If you're just going to sit there doing nothing, at least do nothing correctly

devblogs.microsoft.com/oldnewthing/20240216-00

February 16, 2024

Raymond Chen

There may be times where you need to make an API do nothing. It's important to have it do nothing in the correct way.

For example, Windows has an extensive printing infrastructure. But that infrastructure does not exist on Xbox. What should happen if an app tries to print on an Xbox?

Well, the wrong thing to do is to have the printing functions throw a `NotSupportedException`. The app that the user installed on the Xbox was probably tested primarily, if not exclusively, on a PC, where printing is always available. When run on an Xbox, the exception will probably go unhandled, and the app will crash. Even if the app tried to catch the exception, it would probably display a message like "Oops. That went badly. Call support and provide this incident code."

A better design for "supporting" printing on Xbox is to have the printing functions succeed, but report that there are no printers installed. With this behavior, when the app tries to print, it will ask the user to select a printer, and show an empty list. The user realizes, "Oh, there are no printers," and cancels the printing request.

To deal with apps that get fancy and say "Oh, you have no printers installed, let me help you install one," the function for installing a printer can return immediately with a result code that means "The user cancelled the operation."

The idea here is to have the printing functions all behave in a manner perfectly consistent with printing being fully supported, yet mysteriously there is never a printer to print to.

Now, you probably also want to add a function to check whether printing even works at all. Apps can use this function to hide the Print button from their UI if they are running on a system that doesn't support printing at all. But naïve apps that assume that printing works will still behave in a reasonable manner: You're just on a system that doesn't have any printers and all attempts to install a printer are ineffective.

The name we use to describe this "do nothing" behavior is "inert".

The API surface still exists and functions according to its specification, but it also does nothing. The important thing is that it does nothing in a way that is consistent with its documentation and is least likely to create problems with existing code.

Another example is the retirement of an API that has a variety of functions for creating widget handles, other functions that accept widget handles, and a function for closing widget handles. The team that was doing the retirement originally proposed making the API inert as follows:

```
HRESULT CreateWidget(_Out_ HWIDGET* widget)
{
    *widget = nullptr;
    return S_OK;
}

// Every widget is documented to have at least one alias,
// so we have to produce one dummy alias (empty string).
HRESULT GetWidgetAliases(
    _Out_writes_to_(capacity, *actual) PWSTR* aliases,
    UINT capacity,
    _Out_ UINT* actual)
{
    *actual = 0;

    RETURN_HR_IF(
        HRESULT_FROM_WIN32(ERROR_MORE_DATA),
        capacity < 1);

    aliases[0] = make_cotaskmem_string_nothrow(L"").release();
    RETURN_IF_NULL_ALLOC(aliases[0]);

    *actual = 1;
    return S_OK;
}

// Inert widgets cannot be enabled or disabled.
HRESULT EnableWidget(HWIDGET widget, BOOL value)
{
    return E_HANDLE;
}

HRESULT Close(HWIDGET widget)
{
    RETURN_HR_IF(E_INVALIDARG, widget != nullptr);
    return S_OK;
}
```

I pointed out that having `CreateWidget` succeed but return a null pointer is going to confuse apps. “The call succeeded, but I didn’t get a valid handle back?” I even found some of their own test code that checked whether the handle was null to determine whether the call succeeded, rather than checking the return value.

I also pointed out that having `EnableWidget` return “invalid handle” is also going to create confusion. An app calls `CreateWidget`, and it succeeds, and it takes that handle (which is presumably valid) and tries to use it to enable a widget, and it’s told “That handle isn’t valid.” How can that be? “I asked for a widget, and you gave me one, and then when I showed it to you, you said, ‘That’s not a widget.’ This API is gaslighting me!”

I looked through the existing documentation for their API and found that a documented return value is `ERROR_CANCELLED` to mean that the user cancelled the creation of the widget. Therefore, apps are already dealing with the possibility of widgets not being created due to conditions outside their control, so we can take advantage of that: Any time the app tries to create a widget, just say “Nope, the, uh, user cancelled, yeah, that’s what happened.”

```
HRESULT CreateWidget(_Out_ HWIDGET* widget)
{
    *widget = nullptr;
    return HRESULT_FROM_WIN32(ERROR_CANCELLED);
}

HRESULT GetWidgetAliases(
    _Out_writes_to_(capacity, *actual) PWSTR* aliases,
    UINT capacity,
    _Out_ UINT* actual)
{
    *actual = 0;
    return E_HANDLE;
}

HRESULT EnableWidget(HWIDGET widget, BOOL value)
{
    return E_HANDLE;
}

HRESULT Close(HWIDGET widget)
{
    return E_HANDLE;
}
```

Now we have a proper inert API surface.

If you try to create a widget, we tell you that we couldn’t because the user cancelled. Since all attempts to create a widget fail, there is no such thing as a valid widget handle, and any time you try to use one, we tell you that the handle is invalid.

This also avoids the problem of having to produce dummy aliases for widgets. Since there *are no widgets*, there is no legitimate case where an app could ask a widget for its aliases.

Bonus chatter: To clear up some confusion: The idea here is that the printing API has always existed on desktop, where printing is supported, and the “get me the list of printers” function is documented not to throw an exception. If you want to port the printing API to Xbox, how do you do it in a way that allows existing desktop apps to continue to run on Xbox? The inert behavior is completely truthful: There are no printers on an Xbox. Nobody expects the answer to the question, “How many printers are there?” to be “How dare you ask me such a thing!”

Another scenario where you need to create an inert API surface is if you want to retire an existing API. How do you make the behavior of the API consistent with its contract while still doing nothing useful?