August, 2006

Advisor Answers

Using a framework

VFP 9/8/7

Q: I keep hearing about frameworks. What is a framework and why would I want one?

A: The term "framework" is used in several different ways in the computing world, but in the Visual FoxPro community, it generally refers to a set of class libraries that provide the foundation for an application. Although each application you write is different, most applications require some basic things, such as conflict resolution, user management, and error handling. A framework provides those basics (and often, much more) so you can concentrate on the specifics of your application.

Similarly, within an application, you want consistency so that data entry forms work pretty much the same way regardless of what data they address, and the technique for getting a report is the same across the board, and so forth. Again, a framework makes this easier.

The truth is that most applications use a framework. The difference among them is whether the framework was written by the developer or purchased from a third party, and whether the framework was planned or just happened.

Since it's clearly better to plan the architecture of an application than to let it happen haphazardly, the real question is whether to buy a framework or write your own. As with so many things in FoxPro, the answer is "it depends."

There a number of commercial general purpose frameworks available for VFP. The list includes Visual FoxExpress, Visual MaxFrame Pro, Mere Mortals, Visual Extend, Visual ProMatrix, Codebook and Codemine. VFP 6 and later also include a framework; it's used by the Application Wizard. The various frameworks have lots in common and lots of differences; you'll find a comparison of a number of them at http://fox.wikis.com/wc.dll?Wiki~FrameworkFeatureChart.

The commercial frameworks typically include classes for handling data (usually, an n-tier structure), several basic form classes, a set of

useful controls, a structure for user management and security and an error handler. Many also include integration with other tools (such as Stonefield Database Toolkit).

You can also build your own framework (a task I'm in the midst of right now), but it's not for the faint of heart. A complete framework includes a lot of code and requires a lot of testing.

The way you use a framework varies with the framework. Some provide a comprehensive set of wizards and builders that guide you through much of the work of creating an application. Others offer just the class libraries together with documentation and you handle construction manually.

So why would you use a commercial framework? There are a number of reasons.

Frameworks save time. Although you have to invest a fair amount of time becoming familiar with the framework, in the long run, it's time well spent and probably much less than it would take you to build the same functionality yourself.

Frameworks save money. This is the corollary to frameworks saving time. Commercial frameworks cost a few hundred dollars. Building your own framework costs hundreds of hours, so unless your time is virtually worthless, even adding the time to learn the framework, you're likely to save money.

Frameworks are well-tested. Not only have the developers of the framework tested it, but framework code is in use in dozens, hundreds or even thousands of applications, so the worst bugs have been caught and fixed.

Frameworks are documented. At least any framework worth buying is documented. That makes it easier to pass projects on to new developers, and means that at least some of the code in your application is documented.

Frameworks evolve. Most of the commercial frameworks for VFP are still in active development. This means someone else will incorporate new VFP and Windows features and provide improvements.

Frameworks teach. For the most part, the frameworks have been written by VFP experts, so reading the code (most of them come with full source) is a good way to learn how the pros do it.

Frameworks work around VFP bugs. Just about all of the commercial frameworks for VFP are written by people who are beta testers for the product and who have an intimate knowledge of the language. As a result, they're likely to be more familiar with problems than the average developer and know better how to work around those problems.

Given the strength of those arguments, why would you build your own framework?

In my case, I'm working with an existing application that doesn't use a framework. I need to retrofit a framework into it. Doing that with one of the commercial frameworks would be extremely difficult.

In addition, the commercial frameworks are fairly generic. If you're building applications in a particular niche, it may be worth the time and effort to build your own framework. (In fact, there are some commercial frameworks available for building web applications with VFP.)

You may also have intellectual property issues. If management requires you to own all rights to the applications you produce, your own framework is your only choice (and your management is short-sighted, but that's a topic for another day).

Finally, you can learn a lot writing a framework. For me, doing this after more than a decade with VFP, it's a chance to pull together a lot of accumulated knowledge and to test some ideas about application design.

The bottom line is that you should be using some framework, whether you build or buy. Without a framework, you're producing an application that will be extremely hard to maintain and is likely to be inconsistent both internally and in its user interface. (If you're thinking "I don't use a framework, but my application isn't like that," then you probably are using a framework of your own design, but haven't thought to call it that.)

It's your call whether to invest your time building your own framework or learning one or more of the commercial frameworks. Whichever you choose, it'll be time well spent.

–Tamar