

Re: OOP style

Source: <http://coding.derkeiler.com/Archive/Delphi/comp.lang.pascal.delphi.misc/2005-08/msg00362.html>

- *From:* "Maarten Wiltink" <maarten@xxxxxxxxxxxxxxxxxxxxx>
 - *Date:* Tue, 30 Aug 2005 17:06:06 +0200
-

"swansnow" <schultz@xxxxxxxxxxxxxxxxxxxxx> wrote in message news:1125405979.983216.22920@xxxxxxxxxxxxxxxxxxxxx
>> The root of all evil, if not avoided, is putting functionality
>> into event handler methods of forms.
>
> I'm interested in hearing more about this.
[...]
> At some point, calling functions starts feeling like a Choose Your Own
> Adventure. You go look to see what this method is doing, and it sends
> you to another one, and then you get sent to another one... It's hard to
> build a mental model when you keep following references like that. Or
> is it that I'm just a victim of bad design?

Top-down design can look like that. Like recursion (and what you describe is much like mutual recursion), it should "bottom out" somewhere. While the actual call tree can become a few dozen nodes deep, procedures should be named such that you don't need to read into every function called to understand what it does. That should be obvious from its name.

But the first level is more or less mandatory in what Bjørge proposed. Because there, `_form_` events are mapped, forwarded, translated, to `_application_` events.

Say that you have a property dialog, where you edit the properties of an object that you have loaded into the form. Every property of the object can be thought of as reflected in the dialog, which shows a control for it. The dialog knows which object you're editing in it, but doesn't immediately write back everything you type or click because you might still cancel. It also has an Apply button, which starts out disabled and becomes enabled when you change anything in one of the dialog's controls.

The dialog has a property for every property of the object, and all those values are kept in the controls. There is also an extra property "Dirty", a flag, which is set whenever changes are made. The Apply button's enabled state follows the dirty flag, or perhaps `btnApply.Enabled` is even used to store the Dirty property's value.

OnChange handlers are added for all the controls. The naive approach would be to simply put `"btnApply.Enabled:=True;"` in all of them. That way

Re: OOP style

madness lies, and ad-hockery, and the root of all evil as stated by Bjørge. Because you are piling spaghetti on top of the GUI. Before long, all the dependencies will become too much to keep track of.

What you do is `_abstract_`, think in terms of application events rather than GUI events, and write `"Dirty:=True;"` instead. That allows you to have a write accessor method for the Dirty property that calls an event dispatcher when its value is toggled. That event dispatcher enables or disables the apply button as appropriate, and can easily be extended to do other things like putting the tell-tale asterisk in the title bar.

Without that single place to write everything, if you ever added behaviour to the toggling of the (non-existent) dirty flag, you'd have to go through all your code to find all the places where the apply button was enabled, and hope you don't miss any. The write accessor also makes it easy to raise the event only when the value has actually changed.

Without that single place to write everything, there will be fewer function calls in your code (`"Dirty:=true;"` is really a function call to the property accessor method, and I often have `Soil` and `Cleanup` methods, too). But your code will not be clearer. Far from it.

Groetjes,
Maarten Wiltink

• *References:*

- ◆ *OOP style*
 - ◇ *From: swansnow*
- ◆ *Re: OOP style*
 - ◇ *From: Bjørge*
- ◆ *Re: OOP style*
 - ◇ *From: swansnow*
- Prev by Date: *Re: having difficulty getting the unit to use an include file*
- Next by Date: *Re: HTTPS using Indy*
- Previous by thread: *Re: OOP style*
- Next by thread: *Re: OOP style*
- Index(es):
 - ◆ *Date*
 - ◆ *Thread*