If the Window Runtime PropertyValue is for boxing non-inspectables, why is there a PropertyValue.CreateInspectable?

devblogs.microsoft.com/oldnewthing/20250717-00/?p=111388

July 17, 2025

The Windows Runtime provides a class named `PropertyValue` which is a helper class for boxing non-inspectables. “Boxing” means taking a value type and wrapping inside an object so it can be used as an object type, and in the Windows Runtime, “objects” are represented by the `IInspectable` interface.

There are a variety of static methods of the form `PropertyValue.CreateSomething()`, like `CreateInt32()` or `CreateDateTimeArray()`. These take values and wrap them inside an object that implements the `IPropertyValue` interface,[1] and then you can use the `Type` property to see the type of the value hiding inside, and the corresponding `Get-Something()` method to retrieve the value.

One of the static methods is `PropertyValue.CreateInspectable()`. What does this even mean? Does it wrap an `IInspectable` inside another `PropertyValue`, which is itself an `IInspectable`?

No. There is no wrapping of `IInspectable`s.

The `CreateInspectable()` method merely returns its non-null parameter[2] unchanged. It doesn’t return a wrapper.

This means that if your original object does not implement `IPropertyValue` (and there's no reason to expect it to), then the object that comes out of `PropertyValue.CreateInspectable()` is not an `IPropertyValue`. In practice, it means that no `IPropertyValue`s will ever return `PropertyType::Inspectable`. That enum field is just a mirage.[3]

So why does `PropertyValue.CreateInspectable()` even exist?

I'm not sure. Perhaps it was added for completeness.

[1] The object also implements the corresponding `IReference<T>` interface, where `T` is the wrapped value.

[2] If the parameter is null, then it fails with an invalid argument exception.

[3] I guess you could use it in your own code to mean that "The thing you have is already an `IInspectable`, not a `PropertyValue` wrapper around a value type."

```
PropertyType WhatIsThisThing(IInspectable const& thing)
{
    if (auto propertyValue = thing.try_as<IPropertyValue>()) {
        return propertyValue.Type();
    }
    return PropertyType::Inspectable;
}
```