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

devblogs.microsoft.com/oldnewthing/20231214-00

December 14, 2023

Raymond Chen

In C++, you can specify optional parameters with `std::optional<T>`.[1]

```
void MyMethod(
    std::optional<int> const& a,
    std::optional<int> const& b);

MyMethod(1, 2);                       // both parameters provided
MyMethod(1, std::nullopt);            // only first parameter provided
MyMethod(std::nullopt, 2);            // only second parameter provided
MyMethod(std::nullopt, std::nullopt); // neither parameter provided
```

In C#, you can use `Nullable<T>`. Rust uses `Option<T>`. In JavaScript, you can pass `null`. What about the Windows Runtime?

First of all, this was a trick question.

The C# and JavaScript versions aren't actually optional parameters. The C# `Nullable<T>` and JavaScript cases do not distinguish between passing `null` to mean "I am not passing this parameter" and passing `null` to mean "I am passing a parameter, and it is `null`." C++ distinguishes between the two cases:

```
void MyMethod(
    std::optional<const char*> const& a);

MyMethod(nullptr);      // We are passing a null pointer
MyMethod(std::nullopt); // We aren't passing anything
```

Let's continue the deception and assume that any passed parameter is not null. (In other words, passing null is equivalent to not passing anything.)

For reference types, you can use null as the sentinel value that means "no value".

```
// IDL
static runtimeclass MyClass
{
    static void MyMethod(MyClass a);
}
```

Projection	Reference consumer API producer	Reference producer API consumer
C++/WinRT	`MyMethod(MyClass a) { if (a) use(a); else use_nothing(); }`	`MyMethod(a); MyMethod(nullptr);`
C++/CX	`MyMethod(MyClass^ a) { if (a) use(a); else use_nothing(); }`	`MyMethod(a); MyMethod(nullptr);`
C#	`MyMethod(MyClass a) { if (a != null) use(a); else use_nothing(); }`	`MyMethod(a); MyMethod(null);`
JavaScript	`function myMethod(a) { if (a) use(a); else use_nothing(); }`	`myMethod(a); myMethod(null);`

For value types, you can use the `IReference<T>` interface to add a null value to a value type:

```
// IDL
static runtimeclass MyClass
{
    static void MyMethod(Windows.Foundation.IReference<Int32> a);
}
```

Projection	Projects as	IReference consumer API producer	IReference producer API consumer
C++/WinRT	`IReference<int>`	`MyMethod(IReference<int> a) { if (a) use(a.Value()); else use_nothing(); }`	`MyMethod(42); MyMethod(nullptr);`
C++/CX	`IBox<int>`	`MyMethod(IBox<int>^ a) { if (a) use(a->Value); else use_nothing(); }`	`MyMethod(42); MyMethod(nullptr);`

C#	`Nullable<int>`	`MyMethod(Nullable<int> a)`
		`{`
		`  if (a)`
		`use(a.Value);`
		`  else use_nothing();`
		`}`
			`MyMethod(42);`
			`MyMethod(null);`

JavaScript	`Object`	`function myMethod(a) {`
		`  if (a == null) use(a);`
		`  else use_nothing(); }`
			`myMethod(42);`
			`myMethod(null);`

There's one problem though: What if I want an optional string? Do I use the reference pattern or the value pattern?

It turns out the answer is "no".

We'll look at this some more next time.

[1] Note that optional parameters are not the same thing as default parameters. Default parameters are parameters which the compiler fills in for you if you leave them out of the parameter list. Optional parameters are parameters where you can pass a value, or you can say "No value here."