How do I specify an optional string parameter to a Windows Runtime method?

devblogs.microsoft.com/oldnewthing/20231215-00

December 15, 2023

Raymond Chen

Strings occupy a weird space in the Windows Runtime type system. They are formally classified as value types, even though they have reference behavior.

The problem is that different languages treat strings differently.

Language	String type	Value or reference	Mutable or immutable
C++	`std::wstring`	Value	Mutable
C#	`String`	Reference	Immutable
JavaScript	`String`	Reference	Immutable
Python	`str`	Reference	Immutable
Rust	`String`	Value	Mutable

To avoid creating problems over how to create an `IReference<String>` in languages where strings are already references, the Windows Runtime simply disallows `IReference<String>` outright.

Okay, so what are your options here?

If an empty string is an acceptable sentinel value to mean "No string", then you can just take a `String`. Internally, an empty `HSTRING` is represented by a null pointer, so the two cases are indistinguishable at the ABI layer.

```
runtimeclass Widget
{
    Widget(String name);

    // Property to get/set the Name after construction
    String Name;
}
```

In this case, an empty name is probably what you were going to do anyway if somebody wanted to create a Widget without a name.

In the above case, the optionality of the `name` parameter could be expressed by an overload that simply doesn't have a `name` parameter at all.

```
runtimeclass Widget
{
    Widget();            // create with no name
    Widget(String name); // create with explicit name

    // Property to get/set the Name after construction
    String Name;
}
```

Here's another case where you might be tempted to have an optional string:

```
runtimeclass Connection
{
    void Connect(String token, String server);
}
```

Maybe you want the token to be optional: If not provided, the code will obtain a new token.

And maybe you want the server to be optional: If not provided, the code will use the default server.

Since empty strings are probably not valid tokens or server names, you can use an empty string to mean "not provided".

But what if you want to distinguish between empty strings and no string at all?

```
runtimeclass WidgetFilter
{
    WidgetFilter();

    Windows.Foundation.IReference<Windows.UI.Color> Color;
    Windows.Foundation.IReference<bool> Polarity;
    String Title;
}

runtimeclass WidgetFinder
{
    static IVectorView<Widget> FindAll(WidgetFilter filter);
}
```

In this case, the `FindAll` method takes a `WidgetFilter` which lets you specify which widgets you are looking for. You can filter by color, by polarity, and by title.

The color and polarity can be represented by an `IReference`, where null means “Don’t care”. But we can’t use an empty title string to mean “Don’t care”, because that would prevent us from filtering to widgets whose title is the empty string.

This is, admittedly, a weird and unusual case.

Unfortunately, the solution is also weird and unusual.

To work around the inability to use an `IReference<String>`, you can box the string into an `Object`.

```
runtimeclass WidgetFilter
{
    WidgetFilter();

    Windows.Foundation.IReference<Windows.UI.Color> Color;
    Windows.Foundation.IReference<bool> Polarity;
    Object Title; // null for "don't care" or a boxed string
}

// C++/WinRT
WidgetFilter filter;
filter.Title(nullptr);               // don't care
filter.Title(winrt::box_value(L"")); // empty string as title

// C++/CX
WidgetFilter^ filter = ref new WidgetFilter();
filter->Title = nullptr;            // don't care
filter->Title = PropertyValue::CreateString(L""); // (see discussion)

// C#
WidgetFilter filter = new WidgetFilter();
filter.Title = null;                // don't care
filter.Title = "";                  // empty string as title

// JavaScript
var filter = new WidgetFilter();
filter.title = null;                // don't care
filter.title = "";                  // empty string as title
```

The C++/CX case of boxing an empty string is awkward because C++/CX tries to make strings look like objects. (Related reading: The C++/CX `String^` is not an object, even though it wears a hat.)

To force C++/CX to box an empty string to a non-null object, you put the empty string inside a `PropertyValue`, which implements `IReference<String>`, and then use that as the object.

To recover the string, you need to unbox.

```
// C++/WinRT
// Version 1
IInspectable title = filter.Title();
if (title) {
    add_title_filter(winrt::unbox_value<hstring>(title));
}

// Version 2
IInspectable title = filter.Title();
if (title) {
    add_title_filter(title.as<hstring>());
}

// Version 3
std::optional<hstring> title = filter.Title().try_as<hstring>();
if (title) {
    add_title_filter(title.value());
}

// C++/CX
Object^ title = filter->Title;
if (title) {
    add_title_filter(safe_cast<String^>(title));
}

// C#
object title = filter.Title;
if (title != null)
{
    add_title_filter((string)title);
}

// JavaScript
var title = filter.title;
if (title != null)
{
    add_title_filter(title);
}
```

The case of boxing a string where you need to distinguish between “no string” and “an empty string” is a weird and unusual case in the Windows Runtime. The best approach is to try to design your API so you don’t ever need to do it.

For example, you could change the `Title` property to a pair of methods.

```
runtimeclass WidgetFilter
{
    WidgetFilter();

    Windows.Foundation.IReference<Windows.UI.Color> Color;
    Windows.Foundation.IReference<bool> Polarity;

    void SetTitleFilter(String title);
    void ClearTitleFilter();
}
```

In this way, a caller can specify “I am looking for an empty title” by calling `SetTitle-Filter("")`, or they can specify “I don’t care about the title” by either never specifying a title filter or by clearing any previous title filter by calling `ClearTitleFilter()`.

On the implementation side, you can record the filter in a private `std::optional<winrt::hstring>` to inspect whether a filter has been applied.

Another option is to use a “nullable string at home” by pairing it with a Boolean property.

```
runtimeclass WidgetFilter
{
    WidgetFilter();

    Windows.Foundation.IReference<Windows.UI.Color> Color;
    Windows.Foundation.IReference<bool> Polarity;

    Boolean UseTitle;
    String Title;
}
```