odifying VFP's Wizards and Builders

Doug Hennig

The builders and wizards that come with VFP often don't do quite what you need them to. This month's article shows how to remedy this.

If you're like me, you probably haven't used the builders and wizards that come with VFP very much because they don't quite work the way you want them to. Maybe they don't have enough flexibility or maybe you just don't like the way they do something. Until VFP 6, there was no way to change the behavior of the builders and wizards because Microsoft didn't provide the source code. However, now we get the source code for not only all the builders and wizards, but also the Class Browser, Component Gallery, and Coverage Profiler.

Source code for builders and wizards can be found in XSOURCE.ZIP in the TOOLS\XSOURCE directory under the VFP home directory. When you unzip this file, it creates a VFPSOURCE directory under which all the source code exists. Wizards are found in subdirectories of the WIZARDS directory and builders in subdirectories of BUILDERS (although some common files located in WIZARDS are used by some builders).

So, now that we have source code, we can change the builders and wizards to do what we want, right? Well, a better approach is to create new builders and wizards that use the majority of the classes and programs of the originals but override their behavior by subclassing classes and either cloning and modifying PRGs or creating wrapper PRGs. This month's article will detail exactly how to do this for a couple of the builders that come with VFP, the Grid Builder and Referential Integrity Builder.

Once you've created your replacement builder or wizard, how to you tell VFP to use it rather than the native one? Builders are registered in BUILDER.DBF and wizards in WIZARD.DBF, both of which are in the WIZARDS subdirectory of the VFP home directory. These tables have the same structure, which is shown in Table 1.

Field	Description
NAME	The descriptive name of the builder or wizard.
DESCRIPT	A description for the builder or wizard.
BITMAP	This field appears to be unused.
TYPE	What type of object the builder or wizard is for. In the case of BUILDER.DBF, it's generally the base class for an object (although there are also records with MULTISELECT, AUTOFORMAT, and RI in this field). For WIZARDS.DBF, it's things like FORM, REPORT, and QUERY.
PROGRAM	The name of the APP file containing the builder or wizard.
CLASSLIB	The class to instantiate in the APP file.
CLASSNAME	The class library of the main class in the APP file.
PARMS	Parameters to pass to the builder or wizard.

Table 1. Structure of BUILDER.DBF and WIZARD.DBF.

When you invoke a builder, VFP calls the program specified in the _BUILDER system variable (BUILDER.APP by default). BUILDER.APP looks at the environment it's in (for example, which object the builder is being invoked for), looks for a record that matches the environment in BUILDER.DBF (for example, it looks for the base class of the object in the TYPE field), and invokes the registered builder. Wizards are treated the same way, except the system variable is _WIZARD, which points to WIZARD.APP, which looks in WIZARD.DBF.

To tell VFP to use a different builder or wizard, insert a new record into BUILDER.DBF or WIZARD.DBF that describes how to run your builder or wizard. If you want to have the choice of running the old builder or wizard as well as your new one, leave the record for the original builder or wizard alone. If you want to replace the old one, rather than deleting the original record, simply change its TYPE value to something that will never be recognized (I use a suffix of "X", such as "XGRID"); that way, you can simply restore the former builder by changing its TYPE value back.

Creating a More Useful Grid Builder

The VFP Grid Builder provides a quick way to populate the columns of a grid and create the visual appearance you want. However, I can think of several things I don't like about the builder:

- It doesn't size the columns it creates properly. You have to manually size them to fit the data.
- The control type combo box on the Layout page of the builder lists VFP base classes and classes already existing in columns; there's no way to add your own class to this list.
- The columns and headers it creates are VFP base classes. You might want to substitute your own classes (which must be defined programmatically) to, for example, sort on a column by clicking on the header.

To create a replacement for the VFP Grid Builder, I first created the SFGRIDBLDR project (it's in the GRID subdirectory when you unzip the sample files available from the Subscriber Download Site) and added the following files: BUILDER.VCX (in the BUILDERS\BUILDERS subdirectory of the VFP wizard source directory), GRIDBLDR.VCX (in the BUILDERS\GRIDBLDR directory), THERM.VCX, WIZCTRL.VCX, WIZMOVER.VCX (all in WIZARDS\WZCOMMON), DUMMY.PRG, and WBGRID.PRG (both in BUILDERS). How did I know which files to add to this project? Simple: I just looked at the contents of the GRIDBLDR project.

Next, I subclassed the GridBuilder class into SFGridBuilder (in SFGRIDBLDR.VCX) and overrode the ResetColumns method, which sizes the columns appropriately for the selected field (I only implemented this idea, not the other two listed above). The code for that method is listed below. There are a couple of interesting things to note here. First, I expected to write a lot of complex code to figure out how wide a column should be, based on the width of the field, the font and font size of the grid, etc. Interestingly, it turns out that a method to do this (SetColWidth) already existed in the builder, but rather than passing this method the size of the field, the builder passed it a different value. The other thing is that currently the assignment of the calculated width to loColumn.Width is commented out. For reasons I haven't tracked down yet, changing the width of the column at this point seems to cause a problem in that when the builder is closed, the column width is set to 0. Instead, the width is stored in wbaCols. (In case you're wondering, yes, wbaCols is a public array. I didn't create the builder, so don't blame me for doing it this way!) So, the effect is that when a column is added to the grid, it isn't sized properly immediately, but is as soon as you do anything else (add another column, go to another page in the builder, close the builder, etc.).

```
local lnRows, ;
  lnI, ;
  lcField, ;
  loColumn, ;
  lcHeader, ;
  loHeader
dodefault()
lnRows = alen(wbaCols, 1)
for lnI = 1 to lnRows
  lcField = wbaCols[lnI, 2]
  if not empty(lcField)
    loColumn = evaluate('wbaControl[1].' + ;
      wbaCols[lnI, 7])
    wbaCols[lnI, 1] = This.SetColWidth(fsize(lcField), ;
      loColumn)
*   loColumn.Width  = wbaCols[lnI, 1]
  endif not empty(lcField)
next lnI
```

Finally, I created GRIDMAIN.PRG (using GRIDMAIN.PRG from the BUILDERS\GRIDBLDR directory for ideas) in the directory for my builder and made it the main program in the project. This program adds SFGRIDBLDR.VCX to the open class libraries using SET CLASSLIB so it can find our SFGridBuilder class. GRIDMAIN also auto-registers itself in the VFP BUILDER table if the APP is run directly. Here's the code for this PRG:

```
lparameters tu1, ;
  tu2, ;
  tu3, ;
```

```
  tu4, ;
  tu5, ;
  tu6, ;
  tu7, ;
  tu8, ;
  tu9, ;
  tu10, ;
  tu11, ;
  tu12
local lnSelect, ;
  lcClassLib, ;
  lcLibrary

* Define the class library and VFP builder/wizard.

#define ccCLASSLIB  'SFGridBldr'
#define ccBUILDER   (home() + 'WIZARDS\GRIDBLDR.APP')
#define ccMAIN      'GRIDMAIN'

* Auto-register the builder if it's called directly.

if program(0) == ccMAIN
  lnSelect = select()
  select 0
  use home() + 'wizards\builder' again
  locate for NAME = 'Stonefield Grid Builder'
  if not found()
    locate for NAME = 'Grid Builder'
    scatter memvar memo
    replace TYPE with 'XGRID'
    insert into BUILDER from memvar
    replace NAME with 'Stonefield Grid Builder', ;
      PROGRAM with sys(16), ;
      CLASSLIB with 'sfgridbldr.vcx', ;
      CLASSNAME with 'sfgridbuilder'
  endif not found()
  wait window 'SFGridBldr now registered as the ' + ;
    'Grid Builder'
  use
  select (lnSelect)
  return
endif program(0) == ccMAIN

* Get the current CLASSLIB setting, and add files we'll
* need at the start.

lcClassLib = ccCLASSLIB
lcLibrary  = set('CLASSLIB')
lcLibrary  = iif(empty(lcLibrary), '', ',' + lcLibrary)
set classlib to &lcLibrary &lcClassLib

* Run the "real" builder program.

do ccBUILDER with tu1, tu2, tu3, tu4, tu5, tu6, tu7, ;
  tu8, tu9, tu10, tu11, tu12

* Close the files we opened.

release classlib &lcClassLib
```

To see this builder in action, build and run SFGRIDBLDR.APP to register it as the builder for grids. If you look at BUILDER.DBF after running this APP, you'll find that it's "deregistered" the original grid builder by changing the TYPE column from "GRID" to "XGRID" and added a new record for itself with "GRID" as the TYPE. Next, create a form, drop a grid on it, and invoke the builder. The new builder won't look any different than the old one, but when you add fields to the grid, you'll see that they're properly sized.

Creating a Better Referential Integrity Builder

I have a few problems with the Referential Integrity (RI) Builder that comes with VFP:

- When you click on the OK button to save the RI rule changes, it asks you not once but twice to confirm that you want to go ahead and do this. Not to be snippy, but I believe the purpose of the Cancel button is to allow me to back out; I clicked on the OK button because it was OK, so I don't need a "are you really, really sure you want to do this" confirmation dialog.
- You can't run the RI Builder unless you've packed the database first.
- I really don't need it to back up my current stored procedures to a file called RISP.OLD, which I always delete anyway (or forget to delete and then end up backing it up by mistake).
- The code it generates has at least one bug that prevents it from functioning correctly. For details on the bug, see my article, "Handling Specific Errors", in the February 1998 issue of FoxTalk or the white paper "Error Handling in VFP", available from the Technical Papers page of my Web site (www.stonefield.com).
- The code it generates is bulky and poorly commented. Trying to understand what this code does is a laborious process, and it's possible in a complex database to have RI code that exceeds the 64K limit of compiled programs in VFP. If you purchased "Effective Techniques for Application Development with Visual FoxPro 6.0" by Jim Booth and Steve Sawyer (available from Hentzenwerke Publishing, www.hentzenwerke.com), you have a copy of a faster, leaner, and better routine for maintaining RI. Steve's NEWRI routine is data-driven, so it determines at runtime rather than generation time which rules need to be enforced. The result is clear, tight code rather than the unwieldy bulk produced by the VFP RI Builder.
- The only rules it supports are ignore, cascade, and restrict. What about other options you might want to use, such as nullify (set the foreign key in the child to .NULL.) or assign a new value (set the foreign key to, for example, a default value)?

Fixing these items is relatively easy because we have the source code for the RI Builder. I created SFRIBUILDR.APP, a replacement for the VFP RIBUILDR.APP. I didn't implement additional rules (the last item in the list above) but I did handle all the rest. I first created the SFRIBUILDR project (it's in the RI subdirectory when you unzip the sample files available from the Subscriber Download Site) and added the VFP RIBUILDR.VCX (located in the BUILDERS\RIBUILDR subdirectory of the VFP wizard source directory).

Next, I subclassed the VFP RIBuildr class into SFRIBuildr in SFRIBUILDR.VCX. I overrode the Load method to not give an error if there are any deleted records in the database (you can see where I commented out the existing code). I overrode the Click method of the OK button to not display confirmation dialogs, not copy the current stored procedures to RISP.OLD, and to fix the bug in the generated code. Also, if it detects NEWRI.PRG on your system (in the same directory as the SFRIBUILR.APP), it will place that code into the stored procedures of the database rather than generating any code. This change required a new method, RIMakeNewTr, to create triggers with a different name (__RI_Handler) than the ones used by the RI Builder (__RI_<action>_<table>, such as __RI_Delete_Customer). Because there's quite a bit of code in these methods, it isn't shown here; however, if you look at it in the source code provided, you'll see that I really only commented out a few lines and added a few other lines.

Finally, I copied RIMAIN.PRG from the BUILDERS\RIBUILDR directory to the directory for my builder, added it to the project, and made it the main program. I modified this PRG to point to the library where my subclass of the VFP RI builder class can be found (SFRIBUILDR.VCX) and to auto-register the builder in the VFP BUILDER table if the APP is run directly. Here's the code for the portions of this PRG that I changed:

```
#DEFINE C_VCX        "ribuildr.vcx"
*** DH: added another VCX definition
#DEFINE C_VCX2       "sfribuildr.vcx"
*** DH: changed main program name to RIMAIN.
*#DEFINE C_MAIN      "CMDMAIN"
#DEFINE C_MAIN       "RIMAIN"

*** DH: corrected a bug in the parameter list
*PARAMETERS p1, p2, p3, p4, p5, p6, p7, p8, p9, p10, ;
  p11, p12, p12, p13, p14, p15
```

```
PARAMETERS p1, p2, p3, p4, p5, p6, p7, p8, p9, p10, ;
  p11, p12, p13, p14, p15, p16

PRIVATE m.wbReturnValue, m.cParmstring
LOCAL wbi
IF PROGRAM(0) == C_MAIN
  *- called directly, so fail
*** DH: made it auto-register ourselves if we're
*** called directly.
*  =MESSAGEBOX(C_BADCALL1_LOC)
  lnSelect = select()
  select 0
  use home() + 'wizards\builder' again
  locate for NAME = 'Stonefield Referential ' + ;
    'Integrity Builder'
  if not found()
    locate for NAME = 'Referential Integrity Builder'
    scatter memvar memo
    replace TYPE with 'XRI'
    insert into BUILDER from memvar
    replace NAME with 'Stonefield Referential ' + ;
      'Integrity Builder', ;
      PROGRAM with sys(16), ;
      CLASSLIB with 'sfribuildr.vcx', ;
      CLASSNAME with 'sfribuildr'
  endif not found()
  wait window 'SFRIBuildr now registered as the ' + ;
    'RI Builder'
  use
  select (lnSelect)
*** DH: end of new code
  RETURN
ENDIF

*** DH: included our VCX in the list
*SET CLASSLIB TO C_VCX ADDITIVE
SET CLASSLIB TO C_VCX, C_VCX2 ADDITIVE
```

Building and executing SFRIBUILDR.APP results in this file being registered as the RI Builder instead of the usual RIBUILDR.APP. To see this in action, do the following:

- Move to the RI directory of the sample files for this article.
- If you have Steve's NEWRI.PRG, copy it into this directory.
- Run COPYDEMO.PRG. This program will copy the VFP TESTDATA database to a DATA subdirectory so we don't touch the original database.
- Run DELETETEST.PRG. This will demonstrate a flaw in the code generated by the VFP RI Builder. The first browse shows several orders for the ALFKI customer, then tries to delete that customer. Because there's a cascade delete rule from CUSTOMER to ORDERS but a restrict rule from ORDERS to ORDITEMS, the customer shouldn't be deleted. However, notice that the error dialog appears six times (not just once as you'd expect) and then another browse shows that the customer has been deleted (CUST_ID is .NULL.).
- Run COPYDEMO.PRG again since we've messed up the data.
- Open the DATA\TESTDATA database exclusively.
- Build and run SFRIBUILDR.APP to register it as the RI Builder.
- Open the Database Designer, choose the Edit Referential Integrity function, and then simply click OK in the RI Builder dialog. Notice no confirmation dialogs appeared. Choose the Edit Stored Procedures and notice that the code is different (if you have NEWRI.PRG, this code is placed in the stored procedures; otherwise, the RIDelete and RIUpdate methods have a bug fix implemented). If you have Steve's NEWRI.PRG, modify the CUSTOMER table and note the name of the trigger for each method. Also, you won't find an RISP.OLD file.
- Run DELETETEST.PRG again. This time, you only get the error message once and the ALFKI customer isn't deleted.

Conclusion

Creating your own versions of the builders and wizards that come with VFP is really pretty easy; the hard part is the detective work necessary to see where you need to make the changes. I hope you find the two replacement builders discussed in this article useful, and that you're inspired to create your own replacements so these tools can do exactly what you need them to.

Doug Hennig is a partner with Stonefield Systems Group Inc. in Regina, Saskatchewan, Canada. He is the author of Stonefield's add-on tools for FoxPro developers, including Stonefield Database Toolkit and Stonefield Query. He is also the author of "The Visual FoxPro Data Dictionary" in Pinnacle Publishing's "The Pros Talk Visual FoxPro" series. Doug has spoken at the 1997, 1998, and 1999 Microsoft FoxPro Developers Conferences (DevCon) as well as user groups and regional conferences all over North America. He is a Microsoft Most Valuable Professional (MVP). He can be reached at dhennig@stonefield.com.