On the dangers of giving a product feature the name “new”

devblogs.microsoft.com/oldnewthing/20221110-00

November 10, 2022

Raymond Chen

A customer was having trouble adding a default parameter to their Windows Runtime method. When they used the `[defaultvalue]` attribute, they got the error

```
error MIDL2167: [msg]this attribute only allowed with new format type libraries
[context]: [defaultvalue()]
```

“Why does the error tell me that it’s allowed only with the new thing, when I’m using the Windows Runtime, which is the latest thing?”

This is a case of an error message written to a point in time.

Windows 2000 introduced a new type library file format, and you could select whether you wanted the old format or new format by passing the `/oldtlb` or `/newtlb` command line switches. The error message is saying that the `[defaultvalue]` attribute is supported only with new-style TLBs.

Mind you, those “new” TLBs are now over 20 years old. They may have been new once, but they’re not new any more.

I’m not a fan of giving a product feature the name “new”. It may be new at the time you introduce it, but after a while, it won’t be new, and any messages that refer to it as “new” will only end up confusing.[1]

For example, the New Executable format is now over 35 years old, originally designed for 16-bit Windows. Like anybody even remembers that. It’s so new that it isn’t supported any more.

Instead of calling your thing “new”, name it after the actual new thing. The “New Executable” could have been called a “Segmented Executable”, for example. If you can’t find a name for the new thing, you can chicken out and call it “v2”. In the above case, the error message would at least say “This attribute is allowed only with version 2 type libraries.”

In the case the customer was running into, they were compiling a Windows Runtime interface, and the `[defaultvalue]` attribute is simply not supported by the Windows Runtime at all. It's not listed in the documentation for MIDL 3.0 as a supported keyword.

The Windows Runtime doesn't support default parameters because not all languages supported by the Windows Runtime have the concept of default parameters. C++ and C# have them, but JavaScript for example does not, or at least it didn't have them until ES2015. Fortunately, you don't need default parameters. You can get the same effect with overloads.

```
runtimeclass Widget
{
    // can't do this:
    // void Sparkle(SparkleOptions options = SparkleOptions.None);

    // Do this instead:
    void Sparkle();
    void Sparkle(SparkleOptions options);
}
```

The difference between default parameters and overloads is that default parameters are encoded at the call site and therefore are immutable. On the other hand, the 0-arity overloaded function is implemented by the Widget itself, and it can choose what the default options are, even changing the defaults from version to version.

[1] One of my colleagues told me that when a new shopping mall was built in his home town, the locals called the original one the "old mall" and the new one the "new mall". These nicknames stuck even after the so-called "new mall" was no longer particularly new. What made things even more confusing is that the "old mall" was renovated, but the locals still call it the "old mall", leading to the odd situation where the "old mall" is actually newer than the "new mall".

Raymond Chen

Follow