the specifics of the undesirable behaviors, the problem boils down to the decimal point not being respected as a special character on input. If the cursor is at the decimal point and you type a period, you probably don't want a second decimal point in your number—you are acknowledging the one that is already there. If you are deleting or backspacing and come to the decimal point, perhaps you really do want to delete it; but if one is going to get added back at some other location, why not leave this one where it is and just move on?

This is the approach our `FloatingPointTextBox` control takes. It overrides `System.Windows.Controls.TextBox` and replaces certain keystrokes with movement of the caret. Don't allow the user to add any extra decimal points, and don't let them delete the last one. Done.

Well, almost done. Testing showed one additional irritating case: when using a custom format string (comprised of 0 and # characters), highlighting the entire number and typing in a negative number. After you enter the negative sign and the first digit, the `TextBox` moves the cursor to the end of the string. So if your format string is `00.00` and you highlight whatever number is in the text box and type “-12.34” you end up with “-1.00.34”.

The fix here is to watch specifically for this condition and move the cursor back to the left of the decimal point in time to catch the third character being typed. This is also included in our class—see the uses of the `NegativeIntegerPending` and `NegativeFractionPending` properties.

Using this control in our project has reduced the frustration our users were feeling when entering floating point numbers into our UI. Hopefully, it can give you and your users the same benefits.