The XAML hat rule: Understanding how it works and why it doesn't

devblogs.microsoft.com/oldnewthing/20200131-00

January 31, 2020

Raymond Chen

In the documentation for the `{x:Bind}` markup extension, it says

> If the data source is a Dictionary or Map, then a property path can specify items in the collection by their string name. For example **<TextBlock Text="{x:Bind Players['John Smith']}" />** will look for an item in the dictionary named "John Smith". The name needs to be enclosed in quotes, and either single or double quotes can be used. Hat (^) can be used to escape quotes in strings. Its usually easiest to use alternate quotes from those used for the XAML attribute.

The use of the hat to escape quotes is also mentioned in the documentation for functions in `x:Bind`, which says that an acceptable function argument is a

> Constant string enclosed in quotes — quotes are needed to designate it as a string. Hat (^) can be used to escape quotes in strings.

Okay, we get the idea. The hat can be used to escape quotes in strings.

Let's try it. Let's say I want to bind to the result of a function call whose parameter is a three-character string: `a"b`.

```
<TextBlock Text="{x:Bind Foo("a^"b")}" />
<TextBlock Text="{x:Bind Foo(^"a^"b^")}" />
```

These don't work.

The first thing to understand is that XAML is still XML, so the markup is always parsed first by an XML parser. The XAML hat rule is applied *after* XML parsing has taken place. Therefore, the above two attempts don't work because the result is not legal XML.

Let's try again, but escaping the quotation marks from XML parsing by using the `&quot;` entity.

```
<TextBlock Text="{x:Bind Foo(^&quot;a^&quot;b^&quot;)}" />
```

This at least gets past the XML parser, but the XAML parser still doesn't like it. After the XML parsing, the XAML parser sees the string

```
{x:Bind Foo(^"a^"b^")}
```

The deal is that the hat is for escaping the quotation mark *inside* strings, not for escaping the quotation marks that *delimit* the string.

Let's try that again.

```
<TextBlock Text="{x:Bind Foo(&quot;a^&quot;b&quot;)}" />
```

After XML parsing, the string that gets passed to the XAML parser is

```
{x:Bind Foo("a^"b")}
```

Now we're using the hat in the way the hat rule intended. The hat is escaping the embedded quotation mark. You can use it to escape a double-quote embedded inside a double-quote-delimited string. or to escape a single-quote embedded inside a single-quote-delimited string.

Too bad it still doesn't work. It passes the XAML parser, but you get a compiler error when it tries to compile the code generated by the XAML compiler. The specific code generation and resulting error depends on which language you're using.

```
// C++/WinRT code generation
            ::winrt::hstring p0 = L"a"b";

error C2001: newline in constant

// C++/CX code generation
        ::Platform::String^ p0 = "a"b";

error C2001: newline in constant
error C3688: invalid literal suffix 'b';
             literal operator or literal operator template
             'operator ""b' not found
error C2143: syntax error: missing ';' before 'Platform::String'

// C# code generation
                global::System.String p0 = "a"b";

error CS1002: ; expected
error CS1002: ; expected
error CS1010: Newline in constant
error CS1002: ; expected

' VB code generation
                Dim p0 As Global.System.String = "a"b"

error BC30648: String constants must end with a double quote.
(and a ton of cascade errors)
```

The hat gets the XAML parser to understand your intention: Putting a double-quote inside a string. But then the XAML code generator drops the ball and fails to escape the embedded quote when emitting it.

There are workarounds for this, some good and some bad.

Since VB and the C-derived languages disagree on how the embedded quotation mark should be escaped, you cannot write language-independent markup to work around this. A bad workaround is to write language-specific markup, which kind of breaks the principle that XAML is language-independent. It also means that if the XAML compiler ever fixed the code generation bug, their bug fix would break your code.

```
<!-- C++/WinRT, C++/CX, C# use a backslash to protect the quote -->
<TextBlock Text="{x:Bind Foo(&quot;a\\^&quot;b&quot;)}"/>

<!-- VB doubles the quote -->
<TextBlock Text="{x:Bind Foo(&quot;a^&quot;^&quot;b&quot;)}"/>
```

A better workaround is to move the troublesome string to a property that can be defined by the implementation, and the implementation can produce the embedded quotation mark in a language-dependent way.

```
<!-- add a property called AQuoteB whose value is a"b -->
<TextBlock Text="{x:Bind Foo(AQuoteB)}"/>
```

All is not completely lost. At least the hat rule works for single-quotes:

```
<TextBlock Text="{x:Bind Foo('a^'b')}"/>
```

I asked the XAML compiler team if they could fix the code generation bug, but they explained that they couldn't because it would be a breaking change. Specifically, it would break anybody who used the bad workaround above.

The result of all this is that the XAML hat rule is useful only for single-quotes, not for double-quotes. The XAML hat rule lets you get a quotation mark *into* the XAML parser, but unfortunately, double-quotes can't get back *out*.

Raymond Chen

Follow