API naming principle: If there is no direct object, then the direct object is the source object

devblogs.microsoft.com/oldnewthing/20240724-00

July 24, 2024

Raymond Chen

It is a common practice that method names begin with a verb: `widget.Toggle()`, `widget.SetColor()`, `widget.GetAssociatedDoodad()`.

Often, the verb is followed by a direct object: `widget.SetColor()`, `widget.GetAssociatedDoodad()`. The direct object is the thing that the verb operates on or produces. In the case of `SetColor`, it is setting the color. In the case of `GetAssociatedDoodad` it is getting the associated doodad.

Sometimes, the verb is not followed by a direct object at all, such as the `widget.Toggle()` method above. In that case, the direct object is the source object: The widget. In the above example, `widget.Toggle()` toggles the `widget`.

All of this may sound obvious, but it's easy to lose sight of this principle.

For example, a team proposed an API with a `WidgetNotification` and a method `widgetNotification.Delete()`. As written, it sounds like this deletes the widget notification itself, but the intention was for this to delete a notification listener. The methods for creating and deleting listeners should be named something like `widgetNotification.CreateListener()` and `widgetNotification.DeleteListener()`.

As another example, the name of the `ApplicationDataManager.CreateForPackageFamily` method doesn't say what it's creating, so the assumption is that it creates an `ApplicationDataManager`. But that's not what it creates. It actually creates an `ApplicationData` object.

The method should more properly be named `ApplicationDataManager.CreateDataFor-PackageFamily`.

An exception to this rule is factory objects. The purpose of factory objects is to create things, so a `WidgetFactory.Create()` method is assumed to create a widget. But I wouldn’t complain if you called your method `WidgetFactory.CreateWidget()`, just to be clear about it.