The checkbox: The mating call of the loser

devblogs.microsoft.com/oldnewthing/20090213-00

February 13, 2009

Raymond Chen

(Cultural note: The phrase *the mating call of the loser* is a term of derision. I used it here to create a more provocative headline even though it's stronger than I really intended, but good writing is bold.)

When given a choice between two architectures, some people say that you should give users a checkbox to select which one should be used. That is the ultimate cowardly answer. You can't decide between two fundamentally different approaches, and instead of picking one, you say "Let's do both!", thereby creating triple, perhaps quadruple the work compared to just choosing one or the other.

It's like you're remodeling a book library and somebody asks you, "Should we use Dewey Decimal or Library of Congress?" Your answer, "Let's do both and let the user choose!"

Imagine if there were a checkbox somewhere in the Control Panel that let you specify how Windows XP-styled controls were implemented. Your choices are either to require applications to link to a new `UxCtrl.DLL` (let's call this Method A) or to link to `COMCTL32.DLL` with a custom manifest (let's call this Method B). Well, it means that every component that wanted styled common controls would have to come in two versions, one that linked to `UxCtrl` and used the new class names in its dialog boxes and calls to `CreateWindow`, and one that used a manifest and continued to use the class names under their old names.

```
#ifdef USE_METHODA
 hwnd = CreateWindow(TEXT(“UxButton”), …);
#else
 hwnd = CreateWindow(TEXT(“Button”), …);
#endif


DLG_WHATEVER DIALOG 36, 44, 230, 94
STYLE WS_POPUP | WS_CAPTION | WS_SYSMENU |
      DS_MODALFRAME | DS_SHELLFONT
CAPTION “Whatever”
BEGIN
#ifdef USE_METHODA
    CONTROL         “Whatever”,IDC_WHATEVER,”UxButton”,
                    BS_AUTOCHECKBOX | WS_TABSTOP, 14,101,108,9
#else
    CONTROL         “Whatever”,IDC_WHATEVER,”Button”,
                    BS_AUTOCHECKBOX | WS_TABSTOP, 14,101,108,9
#endif
    …
```

At run time, every program would have to check this global setting and spawn off either the “Method A” binary or the “Method B” binary.

Now you might try to pack this all into a single binary with something like this:

```
if (GetSystemMetrics(SM_USEMANIFESTFORSTYLEDCONTROLS)) {
 hwnd = CreateWindow(TEXT(“Button”), …);
} else {
 hwnd = CreateWindow(TEXT(“UxButton”), …);
}
…


if (GetSystemMetrics(SM_USEMANIFESTFORSTYLEDCONTROLS)) {
  DialogBox(hInstance,
            MAKEINTRESOURCE(DLG_TEMPLATE_WITH_OLDNAME_CONTROLS),
            …
} else {
  DialogBox(hInstance,
            MAKEINTRESOURCE(DLG_TEMPLATE_WITH_UXCONTROLS),
            …
}
```

But it’s not actually that simple because a lot of decisions take place even before your program starts running. For example, if your program specifies a load-time link to `UXCTRL.DLL`, then that DLL will get loaded before your program even runs, even if the

system switch is set to use Method B. A single-binary program that tries to choose between the two methods at runtime will have to do some activation context juggling and delay-loading. Hardly a slam dunk.

Okay, so now you have two versions of every program. And you also have to decide what should happen if somebody writes and ships a program that uses Method A exclusively, even when the system switch is set to Method B. Does everything still work within that program as if Method A were the system setting, while the rest of the system uses Method B? (If you go this route, then you've completely undermined the point of Method B. The whole point of Method B is to allow programs that rely on specific class names to continue working, but this rogue Method A program is running around using the wrong class names!)

Now the entire operating system and application compatibility work needs to be done with the checkbox set both to Method A and to Method B, because the compatibility impact of each of the methods is quite different. Okay, that's double the work. Where is triple and quadruple?

Well, the two different versions of the program need to be kept in sync, since you want them to behave identically. This can of course be managed with judicious use of `#ifdef` s or runtime branches. But you have to remember both ways of doing things and be mindful of the two method each time you modify the program. Somebody else might come in and "fix a little bug" or "add a little feature" to your program, unaware of how your program manages the shuffle of Method A versus Method B. The mental effort necessary to remember two different ways of doing the same thing plus having to expend that effort to correct mistakes in the code, that's the triple.

The quadruple? I'm not sure, maybe the ongoing fragility of such a scheme, especially one that, at the end of the day, is a choice between two things that have no real impact on the typical end user.

Engineering is about making tradeoffs. If you refuse to make the choice, then you're taking the cowardly route and ultimately are creating more work for your team. Instead of solving problems, you're *creating them*.

All because you're too chicken to make a hard decision.

Raymond Chen

Follow