It's not a good idea to give multiple controls on a dialog box the same ID

devblogs.microsoft.com/oldnewthing/20120619-00

June 19, 2012

Raymond Chen

When you build a dialog, either from a template or by explicitly calling `CreateWindow`, one of the pieces of information about each control is a child window identifier. And it's probably in your best interest to make sure two controls on the dialog don't have the same ID number. Of course, one consequence of giving two control the same ID number is that the `GetDlg-Item` function won't know which one to return when you say, "Please give me control number 100." This naturally has a cascade effect on all the other functions which are built atop the `GetDlgItem` function, such as `SetDlgItemInt`. Another reason to avoid duplication is that many notification messages include the control ID, and if you have a duplicate, you won't know which one generated the notification. Okay, this isn't actually the case, because the notification messages typically also include the window handle, so you can use the window handle to distinguish between your two controls both with ID=100. Though it means that you can't use a simple switch statement any more. (See sidebar for discussion of when duplicate IDs are acceptable.) Most of the time, you get away with the duplicate IDs because you can use the window handle to distinguish them. But there is one notable case where duplicate IDs cause problems: Identifying the default pushbutton on the dialog. One of the things that the dialog manager does when it builds a dialog box from a template is keep an eye out for a button control with the `BS_DEFPUSHBUTTON` style. When it finds one, it remembers the control ID so that it can restore it as the default pushbutton when focus is on a non-pushbutton control. (When focus is on a pushbutton, then that button becomes the default pushbutton.) The dialog manager records the initial default pushbutton by sending itself the `DM_SETDEFID` message, and the default handler merely records the value in a safe place so it can return it when somebody sends the `DM_GETDEFID` message. You can send the `DM_SETDEFID` message yourself if you want to change the default pushbutton, and that's where the trouble comes in. The only parameter to the `DM_SETDEFID` message is the ID of the dialog control you want to be the new default, so if your dialog box has two controls with that ID, then you've created a bit of a problem. When the user hits the Enter key, the dialog manager wants to fire a `WM_COMMAND` notification for the default button, but it sees two of them and gets confused. Actually, it doesn't really get confused. It just picks one of them arbitrarily and ignores the other one. And then confusion sets in. If the two buttons with conflicting IDs do different things, then your code which receives the `WM_COMMAND` notification may end up seeing the notification coming from the wrong control. For example, suppose you inadvisedly assign ID number 100 to both the *Reformat* and *Scan* buttons (and out of an abundance of caution, set *Scan* as the default pushbutton). When the user hits Enter, the dialog manager sends a `WM_GETDEFID` message to say, “Hey, what’s the default pushbutton?” The message returns 100, and now the dialog manager is stuck saying to itself, “Um, there are two 100’s. Eeny, meeny, miny, moe. Okay, it’s the *Reformat* button.” Boom, hard drive reformatted. From the same dialog template, suppose you realize, “Oh, I don’t want to let the user reformat the hard drive until they’ve entered a volume label,” so you disable the *Reformat* button if the volume label field is blank. The user hits Enter, and remember, you set *Scan* as the default button. But since *Reformat* and *Scan* have the same control ID, the dialog manager once again plays eeny-meeny-miney-moe, and say it picks the *Reformat* button. But it also sees that the *Reformat* button is disabled, so it just beeps. And then your user wonders why, when they hit Enter and the *Scan* button is the default pushbutton, it doesn’t scan but instead just beeps. Okay, all this discussion seems pretty obvious, doesn’t it, but as we dig deeper into the dialog manager, you’ll see how the principle of “Don’t create a dialog box with conflicting dialog control IDs” has perhaps unexpected consequences. **Sidebar**: If the duplications are all among controls that do not raise notifications and which you do not need to identify programmatically, then you’re not going to run into much trouble at all. By convention, the “control ID for controls where I don’t care about the ID” is −1, although you can use any number you like; the window manager doesn’t care, as long as it doesn’t collide with the ID of a control that you *do* care about. Note that some resource management tools (such as translation toolkits, or interactive dialog editors) assume that there are no duplicate IDs aside from the special don’t-care value −1, so if you’re going to use duplicate IDs because you don’t care, you’d be well-served to stick to the −1 convention. **Bonus chatter**: Why doesn’t `DM_SETDEFID` take a window handle instead of a control ID? That would solve the problem of multiple controls with the same ID, since you now have the window handle, which identifies the control uniquely. Yeah, it could’ve done that. Though it would also have created problems if the default pushbutton is destroyed, and that happens more often than you think. Remember back in the early 16-bit days, we didn’t have parameter validation, so a dangling window handle meant that you crashed when you tried to use it. (Or worse, the window handle could have been reused for another totally unrelated window! Window handle reuse was much more common in 16-bit Windows.) Mapping the window handle back to an ID and then converting the ID to a window on demand meant that you never keep a window handle around, which means that you never have to worry about the handle going bad. Making the `DM_SETDEFID` message handle-based would also make it harder for somebody to pull the “Create two controls with the same ID but hide one of them at runtime” trick described above, because they would also have to remember to send a hypothetical `DM_SETDEFHWND` message whenever they pulled the switcheroo.

And besides, the only people this design change helps out are people who put multiple visible controls on a dialog box with the same ID. You don't optimize for the case where somebody is mis-using your system.

Raymond Chen

Follow