FoxTalk
PINNACLE
Solutions for Microsoft® FoxPro® and Visual FoxPro® Developers

The Application Framework Roundup

Visual MaxFrame Professional 4.0

Kelly Conway

WIN 5.0 6.0

In his previous article, Kelly Conway discussed factors and criteria for evaluating application frameworks (see "Evaluating Application Frameworks" in the October 1999 issue of *FoxTalk*). This month, he applies those criteria to the Visual MaxFrame Professional 4.0 framework from Metamor Worldwide.

THE Visual MaxFrame Professional (VMP) application framework has been around since the early days of Visual FoxPro 3.0. The most recent version, VMP 4.0, was released during the summer of 1999 and supports Visual FoxPro versions 5.0 and 6.0.

Features

Why do developers use frameworks? Some of us probably like (and need!) the guidance that they offer. However, I'd guess that the No. 1 reason why many of us choose to use a particular framework is the list of features that the framework provides. Detailed descriptions of all VMP features would require more space than this publication allows, even in an Extended Article. Here, then, is a list and discussion of the VMP features that I find most compelling. If a feature that you need or want isn't listed here, don't assume that VMP doesn't offer it. Please contact the vendor (see the sidebar "Visual MaxFrame Professional 4.0") for a complete list of all features.

Application setup and configuration

VMP supplies an Application Setup wizard that makes quick work of creating each of your applications. There's no requirement that you use this wizard, but you'd probably be silly not to. The wizard walks you through several steps where you make choices about the application you're creating and the features to be included. Once you complete your choices and click the "Finish" button, VMP creates everything needed by your application (short of your application-specific data, forms, reports, and logic, of course). The items created include your application's directory structure, database container, framework tables, framework subclasses, default menu, configuration tables, FoxPro CONFIG files, and project files. One feature new to this wizard in version 4.0 is support for creating, and subsequently subclassing from, an intermediate set of VMP framework subclasses (see **Figure 1**). Of course, since the wizard comes with complete source code, you can subclass and modify it to your heart's content.

Database services

VMP includes a class ("cusDBCSvc") that provides several database service operations. These services include opening and closing databases and tables, recreating indexes, updating and repairing table structures, and packing tables. Stonefield Database Toolkit (SDT) can be hooked to this class so that SDT's features may be used in place of the ones provided by VMP.

Visual MaxFrame Professional 4.0

Metamor Worldwide
14710 Newbrook Drive, Suite 100
Chantilly, VA 20151
703-631-6500
http://www.maxlink.com/vmpmain.htm
$399 per developer

User logins

VMP provides several services related to tracking application users. By default, a standard login dialog box appears when your application begins. After a user has logged in, information about that user is available within your application through the global oUser object. Other user-related VMP features include the ability to see which users are currently logged in, track user session history, and lock users out of the application for maintenance tasks (see Figure 2).

Security

The security model introduced in VMP 4.0 is nothing short of spectacular. Modeled after Windows NT, the VMP security model includes users, groups, secured items, and permissions. You (or the application administrator) create users and groups. You (the developer) create a table of secured items. VMP includes a utility that picks those items up from the VFP project and allows you to edit the created table. You (or the application administrator) assign groups access to secured items and then assign users to groups (see Figure 3). To make the model even more flexible, you can also assign groups to groups. Exceptions can then be made, at the user level, to either grant or deny a specific user access to a particular secured item. Secured items can include any component of the application—for example, menu choices, forms, reports, pages, grid columns, or individual user interface controls. Access rights vary by component type, but they include no access, read-only access, and full access. I can't imagine a security model more flexible than this one. Most developers probably will find the security setup process a little difficult to grasp at first, but isn't that the usual price of total flexibility? If this model is overkill for a particular application, you can opt to use the less complex version that remains from previous VMP versions.

Forms and the forms manager

VMP contains many form classes in the framework hierarchy, but, when you get right down to it (as is clearly pointed out in the reference guide), there really are only four form classes from which you'd eventually subclass most of your forms. There's a base class for data entry forms and a base class for dialog boxes. Then, there are two data entry form subclasses that create specific user interfaces. The latter two classes are typically used for quick prototypes, but they're also useful as parent classes for your application forms, if you like. As I'll discuss later, the bulk of all code used by forms is included in the abstract data entry form class ("frmDataEntry"). The extensive and robust data-access code contained within the base data entry form class "automatically" handles updates to all buffered tables within the data environment, whether you use the VFP data environment in your forms or manually open tables in form classes.

VMP also includes a global forms manager that knows how to manage a collection of frmDataEntry subclass instances. Through the forms manager, you can, for example, maintain a list of open forms in the "Windows" menu, send messages to open forms telling them to requery a data source that was updated in another open form, and request that all forms close when the user has asked to shut down your application.

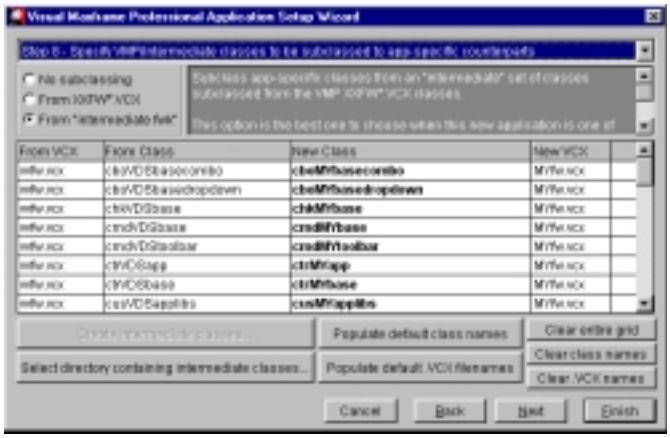

Figure 1. Step 8 of the Application Setup wizard allows both creating and subclassing from a set of intermediate framework classes.

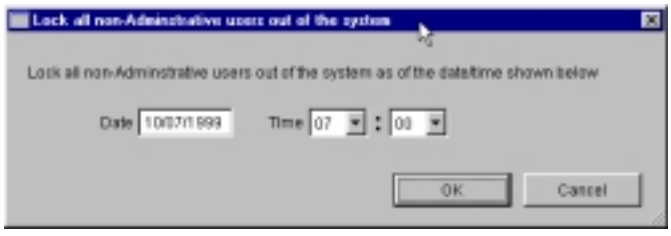

Figure 2. Administrators can lock users out of the application at a specified time.

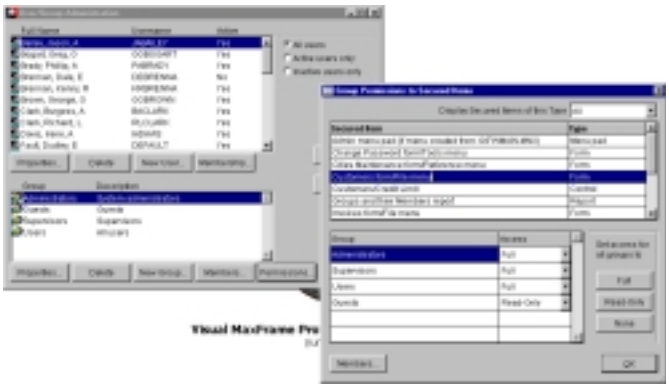

Figure 3. Administrators can assign access to secured items by user and/or group through the included security manager interface.

Data entry controls

VMP includes a host of data entry controls that boast numerous features. Code in each of these controls assumes that the control will be contained within a frmDataEntry subclass. Every bound control knows how to put the form into edit mode when a change is made. There are Quicken-like "quick fill" text boxes that invoke a picklist form when an invalid value has been entered. There are also composite classes that you can use to combine a "quick fill" text box with any number of other controls that can display values for table fields related to the choice you've made (for instance, selecting a ZIP code and displaying the associated city and state). Most data entry controls allow access to a shortcut menu that's full of features for which your users will thank you. Another common feature of VMP data entry controls is the ability to require an entry in a control by setting one property (which also causes the control to display in a different color to indicate that it requires a value).

Toolbars and the toolbars manager

Several toolbar classes are included that automatically work in conjunction with VMP data entry forms. The toolbar buttons invoke the appropriate form methods without requiring the developer to write any code. Each button also enables and disables appropriately, based on the state of the active form. The toolbar manager takes care of counting the number of outstanding references to each toolbar and destroying the toolbar when the last reference is closed.

Data entry grids

Because VFP grids are such complex (and often unstable) beasts, many developers shy away from using grids for data entry. Metamor has gone to great lengths to create robust data entry grid classes that allow you to provide users with fully functional data entry grids. The commented source code within these grid classes documents dozens of VFP bugs that are deftly avoided. The various forms within the example application include several data entry grids, many of which provide good examples of using controls other than the typical text boxes inside grid columns.

Other forms

The framework contains many other form classes in addition to the data entry form classes I've already mentioned. These classes include forms that you can use for picklists, movers, specification criteria (filtering), login, password changes, report selection (catalog), report destination selection, and several other types of dialog boxes.

Running forms standalone

One of the unique features of VMP is that you can run forms in "standalone" mode from the command window or project manager. When run this way, forms "automatically" instantiate classes needed for providing application services. This improves developer productivity by eliminating the need to start up an application to test and debug a single form.

Reports and the report catalog

Reporting is an area where VMP seems to be more thoroughly thought-out (or at least documented) than the other frameworks I've evaluated. The report base class contains all of the code that most VFP developers typically place in a program that calls the report form. For simple listings, you can merely create a quick report and then add an entry to the report catalog table that describes the report and lists the report file (FRX) name. For more complicated circumstances, you can subclass the report class and add your own specific SQL SELECT and/or other VFP code needed to process the report. The optional report catalog form is a nice selection mechanism for the user (see Figure 4).

Message services

For flexible messaging, VMP uses the near-ubiquitous, public domain MSGSVC utility from Steven Black (or the more extensive version that ships with Steven's INTL Toolkit). Using this facility for your own messages is optional, but it's a good idea to do so.

Referential integrity "engine"

VMP implements an optional table-driven referential integrity engine. You simply define your application's table relationships in the provided table and insert calls to the VMP_RI stored procedure in your table definitions.

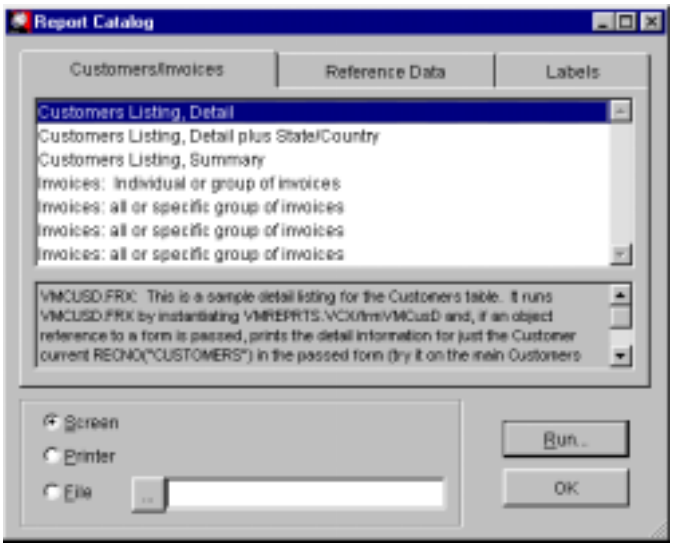

Figure 4. The included report catalog is one interface you can provide users for running reports.

This is another optional feature that merits usage.

Client/server features

Other than the new security model, improved client/server support probably is the biggest enhancement in VMP 4.0. The base data entry form class's data access methods handle any combination of local and remote cursor updates, wrapped in a single transaction. The "cusConnectionSvc" class provides services that wrap the complexity of dealing with an ODBC connection. Library routines wrap the VFP SQLEXEC and REQUERY functions with code that adds robustness and error trapping. The framework documentation and examples detail the steps that you can take to build an application that accesses remote data using either remote views, SQL pass-through, or a combination of both.

Push/pop settings classes

VMP contains several classes that you can use to "push" and "pop" various VFP settings so that you can safely and reliably set and restore the current alias, record number, index order, and other environmental settings in your code.

Error handling and logging

VMP provides a global error handler that alerts the user to error conditions and logs every detail possible when an error does occur. While the details logged are thorough and the user interface for viewing and reporting encountered errors is full-featured, I get the sense that VMP applications have less ability to recover from errors than do applications built with other frameworks. To be specific, practically any kind of error in a VMP application will force the user to close the application (after the error has been logged). Most other frameworks that I've evaluated use an error handler more like the one that Doug Hennig made popular with Stonefield's tools. That error handler, when possible, gives the user the opportunity to cancel their requested operation, leaving the application safely running.

Library routines

Also included with VMP are several general-purpose library routines contained in program (PRG) files. Nearly 100 programs include useful routines for encrypting and decrypting strings, data field uniqueness validation, unique ID generation, object reference comparison, a MESSAGEBOX wrapper, and several wrappers to various Windows API functions. OOP purists might argue that these routines could or should be methods within a "utility" object. However, many of these routines were written for FoxPro 2.x and remain useful.

Application configuration table

Two separate tables allow changing several application configuration options without recompiling your application. Global settings that may be configured in these tables include, among many others, the expression used to determine whether the logged in user is the developer, the color to use for required controls, and the class to be used for each global application component.

Example application

The VMP example application demonstrates practically every feature of the framework. As such, it provides a great tool for learning as well as a great place to "steal" working source code. The application contains several dozen menu options leading to demonstration forms, dialog boxes, and reports. All example forms contain a button that displays the ZReadMe documentation from the class on which the form is based (see **Figure 5**).

Learning curve

Evaluating a framework's learning curve is difficult, especially when it's one that you've used for several months or years. Personally, I found that learning to build VMP applications wasn't difficult at all, but I was exposed to the product early and have since only had to learn about the new features that have come with each release. Putting myself in the shoes of a developer who's just picking up VMP for the first time today (three versions later than I did), I suppose that I'd find the task somewhat more daunting.

VMP does have quite a list of features and also provides abundant flexibility. Learning to access all of those goodies doesn't happen in a day. However, I still feel that the detailed tutorials and the extensive example application provide more than adequate resources for the developer who needs to get started quickly. All of the framework's nuances can only be learned and mastered over time and with experience. All developers obviously

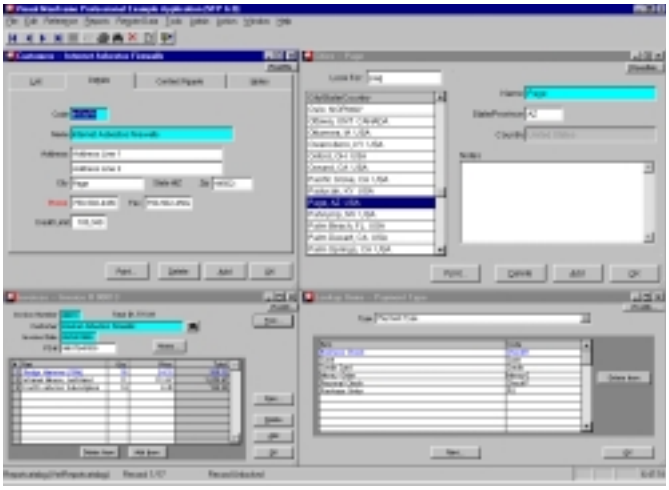

Figure 5. The example application demonstrates most of the framework's features.

would be well-served in this endeavor by working with someone who's ahead of them on both the VMP and VFP learning curves. If you aren't fortunate enough to have people like that in your company, frequent the various online areas for the next best thing.

Documentation

Like most application frameworks, VMP is documented at several levels. VMP's documentation includes a reference guide (which includes several separate tutorials), a guide for upgrading VMP 3.0 applications, commented code, well-described classes, properties, and methods, and an extensive example application.

The reference guide ships as a series of Word documents, each of which contains one of the manual's 51 chapters, table of contents, or index. At the time of this writing, nearly a dozen of the chapters are empty, awaiting attention from Metamor. Still, this is a large manual that will take you several days to read. Luckily for those who want to get started quickly and learn as they go, the last dozen chapters are in a tutorial format. The tutorial chapters detail how to perform various tasks, from starting a new application to creating one-to-many data entry forms to creating report selection criteria dialog boxes. I think the reference guide does a good job of getting new VMP developers up to speed, assuming that they take the time to read it. However, information in the reference guide could be more accessible than it is. Even with the table of contents and index, finding specific information within 50+ separate documents can take a while. You can print the reference guide, of course. Just be sure to have a fast printer, lots of toner, and about a ream of paper. If you have Adobe Acrobat, perhaps a better idea would be to take the time to combine the separate chapters into one document and then create a PDF document that you can view and search.

Developers who wish to move their VMP 3.x applications forward will be both horrified and thrilled by the 74-page upgrade guide. The horrifying part is that it seems that quite a bit of work is required to upgrade your existing applications (much more than was the case when upgrading from VMP 2.0 to 3.0, for example). The thrilling part is that the guide walks you through the upgrade process, step by step, in a way that lets you know that the author has done it before, probably more than once, himself.

When you get beyond reading about VMP applications and into creating them, there will probably be times when you'll want to examine the framework's source code. This can be useful for at least a couple of reasons. Sometimes, you just feel the need to know how something is implemented. While I can't say that I subscribe to this theory, I've heard more than one developer ask, "How can I ship an application to my customers without knowing what every line of code is doing?" There also will be times when you need to augment the framework's behavior, and you'll need to examine the code to see whether a DODEFAULT or perhaps a NODEFAULT is appropriate. Additionally, if you're smart, you'll even take some time to look through the framework's code in a quest to learn more about developing VFP applications.

Regardless of your reasons for perusing the code, you'll appreciate the level of comments inside it. In most cases, it's as if you have Drew Speedie, the framework's architect, sitting next to you at your desk, explaining what's going on. In addition to the "how this works" type of comments, VMP also contains tons of comments that explain various workarounds for VFP bugs. Every VMP class also contains a ZReadMe method that contains general information about the class. These methods often include step-by-step instructions for implementing an instance of the class.

Don't forget about class documentation—those descriptions that developers are supposed to fill out for each class, property, and method. In the case of VMP, every class—and each class's properties and methods—are described in detail. These descriptions, of course, are available in the VFP property window and class browser, as well as in documentation that you might generate from the classes. The files that ship with VMP don't include a design model, although Metamor was passing out poster-sized class hierarchy diagrams at the last conference I attended. While a Visual Modeler or Rose model would be nicer for those of us who use those tools to create and document our designs, the poster, at least, serves as both a quick overview and good reference for the framework's class hierarchy.

Development methodology

The Metamor staff probably would deny that VMP imposes any sort of methodology on developers, and I'd tend to agree. More than most other frameworks I've evaluated, VMP allows you to make complete use of FoxPro at all times. In other words, the framework doesn't add its own developer interface on top of VFP and, therefore, doesn't really lead or require you to develop your applications in any specific way.

However, while the framework's architecture doesn't necessarily dictate it, VMP developers are encouraged by the class hierarchy and examples to place the bulk of their application code in form methods. For this reason, I'd say that VMP is probably best suited for a methodology that focuses on RAD or prototyping. The benefit of its form-centric architecture (I'll talk about the disadvantages of this architecture later) is that forms for single-tier and client/server applications may be created, tested, and modified very quickly.

Development cycle

This is one area where I find that VMP has a decided advantage over any of the frameworks that I've evaluated. While most frameworks require that you run your application's main program (including the load time and login) to test anything, the Metamor developers have gone to great lengths to ensure that nearly all application features can be tested in a standalone mode. For example, while working on a MaxFrame-based form, you can use the VFP toolbar buttons that allow you to jump back and forth between running and designing. The "base" form class contains code that recognizes that you're running in a standalone mode and then instantiates any application-level classes that you might need to support testing your form.

I find that this feature of the framework definitely shortens my development cycle and causes less frustration during those obligatory find-a-bug, fix-a-bug, find-a-bug . . . sessions (I can't be the only one <s>). However, if you're the type of developer who codes all day and then wants to test the full application at once, nothing in VMP prevents that strategy either.

Architecture

To describe the intended architecture of a VMP application, allow me to quote from the framework's documentation: "VMP is specifically designed to handle one-tier and two-tier database applications. However, there are an increasing number of application development situations that require an n-tier solution."

The documentation goes on to explain that header comments of relevant methods within the framework contain brief instructions for substituting middle-tier objects to perform the appropriate actions. These middle-tier objects, of course, can be global, attached to VFP forms or COM objects. However, the task remains for the developer to create the classes from which these objects would be instantiated and to integrate them with the framework. It's my opinion that overestimating this task would be impossible (in other words, it ain't gonna be easy). This section of the reference guide concludes by stating: "Future versions of VMP will add features to be even more 'n-tier friendly.'"

Obviously, if your goal is to begin creating distributed applications immediately, you should give due consideration to some of the competing frameworks. On the other hand, if you're among those developers who are just learning about VFP and OOP and for whom "Windows DNA" might sound like something that O.J. Simpson might (or might not <s>) have left on the glassy surfaces of his Ford Bronco, VMP's architecture might be just what the doctor prescribed.

Without regard for n-tier capabilities, the class hierarchies that make up the VMP framework are well organized and documented. One class library contains each of the obligatory one-off base classes. Other libraries contain classes that are related to each other in function (forms in one library, toolbars in another, database utilities in yet another, and so forth). Classes, and their properties and methods, are well named and described, so as to reveal their purpose.

A running VMP application contains several global object references to component classes that make your application tick and whose names provide insight into their responsibilities—oApp, oUser, oForms, oToolbars, and oSecurity, to name a few. Messaging between these components is fairly well-defined, and each of these objects provides the developer with services that make implementing many otherwise complicated features a piece of cake.

Flexibility

One of the most important framework evaluation criteria, in my book, is flexibility. Most developers need the guidance and pre-built functionality that a framework provides, but they also need a framework that doesn't get in the way when circumstances require a specific implementation or augmentation. VMP, for the most part, excels in this regard. The major methods in most classes (and even some of the utility programs) contain abundant "hooks"—calls to external methods in which you place your own specific code. These "hook" methods, typically going by names that start with the words After, Before, or Shell, provide tremendous flexibility. Because of this, you should *never* have to modify a framework class or program directly. The thing that detracts from VMP's flexibility is the length of some methods (see my discussion of this topic in the "Source code" section).

Another way that a framework can provide flexibility is by allowing you to replace or add application functionality by integrating your own (or other third-party) components. Again, VMP does a good job here. Its application configuration table provides an "Abstract Factory" scheme for substituting classes that will be instantiated at runtime—a form of "late-binding," if you will. In addition, the framework contains classes and documentation that specifically help developers plug in other third-party components such as Stonefield Database Toolkit, Stonefield Query, Classy Components' QBF, Micromega's FoxFire!, Steven Black's INTL, and Take Note's FoxAudit.

Longevity

As I mentioned earlier, VMP was first introduced in the days of Visual FoxPro 3.0. At that time, most VFP developers were brand-new to the ideas of OOP. There was much "analysis paralysis" and gnashing of teeth as most of us struggled to figure out "the best" way to build applications with our new toys, er, I mean, tools. In my estimation, VMP was the first application framework that

was available (and stable) for Visual FoxPro. I was quickly building full-featured, object-oriented applications with it seemingly months before competing products were available to present a choice.

Now that two or three of the competing frameworks have been designed (or redesigned, in some cases) to accommodate n-tier applications and the vendors' increased knowledge of OOP, VMP might be showing its age. Metamor now is in the position, in my opinion, of needing to choose between backward compatibility with previous VMP applications and a substantial overhaul that would allow them to be more DNA-friendly. I think it will be interesting to see which course they choose for their next version and whether a suitable compromise (supporting newer technology without requiring massive retrofits) can be made.

Vendor/principals

This category is sort of a mixed bag for VMP. The principal framework architect, Drew Speedie, should be a stranger to no VFP developer (let's put it this way—if you haven't heard of Drew, don't bother sending me your resume for a VFP position <s>). The original company for which Drew created VMP, MaxTech, also was fairly well-known in the VFP community. However, since those beginnings, MaxTech was acquired by GE Capital Consulting (yes, *that* GE) and then merged with or was swallowed whole by Metamor Worldwide.

While I know little or nothing about Metamor (or GECC before it), I haven't noticed any significant impacts on service or support from either of these business changes. Metamor and Drew, along with several other VFP developers from the company, were present at all of the 1999 VFP developer conferences.

Performance

As I mentioned in my previous article, nearly all application frameworks must contain a certain amount—perhaps lots—of "generic" code to handle various situations. This additional code and flexibility almost *has* to result in applications that run slower than ones for which you handcraft each line. So, for the purposes of our evaluations, we seek only to determine whether each particular framework performs about the same as the others, much faster than the others, or much slower than the others.

The typical places where a framework can incur significant overhead include application load, form load, and record saving/navigation. In each of these areas, my perception of VMP's performance is that it's about average, compared to the other frameworks I've evaluated. So, if you've never used a framework before, you'll likely find that VMP applications seem a little more sluggish in some operations than do your custom apps. However, anyone for whom absolute speed is a major criterion probably stopped reading this article when they encountered the word "framework." For most of us, the performance of applications built with VMP shouldn't be an issue.

Source code

As mentioned previously, VMP comes with full source code, including the code and classes for all developer tools. Also, as mentioned previously, the code is well-commented and, therefore, is usually fairly easy to understand. However, there are times when an application error occurs within VMP code (in my case, usually because I've forgotten to set a particular property or made a similar mistake). In those situations, it's often difficult to determine the actual cause of the error and find your way back out. This can happen when running any unfamiliar code, of course, so VMP probably presents neither a lesser nor a greater obstacle than other frameworks in this situation.

One of the things that I feel could be improved in VMP's source code is that many of the methods are quite long and often contain embedded messaging that's unconditional. These long methods are difficult to subclass when what you want is to execute some framework behavior, add your own behavior, and then possibly execute the additional framework code. In the case of the messages, VMP does do a great job of making all messages flexible by using Steven Black's public domain MSGSVC scheme. There's no provision, however, for suppressing messages when, for example, a method is called as part of a COM component running on a server. Situations like this lead you to copy the framework's code into your subclasses (<yeech>) and then remove the offending pieces. Admittedly, these cases, for many developers, might be few and far between.

I can never evaluate source code without thinking about coding standards. The coding standards used for VMP are described in the documentation and followed religiously. The standards themselves are pretty "standard" compared to what I've seen in the FoxPro community. Hungarian notation is used to indicate the scope and type of variables. Hungarian notation also is used in class and property names. The one departure from familiar standards, for me, is the use of "i" (standing for "instance") as the scope prefix for properties. VMP doesn't force the use of any coding standards. If, however, you don't already have your own standards, you could do far worse than to follow the ones recommended in the VMP documentation.

Data access

MaxFrame currently supports local VFP data (with or without local views) as well as client/server data via remote views (ODBC) and/or SQL pass-through (SPT). There's no built-in support for ADO recordsets. I've heard

people say that MaxFrame "requires" developers to use views for local data. While this is most definitely untrue, the framework does do a nice job of encouraging the proper use of views. The documentation and examples clearly explain how to switch your local views to remote views when you need to move your application's data to a client/server database.

The location of data access code in VMP is one of the framework's disadvantages. Remember that VMP has been around since the early days of Visual FoxPro 3.0. While the framework has benefited from extensive enhancements over the years, the basic "form-centric" architecture hasn't changed much. All data access code is held hostage within methods of VMP's "base" data entry form class. The code there works as advertised and is quite robust, accounting for and handling many situations and potential error conditions. I suppose that there's some percentage of developers for whom this architecture is even preferred. For example, those who feel they have no need to build distributed applications or who possibly are just moving up from FoxPro 2.x might be overwhelmed by a more flexible design, but if your goal is to build distributed applications and to create a data layer that can be called upon from multiple interfaces, this code is simply in the wrong place.

As Drew Speedie is always quick to mention, the architecture of VMP in no way precludes a developer from creating his or her own "business objects" and then using them within VMP forms and other interfaces. While this certainly is true, I'm looking forward to a future version of VMP where n-tier applications might be directly supported rather than requiring developers to create their own abstract classes for business objects and the data access layer.

Included developer tools

VMP provides a wealth of tools for the developer, and each of them comes with complete source code. I've listed those that are the most useful to me. VMP includes several other useful developer tools, but every VFP developer should have, at a minimum, the following tools (or equivalents) in their bag:

- *A*—Performs a very thorough reset of the development environment. Includes hooks for executing your own code.

- *XXDTHACK*—Allows "controlled hacking" of forms and class libraries (see **Figure 6**).

- *XXDTSRCH*—Allows global searches of your project or specific components. Improvements in VMP 4.0 allow you to directly edit components in which the search phrase was located.

- *XXDTPOPC*—Generates code to recreate a database and its contents. The generated code includes INSERT INTO statements appropriate for rebuilding test data or distributing starter data with an application.

Support

Support for VMP is offered in several ways. Free support is available in the VMP Discussion area on the MaxLink Web site (http://www.maxlink.com) and in the FoxUser forum on CompuServe. While not specifically mentioned under the heading of "Support" in the product's documentation, I've seen support requested and provided in the Universal Thread (http://www.universalthread.com) and in Microsoft's VFP news groups. All support provided via these forums is free. There is, of course, no guarantee that your online question will get answered correctly, quickly, or even at all. However, if you've visited any of these FoxPro "communities" in the past, then you're probably aware of the wonderful support network the citizens there typically provide.

For those times when you need the right answer and you need it now, you can call Metamor directly on their toll-free support line. All support incidents are charged (to your credit card) at the rate of $150 per hour, with a $75 (half hour) minimum per incident. You're charged for all time spent researching the problem and communicating its details. Metamor plans a change in this area so that you'd submit even paid support requests to their Web site. So, by the time you read this, you might be directed to post your request online if you call the support number. While not guaranteed, Metamor strives to turn around all support items by the close of the following business day. If they determine that your question/problem is due to a bug in VMP, there's no charge for the support incident.

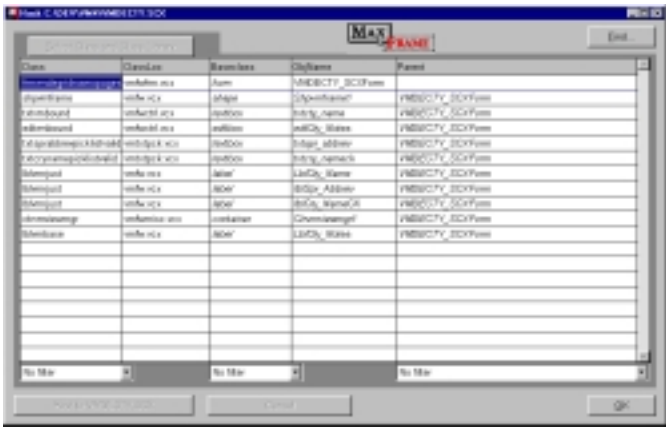

Figure 6. Developers can use the "hacking" tool to change the parent class of contained objects.

Training

Metamor offers MaxFrame training several times per year at various locations. While I haven't attended a Metamor class, I've seen the training materials that are used. The materials appear to do a good job of leading students through the development of a typical application. The courses are taught either by Drew Speedie or by one of several other Metamor developers who work with MaxFrame consistently. I suggest that you visit the vendor's Web site for details.

And, okay, I'll go ahead and say it. I also teach a Visual MaxFrame Professional class at the Vision Data Solutions training facility in Independence, MO. Feel free to contact me directly for the details of that class.

Cost

Visual MaxFrame Professional 4.0 costs $399 per developer. There are no runtime royalties. However, if you need to distribute the source code, it would, perhaps obviously, be at the cost per copy. As is typical, version upgrades for current customers are available at a reduced price.

Metamor also maintains a Web site where developers can get the latest bug fixes. Those fixes, for some reason, are posted as HTML text that each developer must copy and paste into his or her own version. While I'd prefer the ability to merely download the updated class libraries, this beats having to wait for the next release to fix a bug that's causing your users grief. Fairly frequent updates are available for download, and these updates incorporate any bug fixes and other enhancements since the last release.

Summary

When Visual MaxFrame Professional was first released, the advertised target audience was experienced FoxPro 2.x developers looking for a way to quickly create Visual FoxPro applications and climb the VFP learning curve. The current version of VMP still addresses that market very well. VMP remains one of the few frameworks that I recommend to many developers without hesitation. Even if you never were to build an application with the framework, I'm certain that most of us will find at least $400 worth of learning potential in the mature code, comments, and documentation.

If you're already using VMP, there's no doubt that you should upgrade to version 4. If you don't see a distributed application in your immediate future or if you're willing to provide your own data access and business layer classes, you might do well to stay with VMP and wait for the possibility of direct n-tier support in the next version. However, the growing number of developers who are interested in, or already working on, distributed applications probably would be better served to evaluate the newer frameworks that have been built with n-tier designs in mind.

(*Note:* Dave Aring, a Vision Data colleague, provided valuable input on this article.) ▲

A senior developer, project manager, and trainer with Vision Data Solutions in Independence, MO, Kelly Conway has worked with every version of FoxPro since 2.0. He teaches several classes on Visual FoxPro development, application frameworks, and software development processes. He has also had several articles published and is a regular contributor at the Midwest Fox Pros User Group. kconway@visionds.com.