How can I tell whether a window is modal?

devblogs.microsoft.com/oldnewthing/20111212-00

December 12, 2011

Raymond Chen

A customer wanted a way to determine whether a particular window is modal. They listed a few methods they had tried but found that it didn't work and asked for assistance.

As Eric Lippert is fond of saying, "First, write your spec." Until you know what you want, you won't know how to get it.

First, you need to define what you mean by *a modal window*. There are multiple competing definitions.

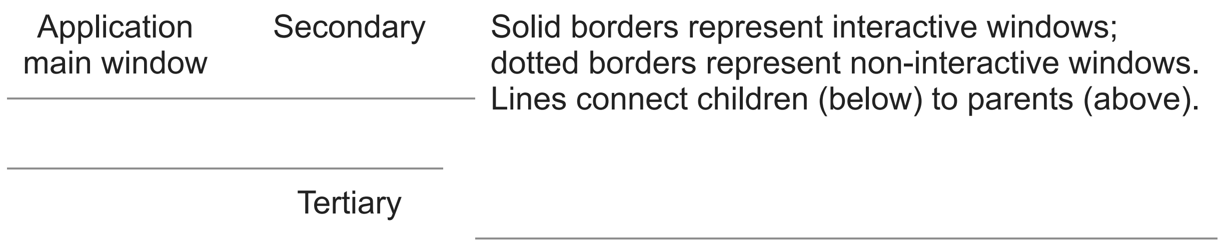
The customer decided that the definition of *modal window* they want is this one:

> A modal window is a child window that requires the user to interact with it before they can return to operating the parent application, thus preventing any work on the application main window.

One thing you notice in this definition is that it talks both about *windows* and *applications*. You have a child *window*, a parent *application*, and even an *application main window*. This implies that a modal window must be in a different application from its parent. (If it were in the same application, then it vacuously does not prevent you from interacting with the parent application because it *is* the parent application.) But modality is a user interface concept, not a process management concept, so it's unclear why process considerations appear in the definition. End-users sitting in front of a user interface see windows, not processes. I'm going to assume that the use of the term *application* here is a mistake, and that all we're talking about is *windows*.

The second thing you realize from this definition is that it is describing something impossible. In Windows, child windows cannot be interacted with when their parent window is disabled. This definition appears to be using a common abuse of terminology, using the words *child* and *parent* instead of the more accurate but clumsier *owned* and *owner*. This common abuse of terminology rarely causes trouble among people with experience programming the Windows user interface, but it is often a source of confusion for beginners, which is why I try to use the precise terminology rather than the casual terminology. And this question was clearly asked by a beginner.

Another thing you notice about this definition is that it involves not two but *three* windows: The child window, the parent window, and the application main window. Consider the situation where you have an application main window (which is interactive), a secondary window (which is not interactive), and a tertiary window which is a child of by the secondary window with which the user must interact in order to return to operating the secondary window.


Application main window
Secondary
Tertiary
Solid borders represent interactive windows; dotted borders represent non-interactive windows. Lines connect children (below) to parents (above).

Is the tertiary window modal, according to this definition? I'm not sure. It is not clear to me whether the clause "thus preventing any work on the application main window" is an additional constraint or is merely elaborative. If the clause is an additional contraint, then the situation is not modal, because the application main window is still interactive. On the other hand, if the clause is merely elaborative, then the situation *is* modal, because the tertiary window prevents the user from interacting with the secondary window.

The fourth thing you realize from this definition is that it requires predicting the future. How do you know that the owner window will be available for use once you dismiss the owned window? Mabe the application does " `if (time(NULL) % 2) make_parent_available();` ". (Perhaps we can call upon the graduates of the DePaul University with a degree in predicting the future to help us here.)

Even if the result doesn't depend on predicting the future, determining whether the window will re-enable its parent requires a level of code understanding beyond what can easily be achieved programmatically. (You would have to find the code in the other program and study it to determine whether it re-enables the parent window as part of its interaction. This can be hard to do by a human being with source code, much less by a computer program with only object code, especially if the object code is in an interpreted language, since you now have to reverse-engineer the interpreter too!)

No wonder the problem is so difficult: The spec uses imprecise terminology, is unclear on its criteria, and requires metaphysical certitude beyond the current level of scientific understanding.

Let's see what we can salvage from this definition. First, let's make the terminology more precise:

> A modal window is an owned window that requires the user to interact with it before they can return to operating the owner window, thus preventing any work on the application main window.

Next, let’s delete the clause whose meaning is unclear.

> A modal window is an owned window that requires the user to interact with it before they can return to operating the owner window.

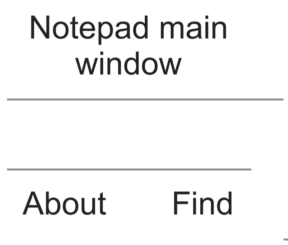
Finally, let’s remove the part that requires predicting the future. Instead of describing future behavior (which is hard to predict), we’ll make our requirements based on *present* behavior (which can be observed without the aid of a time machine).

> A modal window is an owned window whose owner window cannot be interacted with.

The revised spec says that a modal window is an owned window whose owner is disabled. Bingo, there’s your algorithm for detecting whether a window is modal. Once you have a good spec, the code pretty much writes itself:

```
BOOL IsModalWindowAccordingToThisParticularSpec(HWND hwnd)
{
 // child windows cannot have owners
 if (GetWindowStyle(hwnd) & WS_CHILD) return FALSE;
 HWND hwndOwner = GetWindow(hwnd, GW_OWNER);
 if (hwndOwner == NULL) return FALSE; // not an owned window
 if (IsWindowEnabled(hwndOwner)) return FALSE; // owner is enabled
 return TRUE; // an owned window whose owner is disabled
}
```

Mind you, this spec may still not be what you actually want. Consider the Notepad program. Type Ctrl+F to call up the Find dialog. This is a modeless dialog: The main window is still interactive. While the Find dialog is up, call up the About dialog from the Help menu. You now have the main Notepad window with two owned windows, an About dialog that will re-enable the main Notepad window when it is dismissed, and a Find dialog that will not.



A connector is solid if the owned window re-enables the owner, dotted if it does not. (Remember, whether the line is dotted or not cannot be determined algorithmically.)

According to our spec, which of these windows is modal? Does that match your intuitive sense?

Here’s another case: From Notepad’s Open dialog, type the name of a file that does not exist.

Notepad main window

Open

File not found

Which of these windows is a modal window?

Still unresolved is whether this is the right definition for the customer's needs. The customer never explained why they needed to identify modal windows, and once we gave them the `IsModalWindowAccordingToThisParticularSpec` function, they never wrote back.

If they were trying to identify modal windows so they could try to close them, then in the *File not found* case above, they may try to close the *Open* window, when the correct window to close first is the *File not found* window, because you need to respect a window's disabled state.

Since the customer never wrote back, we will never know.

Raymond Chen

Follow