Why does XAML complain that none of the overloads of winrt::to_hstring could be used?

devblogs.microsoft.com/oldnewthing/20230504-00

May 4, 2023

Raymond Chen

A customer was having trouble compiling their C++/WinRT project that used XAML binding.

They had a data template that binds the `Content` property of the `AwesomeThing` to a `TextBlock`'s text:

```
<!-- XAML -->
<DataTemplate x:DataType="local:AwesomeThing">
    <TextBlock Text="{x:Bind Content}" />
</DataTemplate>
```

But when they tried to compile this, they got a C++/WinRT error:

```
MainPage.xaml.g.hpp(200): error C2665: 'winrt::to_hstring': none of the 13 overloads
could convert all the argument types
    while trying to match the argument list '(winrt::Windows::Foundation::
IInspectable)'
    while compiling class template member function 'void winrt::Contoso::
implementation::MainPageT<winrt::Contoso::implementation::MainPage>::MainPage_obj1_
Bindings::Update_Content(winrt::Windows::Foundation::IInspectable,int32_t)'
    see reference to class template instantiation 'winrt::Contoso::implementation::
MainPageT<winrt::Contoso::implementation::MainPage>::MainPage_obj1_Bindings' being
compiled
MainPage.xaml.g.hpp(200): error C2660: 'winrt::Contoso::implementation::MainPageT<
winrt::Contoso::implementation::MainPage>::MainPage_obj1_Bindings::Set_Windows_UI_
Xaml_Controls_TextBlock_Text': function does not take 1 arguments
    see declaration of 'winrt::Contoso::implementation::MainPageT<winrt::Contoso::
implementation::MainPage>::MainPage_obj1_Bindings::Set_Windows_UI_Xaml_Controls_
TextBlock_Text'
```

What's going on here?

Let's see if we can decode the error message.

The first error says that somebody is trying to call `winrt::to_hstring` with something that can't be converted to a string. How curious.

The Visual Studio Error List window shows only the first line of each error message. To see the important supplemental information, you have to switch to the Output window. That's what tells us that the argument that couldn't be converted to a string is an `IInspectable`.

The second error says that somebody passed only one parameter to the `Set_Windows_UI_Xaml_Controls_TextBlock_Text` function. This is probably a cascade failure: The function takes two parameters, but one of them ended up being an error, so there was only one valid parameter left.

Looking at the source code confirms this diagnosis:

```
void Update_Content(::winrt::Windows::Foundation::IInspectable obj, int32_t phase)
{
    if((phase & ((1 << 0) | NOT_PHASED )) != 0)
    {
        Set_Windows_UI_Xaml_Controls_TextBlock_Text(obj7, ::winrt::to_hstring(obj));
    }
}
```

What we've learned so far is that the XAML compiler generated some code that takes an `IInspectable` and tries to convert to a string (via `to_hstring`), so it can set the `Text` property of a `TextBlock`.

Gosh, where in our XAML markup could we be trying to assign an `IInspectable` to a `Text` property?

Well, I made it easy because it is in fact the only binding in the entire XAML fragment: It's the `Text="{x:Bind Content}"`. (The other clue is that the method is named `Update_Content`.)

The `Content` property was declared like this:

```
// Contoso.idl

namespace Contoso
{
    runtimeclass AwesomeThing
    {
        Object Name{ get; };
        Object Content{ get; };
        /* and other properties */
    }
}
```

Notice that the `Content` property is declared as an `Object`, which is the MIDL compiler's name for what in the ABI is called `IInspectable`.

And that's why we have the error. The XAML markup is trying to bind the `TextBlock.Text` property (which is a string) to the `AwesomeThing.Content` property (which is an `IInspectable`). The XAML compiler sees that this is a type mismatch and tries to coerce the conversion by calling `to_hstring` and hoping for the best.

Unfortunately, it didn't work.

You can fix this in a few ways.

The simplest way is to change the type of the `Content` property to `String`. Then you're binding a string to a string, and everything lines up.

On the other hand, maybe your `Content` property is an `Object` on purpose, say, because it can hold multiple things. Maybe it's a string. Maybe it's an integer. Maybe it's a XAML element tree.

In that case, you'll have to teach XAML how to convert that mysterious object to a string. You can do this with a converter, or since you're already invested in `x:Bind`, you can use a function binding. Write a function to convert that mysterious object into a string, using an algorithm that describes how you want the conversion to occur. For example, maybe you'll write it like this:

```
// Contoso.idl

namespace Contoso
{
    runtimeclass AwesomeThing
    {
        Object Name{ get; };
        Object Content{ get; };
        /* and other properties */

        static String ContentToString(Object o);
    }
}

// C++/WinRT implementation

winrt::hstring AwesomeThing::ContentToString(IInspectable const& o)
{
    if (auto s = o.try_as<winrt::hstring>()) {
        return s.value();
    }
    if (auto i = o.try_as<int>()) {
        // assume that "%d" is acceptable
        return winrt::to_hstring(i.value());
    }
    if (auto e = o.try_as<winrt::FrameworkElement>()) {
        // ?? figure out what to return here
        return L"";
    }
    return L"";
}

<!-- XAML -->
<DataTemplate x:DataType="local:AwesomeThing">
    <TextBlock Text="{x:Bind local:AwesomeThing.ContentToString(Content)}" />
</DataTemplate>
```

You might have thought of adding a new property that does the conversion:

```
// Contoso.idl

namespace Contoso
{
    runtimeclass AwesomeThing
    {
        Object Name{ get; };
        Object Content{ get; };
        /* and other properties */

        String ContentAsString{ get; };
    }
}

// C++/WinRT implementation

winrt::hstring AwesomeThing::ContentAsString()
{
    auto o = Content();

    if (auto s = o.try_as<winrt::hstring>()) {
        return s.value();
    }
    if (auto i = o.try_as<int>()) {
        // assume that "%d" is acceptable
        return winrt::to_hstring(i.value());
    }
    if (auto e = o.try_as<winrt::FrameworkElement>()) {
        // ?? figure out what to return here
        return L"";
    }
    return L"";
}

<!-- XAML -->
<DataTemplate x:DataType="local:AwesomeThing">
    <TextBlock Text="{x:Bind ContentAsString}" />
</DataTemplate>
```

This works, but only because `x:Bind` defaults to one-time binding. If you switch to one-way binding:

```
<!-- XAML -->
<DataTemplate x:DataType="local:AwesomeThing">
    <TextBlock Text="{x:Bind ContentAsString, Mode=OneWay}" />
</DataTemplate>
```

then changes to the `Content` property will not trigger a recalculation of `ContentAsString`. You have to remember to raise a `PropertyChanged` event for `ContentAsString` whenever the `Content` property changes.

The function binding avoids this need to remember to raise extra property change notifications: Passing `Content` as a parameter to the function binding not only provides a parameter to the `ContentAsString` function, the XAML compiler will see that the parameter is a property and recalculate the binding whenever `Content` changes.