What is a static Windows Runtime class, how should I express it, and when should I use it?

devblogs.microsoft.com/oldnewthing/20210804-00

August 4, 2021

Raymond Chen

The Windows Runtime has this thing called “static classes”. What are they?

A static Windows Runtime class is a class that has no instances. An example of this is the `FileIO` class:

```
static runtimeclass FileIO
{
    static IAsyncOperation<String> ReadTextAsync(IStorageFile file);
    static IAsyncAction WriteText(IStorageFile file, String contents);

    static IAsyncOperation<IVector<String>> ReadLinesAsync(IStorageFile file);
    static IAsyncAction WriteLines(IStorageFile file, IIterable<String> lines);

    static IAsyncOperation<IBuffer> ReadBufferAsync(IStorageFile file);
    static IAsyncAction WriteBuffer(IStorageFile file, IBuffer buffer);

    /* etc */
}
```

There is no `FileIO` object. You can’t say

```
// C#
var fileIO = new FileIO(); // nope

// C++/WinRT
FileIO fileIO; // nope
auto fileIO = FileIO(); // nope

// C++/CX
auto fileIO = ref new FileIO(); // nope

// JavaScript
let fileIO = new FileIO(); // nope
```

None of these work because `FileIO` is not an object. It’s just a way to gather related functions under a common umbrella, similar to a namespace.

The term `static class` comes from the C# concept of the same name.

The way to express a static class is to put the word `static` in front of the class declaration, like we did above with `FileIO`:

```
static runtimeclass FileIO
{
    ...
}
```

If you leave out the `static` keyword (which is easily overlooked), then what you have is a class with no nonstatic members, also known as an *empty class*.

Empty classes are also a thing. They represent objects that you can't do anything with except pass to other methods. They are often used to capture some information into an opaque object which can then be passed around.

```
[default_interface]
runtimeclass WidgetLocation
{
}

runtimeclass Widget
{
    WidgetLocation Location;
}

// C#
// Move widget1 to the same location as widget2.
widget1.Location = widget2.Location;
```

The only thing you can do with a `WidgetLocation` is pass it to something that wants a `WidgetLocation`. You can't inspect the `WidgetLocation` to learn anything about it directly. You have to ask somebody else to interpret it for you.

The `FileIO` class is not one of these empty classes. It's not like you get an opaque `FileIO` object that represents some internal state. There simply isn't any such thing as a `FileIO` object at all.

And for that case, what you have is a static class.

Bonus chatter: The MIDL3 compiler leads you into a pit of failure here. There are five patterns for runtime classes:

	No static members	Has static members

No instances	N/A	`static runtimeclass C`
		`{`
		`static void S();`
		`}`

Has instances, empty	`[default_interface]`
	`runtimeclass C`
	`{`
	`}`
		`[default_interface]`
		`runtimeclass C`
		`{`
		`static void S();`
		`}`

Has instances, nonempty	`runtimeclass C`
	`{`
	`void M();`
	`}`
		`runtimeclass C`
		`{`
		`void M();`
		`static void S();`
		`}`

The top left corner is N/A because if a class has no static members and no instances, then there's nothing there.

If there are no instances, then you say that you are a `static runtimeclass`. That takes care of the first row.

If you are an empty class, then you need to say `[default_interface]` to tell the MIDL3 compiler that you want it to generate an empty interface to represent the empty instances.

If you are a nonempty class, then you don't need to say `[default_interface]` (although it's harmless to do so) because the MIDL3 compiler is already forced to generate an interface to represent the instance methods.

Notice that this case

```
runtimeclass C
{
 static void S();
}
```

is not even on the list of legal declarations!

The MIDL3 compiler lets you use this erroneous declaration. It assumes that you meant "Has instances, empty" and secretly synthesizes an empty `[default_interface]` for you. It's only when you try to do anything that requires this synthesized `[default_interface]`, that the MIDL3 compiler then yells at you, "Hey, you forgot to say `[default_interface]`."

The trap is that if you intended the class to be a static class, but simply forgot to apply the `static` keyword, then you will never do anything that requires the synthesized `[default_interface]`, and you never get any error message. The secretly synthesized empty interface is still generated, but it is completely useless since there is no way to obtain any objects that implement it.

Raymond Chen

Follow