

Re: Data access from multiple code modules

Source: <http://coding.derkeiler.com/Archive/Python/comp.lang.python/2006-07/msg01713.html>

- *From:* Bruno Desthuilliers <onurb@xxxxxxxxxxx>
 - *Date:* Thu, 13 Jul 2006 13:27:01 +0200
-

simon.hibbs@xxxxxxxx wrote:

Bruno Desthuilliers wrote:

Do you mean the code effectively doing these operations is in the gui ?
If yes, it would be better to factor it out IMHO.

The GUI has to be able to access the data object, otherwise how does the user affect any changes to the application data? If I modify a value in a text box and hit Apply, that change must be propagated to the data model somehow.

Yes, obviously. But that's not what I said. My advice is to separate the GUI code (view) from the code operating on data (controller), so operations are independent from the GUI code itself. Of course the view needs to call back on the controller(s), and the controller(s) need to have a reference to the model.

Similarly, the GUI must access the data to display it.

Yes, obviously. But here again, it may be better to just have the GUI call back on the controller(s) to get needed data.

Actually this has nothing to do with code modules. The problem is you have one object that contains your data, and your app consists of several objects that need to interact with it, how do these other objects access it?

I'm fairly new to OOP and Python so I don't know it's scoping rules very well.

Simple example:

Re: Data access from multiple code modules

One way would be to use the database as the point of contact for all the object. Every object that wants to read or write to the database instantiates an object to interact with the database and performs whatever actions it needs.

If by "any object", you mean "any GUI widget", then this is how languages like VB and Delphi (and other "DB->GUI pipeline" tools) work, and it's an approach that is known to have some drawbacks.

That way each top-level object will contain its own instanced database IO object. Their only point of contact with each other is the database itself. This could lead to synchronisation problems though.

Another way would be to create one instance of the database IO object and then pass it as a parameter to the other application objects when they are created. The point of contact between the application objects would be the single database IO object, with the database behind it. This seems to me to be the superior approach.

I don't know enough of your "database IO object" and "other application objects" is a too vague definition, so I can't really comment on this, but it seems like we are in the same situation as above.

Are there any other architectural options that anyone could suggest?

Separating operations on data (model/controller) from GUI code (view). The controller(s) have a reference on the model. The views have a reference on the controller(s), and call on the controller to get data to display or act on data.

My 2 cents

--

bruno desthuilliers

```
python -c "print '@'.join(['.'.join([w[:-1] for w in p.split('.')]) for p in 'onurb@xxxxxxxxxxx'.split('@)])"
```

.