April, 2001

Advisor Answers

Duplicating VFP Wizards

VFP 7.0/6.0

Q: I am working on a project that requires an import wizard similar to the one supplied with Visual FoxPro. I have a fixed one-to-many database that the data ends up in and am trying to make the program as easy for the user as possible.

I have duplicated the wizard completely in Visual FoxPro except for one item. It is step2a where the user can insert vertical lines on the grid with mouse clicks. The vertical lines represent the field delimiters. As a work-around, I have created a grid where the user manually enters the column positions and widths. However, the Microsoft solution is much simpler and cleaner from a user prospective.

Perhaps there is an ActiveX control or .DLL file that can do this? Perhaps there's a feature in FoxPro that I am not aware of?

–Ed DiCampli (via Advisor.Com)

A: One of the great things about working with VFP is the openness and extensibility of the product. Most of the key data in VFP (including form, class, menu, report and project definitions) is stored in tables, which gives us access to it outside the Designers.

Many of the VFP tools provide built-in extension techniques. For example, the Class Browser, Component Gallery and Coverage Profiler (as well as VFP 7's new Object Browser) allow users to hook on add-ins. The Wizard and Builder systems are table-driven so that you can add new wizards and builders just by registering them.

In addition, a number of the utility programs provided are specified by using a system variable, so that you can replace the program provided with your own. For example, the _GENXTAB variable indicates the program to use for generating cross-tabs. In VFP 7, the set of programs specified this way grows again with the addition of the _TASKLIST and _CODESENSE variables to specify the Task List and Intellisense Manager applets, respectively.

But even having built in all this openness, the VFP team goes farther. Source code is provided for a number of the tools that are built in VFP.

We've always had source for some items, such as GENMENU, the program that turns our menu designs into programs and GENDBC, a utility that creates a program to regenerate a database.

Beginning with VFP 6, the source code for the builders and wizards (as well as a number of other VFP components) is also provided. Since the full source is more than 11MB, it's provided in ZIP format. You'll find the file in the Tools\XSource subdirectory of your VFP installation. Unzip it to create a VFPSource subdirectory containing a number of its own subdirectories.

Once you expand the files, you'll find the code for the Import Wizard in the Wizards\WZImport subdirectory. The bulk of the work is done in the ImportWizard class of the Import class library.

Examining the ImportWizard class isn't easy. It uses a tabless pageframe (actually, that's how VFP's wizards work in general), which makes it hard to see what's going on. To make matters worse, many of the objects in this class still have their default names (like PageFrame1 and Text1) instead of meaningful names.

However, by working back and forth between the wizard itself and the class (you may find it helpful to run the wizard in one instance of VFP while examining the class in another), we can eventually track down the objects that provide the functionality you're interested in. Digging into the form, we find that the first page contains another tabless pageframe. Page 2 of this inner pageframe contains yet another pageframe, and page 2 of that one contains the objects we're interested in. The key object is:

```
Form1.pageframe1.page1.pageframe1.page2.;
pgfStep1a.page2.shpClick
```

This object is the one that's filled with text (actually, that's an illusion) and on which you can drop dividers and drag them around. It contains the key code that manages the dividers – look at the MouseDown, MouseMove and MouseUp events.

A little more exploration determines that a set of labels (Wizlabel1 through Wizlabel4 on the same page) contain the actual text that's displayed. I figured this out by sending the shape to back and then clicking in the text area. (Be sure when you do this sort of thing that you don't save your changes.)

There are also controls to provide scrolling, as well as all the labels and so forth. Overall, the construction of this functionality is not

simple. I'll leave it to you to dig out exactly the pieces you need to complete your form. (I wouldn't just copy the code that's there since it's so complex and hard to follow. If I were writing it, I'd use some custom methods to make the code more comprehensible.) You may find it helpful to use the Class Browser's View Code option to provide you with a complete listing of the code for this class (and perhaps do the same for some of the classes it uses).

The key lesson here is that there's a treasure trove of code and tools provided with VFP. (In addition to everything mentioned above, don't forget about all the code in the FoxPro Foundation classes and in the Solutions examples.) Before you go off to duplicate existing functionality, make sure you don't already have it available for either subclassing or simply "borrowing" the parts you need. (Be careful about license issues when you take this approach. Check out the topic "Removing Restricted Visual FoxPro Features and Files" in Chapter 25, "Building an Application for Distribution" in the Programmer's Guide. Note also that the file License.TXT referenced there doesn't exist for VFP 6, but there is a file called Redist.TXT in the COMMON\REDIST folder of the CD 1 of the Visual Studio installation set.)

–Tamar