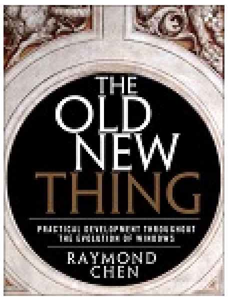
API design note: Don’t make up multiple names for the same thing

devblogs.microsoft.com/oldnewthing/20250728-00/?p=111415

July 28, 2025



A recurring problem I encounter when reviewing API proposals is that teams tend not to be precise in their use of terminology. This casualness is inevitable when you work with a feature for a long time and develop notational shortcuts, but the people who are learning your API don’t have the same level of familiarity that you do, and shifting terminology tends to create confusion.

For example, there was an API proposal that included two methods.

```
runtimeclass Widget
{
    void EnableFilter(WidgetFilter filter);
    Boolean AreAnyFiltersApplied();
}
```

The first method talks about “enabling” but the second talks about “applying”.

For somebody encountering this API for the first time, the existence of two different terms raises questions. Is enabling a filter the same as applying it? Or are there two steps to making a filter active, first you enable it, then you apply it? (Or do you apply it first, and then enable it?)

When I asked the team, they said that enabling and applying are two names for the same thing. They internally use both terms to refer to adding filters to a widget.

I recommended that they not use multiple names for the same concept. This makes it harder to see that the two methods are counterparts to each other. Pick a name and stick with it.

They chose to use “Enable” throughout, so the second method was renamed to `AreAnyFiltersEnabled()`.

This consistency extends beyond method names. If there is a parameter that corresponds to a property, use the same name for both the parameter and the property in order to make the connection clear.

```
runtimeclass Widget
{
    Widget(String id);

    String Name { get; }
}
```

In this case, the intention is that the `id` parameter passed to the constructor can be read back by reading the `Name` property. In that case, the parameter and property should either both be called `Id` or both called `Name`.

```
// Option 1
runtimeclass Widget
{
    Widget(String id);

    String Id { get; }
}

// Option 2
runtimeclass Widget
{
    Widget(String name);

    String Name { get; }
}
```