

My Application Object – Ban Global Vars!

by Jason Chapman

There have been some rumblings in the Developers Group newsgroup about what people want from the group and how to appeal to as many people as possible. One thing that raises its head from time to time is good programming techniques. I am used to going onto sites to do either trouble-shooting, mentoring or QA. As a fairly bright bloke, I judge the quality of the software team by how quickly I can pick up the code and work out what is going on (from the code, documentation etc). This is fairly presumptuous of me: it assumes I am able to pick up all manner of projects from any problem and technology domain and that my brain is as happy with one methodology as another.

The subject of this article is a little something that I have been using since Delphi 1 and can be seen in most of the (non-trivial) projects I write: it is my application object (FAppObj as I refer to it). I have read books, most of which I have forgotten, as well as white papers, so to everyone I may have plagiarised, consider yourself acknowledged – thank you; if this does not suffice, contact me and I will apologise further. This is my solution to a problem, please read and give me some feedback; I am always searching for the best way to solve problems.

Among the facets of a development that increase the complexity hugely are global variables or global objects. In any Delphi application there are a number of them, but ones that spring to mind are Application : TApplication and Screen : TScreen. I like these. Others like ShortDateFormat: string don't light my fire and fall into part of the problem and not an elegant solution.

Global objects / variables are visible from anywhere in the application (that uses the unit): the intention is that it is something you need everywhere, so make it available everywhere. The problem comes when you have a lot of these and their usage is littered throughout your application. Many units read the variables and some change their values. This is bug city, just waiting to bite you. If you want to find all occurrences, you do a grep search (I love my GExperts Grep). Before you know it, they are being used for all kinds of stuff, e.g. isAdminUser, colorForEdits, debugFlag. When they pop up, you generally nip off, find the declaration, look for comments or usage within the application and get back to what you were trying to solve. To me, they have smacks of bodgery (if there were such a word), "I need this here, but I need to set it there, oh let's have a global variable".

My first solution is to have a big carrier bag, nah, a class to group all these global objects together, then split them into smaller classes/objects that have members/methods attributes that "go together". Examples of these may be userName, usersForename, usersSurname. To some this is all second nature and you have no idea why anyone would be writing an article about it; others - and I know there are others as I have reviewed thousands of lines of their code - don't naturally get this or feel that it is a pain to have to produce extra code in creating classes etc to make it all fit together.

So, to recap, we have gone from having a large number of global variables to having objects that collect them together. As soon as you have moved away from global variables into global objects, your code makes more sense. Consider:

```
result:=FAppObj.UserObj.EmailAddress
```

What do you think you might need to do to find other information or available data about the currently logged-in user? Ctrl+Space on UserObj and you are away. I realise that I have missed a few steps with this; where did FAppObj suddenly appear from?

FAppObj is like "Application : TApplication", but specific to this application and this problem domain. All my applications have them, but some have more associated information than others. The other difference is that I do not surface this as a global object; instead it is passed around as objects are initialised. I prefer this as it sets out a strict contract and makes the code very easy to follow, even if it is printed out on paper. Here is the application object from my current and ongoing pension project. It is currently in Delphi 7:

```
TApplicationControl = class(TComponent)
protected
    FElectedUserObj : TthUser;
```

```

public
    UserObj : TjcUserLogged;
    dmAudit : TdmAuditor;
    dmSimpleLookups : TdmSimpleLookups;
    dmSimpleLookupsMKII : TdmSimpleLookupsMKII;
    dmMainFibDB : TdmMainFibDB;
    MainDBName : String;
    sysParams : TjcSysParams;
    frmAppBookmarks : TfrmAppBookmarks;
    function ElectedUserObj : TthUser;
    procedure SetElectedUserObj(vuserId : Integer);
public
    destructor destroy; override;
    constructor create(AOwner: TComponent); override;
end;

TjcSysParams = class(TObject)
public
    FDocDir : String;
    FWordTemplateDir : String;
    function tokenToStr(vtoken : String) : String;
end;

```

I haven't tinkered with this to make it perfect or beautiful; this one started life before my millennium celebrations by about 3 years, so it has evolved. The last member added was ElectedUserObj, but I'll come onto that. When it gets a bit out of hand or I add something that doesn't fit well, it gets a bit of a makeover. I have a list of things I would like to do to make it a bit more elegant. Each one of their objects could have been rendered as a singleton (ask JoannaC, she'll rattle off the Gang of Five definition – oh you thought they were the gang of four, no twas five: the fifth wanted things like perl, cable, crochet and the like, and so got booted out). I find it difficult to justify the aggregate class over separate singletons, other than it feels right and makes me look at the sibling objects when deciding where and how a new member should be added.

OK, so let's talk about the warts. The objects are public and are not protected well (compare UserObj and ElectedUserObj): someone could go FAppObj.UserObj:=nil, but they would be dumb, right? Actually looking at this now, it could do with a bit of a tidy. I have never broken it by doing something like that, but if it were to be a shipped component or used by a large team of developers, it would pay dividends to make it a bit more bulletproof.

Maybe looking at each of the members will sell a few of you on the virtues of having an FAppObj. Also, it might whet your appetite to ask about other parts of the system, which would make other good articles.

UserObj : TjcUserLogged;

This object is instantiated when the user logs into the system; it holds information about the user, plus their security, i.e. you can ask it if a user is allowed to do something. In fact there is a TjcUser and TjcUserLogged is a descendant. TjcUser has no idea about security; it is added in TjcUserLogged. Common usage in my application:

```
FAppObj.UserObj.CanIwithEx(gSecurityArea, TroRead, className, cmethod);
```

Or

```
qryLocateOneContactUSER_ID.value:=FAppObj.UserObj.UserID;
```

The first checks whether a user can do something and throws an exception if they do not have the correct privileges. I could go on for a week on the user and security: maybe another day (MAD). The second is probably self-explanatory.

When I was discussing letter production with the staff at this company, out of the blue they said, oh Fred often produces letters for Ann and it is a pain in the neck having to log on as her. Argh! All that effort getting a good security model confounded by an oversight of mine. One hour later I had created the ElectedUserObj: a user can go into a menu option and select another user and where appropriate in the system, it will put in the elected user, e.g.

```
result:=FAppObj.ElectedUserObj.FullName
```

Now, how long would that have taken me with global variables and how many occurrences would I have missed?

BDE. dmSimplelookups is BDE aware and won't change until I can regroup it (see above [any suggestions welcome]). I needed a really neat way of creating bookmarks, so that if the Workflow module found data that was inconsistent, I could create a durable bookmark that would allow the user to go straight to that part of the system. This is so neat, it makes me wee a little bit every time I try to explain it to people (MAD). Hence dmSimpleLookupsMKII. One day I'll explain how, when I have a large class to refactor, and still maintain a working system, I use abcMKII for methods and classes, so I have old and new working methods in the same app at the same time, while I incrementally update all the other code, MAD.

Back to the plot. In my main application I need this bookmark manufacturing code and don't want to back port it to the BDE, so I currently have this object in FAppObj as well.

You must be getting bored by now, so I'll leave dmMainFibDB, MainDBName, sysParams, frmAppBookmarks for another day (MAD).

Lifecycle

This has always been a bit tricky. Different global objects need to come into being at different times. If you consider my database application, on running the application the target database is known, hence dmSimplelookups needs to come to life immediately; the application / auditing may need to know what day it is i.e. FappObj.dmsimplelookups.gettoday. However the user has not logged in, so having a user object may not be appropriate. This is where I need to spend a bit of time over the next month: looking at it now it is a bit of a mess. Things get created, injected into the FAppObj, but to get the lifecycle, you have to look at the main form, yuck. Really I think that all this should have been managed in the application objects (FAppObj), MAD.

Also, interestingly, none of the "global" objects can easily know about the application object, a limitation of Delphi, with an inelegant kludge to get it to work. How many times have I come across A needing to know about B and B needing to know about A and me having to implement the AsTABC internal function kludge? Wish they would sort that out. Just to clarify, two units cannot have each other in their interface section uses clause: the compiler raises a circular reference. You can have unit A referencing unit B in the interface section and then B using A in the implementation section, but this doesn't work well for class definitions where you want each class to refer to the other in some member.

Passing it Around

All of my major classes are "inited", i.e. they are created, then I call the init method (a custom method I add): this should pass in all that is needed for it to do its job. By having an init method and not using global variables, code becomes very easy to understand which in my opinion is well worth the extra code.

So TfrmMain creates FappObj:

```
TfrmMain = class(TForm)
..
private
{ Private declarations }
FAppObj : TApplicationControl;

if FAppObj=nil then
FAppObj:=TApplicationControl.create(self);
```

Later it creates another form and calls its init:

```
thefrmSIPPBrowser:=TfrmSIPPBrowser.create(self);
try
hide;
thefrmSIPPBrowser.init(dbSIPPWinSrc.databasesname, dbFTPrices.DatabaseName ,BrowseTitle,
FAppObj);
```

and so on and so on. When you get heavily inherited forms/frames and datamodules, you don't end up redeclaring the code all over the place as it exists in the ancestor. As for inherited frames and datamodules and why I have so many datamodules with so few datasets on each, MAD.

Conclusion

That was a little bit of a brain dump and very therapeutic to me, thank you. I hope it has raised some interesting points and you might want me to follow up on some of the points raised: let me know. Using the

application object has made my code more robust and easier to maintain. Making large-scale changes to how my systems work and step changes in functionality has been easier due to this sensible use of inheritance (MAD).

If I were asked for a couple more bits of advice I would say:

- Don't leave code in a project that is not required – cruft *[isn't that a dog show? Ed]*
- Have a framework
- Have a methodology
- Be consistent

We are on the lookout for more work at the moment: we develop software, support development teams and come in when it all goes horribly wrong. If you need a hand or you have any comments, get in touch.



Jason Chapman is a developer, mentor, consultant and trainer and sometimes a marathon runner. After working for a London based training & consultancy company, Jason worked as a contractor, then an IT Director. Jason is currently developing a pensions system (SIPP & SSAS), whilst supporting software houses and getting to grips with marketing to drum up some more business.

[This article was kindly tech-edited by Brian Long. Any editorial errors remaining are mine. Ed]
