# The resource compiler will helpfully add window styles for you, but if you're building a dialog template yourself, you don't get that help

devblogs.microsoft.com/oldnewthing/20121122-00

November 22, 2012

Raymond Chen

A customer was having trouble with nested dialogs. They were doing something very similar to a property sheet, with a main frame dialog, and then a collection of child dialogs that take turns appearing inside the frame dialog based on what the user is doing. The customer found that if they created the child dialogs with the `CreateDialogParam` function, everything worked great, but if they built the template at run-time, keyboard navigation wasn't working right. Specifically, one of their child dialogs contained an edit control, and while you could put focus on it with the mouse, it was not possible to tab to the control. On the other hand, a resource template didn't have this problem. Tabbing in and out worked just fine.

There is logically no difference between a resource-based dialog template and a memory-based one, because the resource-based one is implemented in terms of the memory-based one.

The real problem is the memory-based template you created differs somehow from the one the resource compiler generated.

One way to identify this discrepancy is simply to do a `memcmp` of the two dialog templates, the resource-based one and the memory-based one, and see where they differ. After all, if you want to know why your template doesn't match the one generated by the resource compiler, you can just *ask the resource compiler to generate the template* and then compare the two versions.

Instead of explaining this, I decided to invoke my psychic powers.

> My psychic powers tell me that you neglected to provide the `WS_TABSTOP` style to the edit control when you created your in-memory template. (You probably didn't realize that you needed to do this, because the resource compile adds that style by default.)

When you use the resource compiler to generate a dialog template, it sets a bunch of styles by default, depending on the type of control. For example, `EDITTEXT` says "If you do not specify a style, the default style is `ES_LEFT | WS_BORDER | WS_TABSTOP`."

Not mentioned is that the default style is in addition to the defaults for all controls: `WS_CHILD | WS_VISIBLE`.

If you want to turn off one of the default styles for a control, you do so with the `NOT` keyword. For example, if you write

```
   EDITTEXT IDC_AWESOME, 10, 10, 100, 100, ES_MULTILINE | NOT WS_VISIBLE
```

the resource compiler starts with the default style of

```
dwStyle = WS_CHILD | WS_VISIBLE | ES_LEFT | WS_BORDER | WS_TABSTOP;
```

then it adds `ES_MULTILINE`:

```
dwStyle |= ES_MULTILINE;
// dwStyle value is now
// WS_CHILD | WS_VISIBLE | ES_LEFT | WS_BORDER | WS_TABSTOP | ES_MULTILINE
```

and then it removes `WS_VISIBLE`:

```
dwStyle &= ~WS_VISIBLE;
// dwStyle value is now
// WS_CHILD | ES_LEFT | WS_BORDER | WS_TABSTOP | ES_MULTILINE
```

which is the final style applied to the control.

The resource compiler is trying to help you out by pre-setting the styles that you probably want, but if you don't realize that those defaults are in place, you may not realize that you need to provide them yourself when you don't use the resource compiler. Maybe it was being *too* helpful and ended up resulting in helplessness.

The customer was kind enough to write back.

> Thanks! That did the trick.

For completeness, the default dialog style is `WS_POPUPWINDOW = WS_POPUP | WS_BORDER | WS_SYSMENU`. If you have a custom font, then you also get `DS_SETFONT`, and if you have a caption, then you get `WS_CAPTION`.

Raymond Chen

**Follow**