Cool Tool

Edit Your Superclass Methods with SUPERCLASS

By Whil Hentzen

Ever had one of those days? The type of day where nothing goes right? You're working in Visual FoxPro, and you've got to look at the Language Reference for just about every command you type. For some reason, you can't remember the syntax for even the simplest commands, like WAIT WINDOW or SKIP. You even tried to do some simple data entry in a Browse Window, but couldn't figure out why CTRL-N wasn't working when you were adding new records.

Of course, according to Murphy's Law, you've probably got a ton of class and form design work to do on this type of day. As a result, you keep running into that annoying problem of not being able to work with a class and a subclass, or a form and the class used to create objects on that form. Typically, what happens is the following:

1. Grab a class library and place it on the Form Controls toolbar.
2. Create a form, and place some controls on the form using classes in the class library.
3. Realize that you have to change something about the class - some property, say, or perhaps you have to add a method.
4. Try to modify the class, and face that obnoxious error message: "Cannot modify a class that is in use."
5. Close the form.
6. Modify and save the class.
7. Open up the form again.
8. Try to remember what it was you were doing in the form that required changes to the class.

The reason that you get this message is that the appropriate record in the class library file (the .VCX table) is locked because an instance is open in the designer.

SUPERCLASS to the rescue!

Well, since there is no need for GENSCRNX 3.0 in Visual FoxPro, Ken Levy has had to keep busy doing something else. He's spent his time reading the minds of frustrated developers who are "having one of those days." One of the first results is SUPERCLASS, a utility that enables you to view and edit the methods of an object's superclass. You'll still have to close the form or class if you want to do other things (like modify properties of a superclass), but SUPERCLASS will save you hours of opening and closing the designers simply to tweak the method of a superclass.

Using SUPERCLASS is one of those operations that takes longer to explain than it takes to do. First, run SUPERCLS.PRG. I recommend you put SUPERCLASS in your developer utilities directory, or somewhere else in your path, and run it during Visual FoxPro startup. SUPERCLASS stays hidden until you open up a method-editing window in the Form Designer or Class Designer. At this point, the SUPERCLASS toolbar appears with three buttons.

*\\\ 9511TBAR.BMP: "The SUPERCLASS toolbar contains buttons for viewing/editing the method of the superclass and inserting a superclass method call."

SUPERCLASS functionality

The left-most button allows you to view or edit the superclass method of the object that you were working on - precisely what you weren't able to do before. Furthermore, it's just a mouse-click away - you don't have to open up the class designer and all that nonsense.

However, there's a caveat to be aware of when editing superclass methods. Behind the scenes, SUPERCLASS has made a temporary VCX with the record of the Methods field. When you change the superclass method, those changes are placed in this temporary file. The temporary file is then closed, recompiled, and the resultant ObjCode field contents are placed in the VCX. However, the instance of the original object is still in memory, and this doesn't get changed. As a result, if you continue to create instances of that class, those new instances will still contain the old method code. You'll want to CLEAR CLASS or CLEAR ALL, and call SUPERCLASS whenever you CLEAR ALL.

The middle button on the SUPERCLASS toolbar allows you to insert a superclass method call in the current method window at the current cursor position. However, if you're inclined not to read the documentation, you'll miss a trick. Left-clicking will insert a call to the superclass method. In other words, suppose you have a command button class called cmdBase, and in the Click() method of that button, you've placed the following code:

```
wait window "Jeepers. I'm in the Click method"
```

Then, you've placed an instance of that command button class on a form (it could be in another class, of course). Opening up the Click() method of the command button instance will bring forward the SUPERCLASS toolbar. Clicking on the middle button will insert the command

```
cmdBase::click()
```

which will execute that code (the Jeepers WAIT WINDOW) in addition to any code in the instance's method. It's just a simple timesaver, but isn't that what I promised earlier in this article?

Right-clicking on the middle button will insert the text

```
=EVALUATE(this.ParentClass+"::method()")
```

if you're editing a subclass, or

```
=EVALUATE(this.Class+"::method()")
```

if you're editing an object member of a container. If you think that a direct class or parent class might be changed, you might want to use this, since it inserts a generic reference that is evaluated correctly regardless of how the superclass reference changes.

Finally, the right-most button on the toolbar brings up the Help file. Now you don't have any excuse for reading the documentation! And a nice touch: Right-clicking on the Help button will display the version number of SUPERCLASS - an enhancement we all would do well to emulate.

Where to find SUPERCLASS

SUPERC.ZIP is public domain and can be found on this month's Companion Disk as well as in Library 9 of FoxUser and Library 3 of VFOX.

Book of the Month

This month's "must have" is the next in the series of three Microsoft Press books on software development. "Writing Solid Code" by Steve Maguire (ISBN 1-55615-551-4) is billed as "Microsoft's Techniques for Developing Bug-Free C Programs", but that shouldn't scare Fox developers off. The techniques described in these 250 pages are sometimes specific to C code, such as "Don't use output memory as workspace buffers" and "Use LINT to catch bugs that your compiler may miss", and the examples are all in C, but it would be hard to turn two pages in a row without picking up a valuable idea.

Maguire starts out by discussing how bugs come about. He proposes that a programmer ask two questions about every bug they encounter: "How could I have prevented this bug?", and "How could I have automatically detected this bug?" He then guides you through assertions, code step throughs, bug-proof functions, methods to code safer algorithms, risky programming practices, and finally covers attitude. "If a programmer regularly 'cleans up' code or simply 'tries' haphazard solutions to problems, hoping to hit upon something that works, writing bug-free code will be an uphill battle."

This last chapter is perhaps the most valuable. In it, he describes a number of real-life events, including the cancellation of an unannounced Microsoft product due to a runaway bug list, and what techniques could have been used to avoid those situations. This book is an easy read, but may well be one of the most valuable books on software development that you can own.