# More on the mysterious [default_interface] attribute in Windows Runtime classes

devblogs.microsoft.com/oldnewthing/20240918-00

September 18, 2024


There is this mysterious attribute `[default_interface]` attribute that sometimes gets applied to Windows Runtime classes. What does this attribute mean, and when should you use it?

In the Windows Runtime, the "default interface" for a runtime class is the interface that is used at the ABI level to represent the object. We discussed default interfaces some time ago, so I'll consider that page background reading.

Normally, the default interface for a class is the autogenerated interface that has the same name as the runtime class (with an "I" in front). But what if there is no interface to autogenerate?

```
runtimeclass WidgetManager
{
    static void DisconnectAllWidgets();
}
```

This class consists only of static members. (In this case, it's a static method.) Those static methods go on the activation factory, as we discussed earlier.

There are no instance members: No instance properties, no instance methods, no instance events. There is nothing to put in the autogenerated `IWidget` interface.

The MIDL compiler finds this an odd state of affairs. You have an object that you can't do anything with. What's the point of that? So the compiler says, "Are you sure?"

There are two responses to this situation.

"Oops, sorry. I didn't mean for there to be any instances of this class. I should have declared this as a static class." In that case, make your class a static class.

```
static runtimeclass WidgetManager
{
    static void DisconnectAllWidgets();
}
```

Another response is “No really, I want there to be instances of this class. I know it looks funny to have an object that you can’t do anything with, but trust me, that’s what I want.” In that case, you add the `[default_interface]` attribute.

Now, there are cases where the MIDL compiler thinks that you created an object with no methods, but in fact the object does have methods because the methods are implemented by a base class or an implemented interface.

In the case of an interface, you should make that implemented interface be the default interface, so that the default interface is actually useful for something. If you ask for an empty default interface, then the currency for your object is a pointer to an interface that doesn’t do anything, and any operation people want to perform will have to perform an interface query.

```
runtimeclass ExtraHeadersHttpFilter : IHttpFilter
{
    ExtraHeadersHttpFilter(
        IIterable<IKeyValuePair<String, String> > headers);
}
```

The MIDL compiler wonders why you have an `ExtraHeadersHttpFilter` that you can’t do anything with. What it doesn’t realize is that you *can* do things with it: You can do `IHttpFilter` things.

You can tell the MIDL compiler to shut up by saying, “Oh, go ahead, make an empty `IExtra-HeadersHttpFilter` interface.”

```
// not the best solution
[default_interface]
runtimeclass ExtraHeadersHttpFilter : IHttpFilter
{
    ExtraHeadersHttpFilter(
        IIterable<IKeyValuePair<String, String> > headers);
}
```

This is not the best solution because it means that passing an `ExtraHeadersHttpFilter` object as a parameter passes the (empty) `IExtraHeadersHttpFilter` interface, which is not directly usable since it has no members. If somebody wants to call the `IHttpFilter.Send-RequestAsync` method, they will have to perform a `QueryInterface` to convert the `IExtra-HeadersHttpFilter` to an `IHttpFilter`, call the `SendRequestAsync` method, and then release the `IHttpFilter` interface. Much more efficient would be use `IHttpFilter` as the default interface, so that the recipient can just call `IHttpFilter` methods immediately.

Therefore, a better fix here is *not* to add `[default_interface]` to say "That's okay, give me an empty default interface." Instead, you say that `IHttpFilter` is your default interface.

```
runtimeclass ExtraHeadersHttpFilter : [default] IHttpFilter
{
    ExtraHeadersHttpFilter(
        IIterable<IKeyValuePair<String, String> > headers);
}
```

Another case where it looks like you have an empty object, but it really does have members is the case where you derive from another class, and the interesting methods are on the base class.

```
runtimeclass MyPage : Page
{
    MyPage();
}
```

The Windows Runtime doesn't let you name a base class as a default interface, so you are forced to create an empty default interface.

```
[default_interface]
runtimeclass MyPage : Page
{
    MyPage();
}
```

**Bonus chatter**: What goes wrong if I say `[default_interface]` when my runtime class does contain instance members? Do I get two interfaces, an empty default interface and a second interface that has the instance members?

No. If the runtime class has instance members, then `[default_interface]` is redundant but not harmful. There was going to be a default interface anyway.