Subject: Re: Objects with Widgets, Save/Restore
Posted by John-David T. Smith on Fri, 03 Aug 2001 22:27:33 GMT
View Forum Message <> Reply to Message

David Fanning wrote:

```
>
> JD Smith writes:
>
>> The secret is this same notion of pruning out the bits you don't need.
>> Here's a sketch of how I'd approach your problem.
>>
>> pro DavidsPlayhouse::Restore
>>   oldself=self              ;for killing later
>>   wInfo=self.wInfo          ;save the GUI, eat soy products
>>   self.wInfo=ptr_new()      ;detach, avoiding the carnage
>>   restore_obj, self.SaveFile ;travel back in time
>>   obj_destroy,oldself       ;kill our old self, except wInfo
>>   self.wInfo=wInfo          ;reattach our saved widget info
>>   self->UpdateGUI           ;I'm not who I think I am
>> end
>
> Humm. Since I'm within hours of delivering
> the first truly production version of this
> code, I'm loath to open up this topic again.
>
> But I think now what got me into trouble
> is that inside my larger application, which
> is written as an object, there was an
> ROI Window Object. This is a compound widget
> (window) that doesn't have any notion of what's
> inside it, but it can draw various types of
> ROI's on itself.  This too is written as an object,
> and it reports its results (perimeter points,
> interior indices, etc.) to the event method handler
> of the parent widget object when the ROI
> is completely drawn. Of course, it relies on
> widget identifiers to communicate with its
> "parent" properly. And it has it's own widget
> identifiers to deal with.
>
```

Probably you'd just have to use this method to do some more aggressive pruning of your (obviously more complex) class structure. If your objects contains objects which themselves are quite tricky, you might consider a "Prune" and "Reattach" method in each, which readies an object for saving by clipping out the sensitive widget portions, and puts them back in place upon restoration. The question of course becomes, where do you put that sensitive information (or, more probably,

the pointer to it), in the meantime. You can't put it any place in the
object itself, since it is about to be overwritten from disk!

In my simple solution, all pruning was controlled from a single calling
level, and it was enough for me to save everything in a method local
pointer variable (wInfo) to survive the restore. In other situations,
in which objects contain objects which contain objects, all of which
will potentially be saved to disk in a big fat file, you have to get
more clever.

I would seriously consider something like a system variable or a common
block for this rare operation, but if the real paranoia even those words
instill in us cannot be overcome, you could always collect up the
pointers and propogate them upwards to the top-level Prune method, which
does the restore, and then pushes them back down through the stack of
Reattach methods. If you keep everything in order between Prune and
Reattach, you should be able to pair up all the correct wInfo's (etc.)
with all the correct objects. The best structure for this would of
course be a pointer tree, the creation of which is a rewarding exercise
in its own right. The main object would create the head of the tree --
a node of form, say

```
 superinfo={INFO_TREE, sibling:ptr_new(), child:ptr_new(),$
     info:ptr_new()}
```

This would create and hand down superinfo.child (with its sibling
pointer suitably set), to the first child, and etc., all the way down.
The giant info tree could be lifted up over the restoration of the
nested objects, and then pushed back down through the tree in the same
fashion for reattachment. Voila.

In fact, now that I think of it, there's no reason there couldn't be a
generic superclass SaveRestore which hides this tree logic, and makes it
simple for objects to perform this dance. Probably not in time for you
deadline though ;)

JD