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

This is an exclusive supplement for *FoxTalk* subscribers. For more information about *FoxTalk*, call us at **1-800-788-1900** or visit our Web site at **www.pinpub.com/foxtalk.**

Extended Article

Visual Basic for Dataheads: A New Beginning

Whil Hentzen

LET'S face it. You're going to have to learn Visual Basic one of these days. It's not because "they're going to kill Visual FoxPro"—far from it.

It's not because Visual Basic is "better" than Visual FoxPro. It's a general-purpose programming language with its own strengths and weaknesses. Fox is a data-centric programming language, again, with its strengths and weaknesses. Each does some things well and other things, er, poorly. "Better" is truly in the eyes of the developer.

And it's not because I've gone bonkers. That happened long ago.

There are three reasons. First, you might think I'm lying about the future of VFP and figure it's safe to cover your bases. Well, I'm not lying—just like any dutiful Microsoft shill, I'm simply repeating what I'm told to. <g> But not putting all of your eggs in the same basket is probably a good idea, regardless. Ever *really* wonder why we're still saddled with a Report Writer and Menu Builder from 1992, and there are no plans for updates? Maybe it *is* time to look elsewhere for those components!

Second, "everybody" has it. And you want to be popular, don't you? <g> How many of you preferred another word processor or spreadsheet but gave up because every document you received was in Word 6.0 format, and you got tired of converting—and having pieces lost in the process? It was just easier to go with the tide—and, besides, you probably got a copy of it free from somewhere or another.

As a result of everyone having VB, when you see sample source code, it's often in VB (or VBA). Wouldn't it be nice if you had a better familiarity with the language so you understood what "DIMs" and "VARIANTs" were, instead of having to guess?

The last reason is that, as alluded to earlier, there are some things that Visual FoxPro just doesn't do very well. For example, I've been wrestling with the integration of a high-end imaging ActiveX control in VFP. It's been an ugly road, and, as I'm writing this, it looks like the road is a dead-end. The customer isn't really very interested in hearing about "the differences between VFP's and VB's containership and hosting abilities" (there's the one guy in their IS department who's muttering, "I knew we should have written this in VB"). They just want it to work.

Why spend hours and days working around a weakness when another tool can be used that doesn't have that weakness? How about, as Dan Freeman suggested, "building an OCX in VB? Just wrap it around the recalcitrant control. The control will think it's hosted in VB, and Fox seems to work better with VB controls."

And, if you've been following John Petersen's column, you've seen the differences between VFP and VB in talking to ADO—VFP just doesn't cut it in some places. Why not plug a VB module in to take up the slack?

Well, if you go that route—and I think it's one worth considering—you'll need to get up to speed with VB. While that's not a significant challenge (hey, there are only something like 19 keywords in the entire language <g>), it could still be a longer process than you have time for. You could pick up one of those "Learn VB in 21 Days" books—you know, one that sits next to the 2,000-page "Write Your Own NT Kernel in 24 Hours" book (I swear—I *saw* one of these books!). But those are written to be as generic as possible—aimed at everyone from the experienced VB5 developer to the novice who still hasn't gotten the hang of the mouse.

What I'm going to do in this column over the next year is tackle VB from a Fox developer's point of view, pointing out similarities and differences—what to look at,

what to ignore. And, most importantly, I'm going to provide a very fast path to getting up to speed, because I know what your background is—so, unlike those general-purpose books, I don't have to write for a wide range of developers in the audience.

This month, I'm going to introduce you to the VB IDE—it's a little different from the sparse Command Window that we're used to. Actually, it's a lot different.

Lesson 1: The Visual Basic IDE

I'm assuming you've installed Visual Basic 6.0 that comes with the Visual Studio 6.0 package, and for the time being, it doesn't matter whether you have the Professional or Enterprise version. You can also get VB in some other versions—Training, Standard, and Mega-Galactica, but I'll ignore those. When you load VB, it will present you with a dialog box that asks you what type of project you want to create. I checked the "Don't show this dialog box ever, ever, ever, ever again" check box because when you create a new project from the File, New Project menu command, you'll get access to the same dialog box.

MDI vs. SDI

Once VB loads, you'll see an MDI window filled with windows, much like Visual InterDev (see **Figure 1**). This might startle some of you who have seen previous versions of VB—those used an SDI interface, which was rather disconcerting to us Fox developers; the windows just sat on the desktop, and you could see the desktop showing through the holes. (You can turn SDI on if you like through the Tools, Options, Advanced tab.) See **Figure 2**.

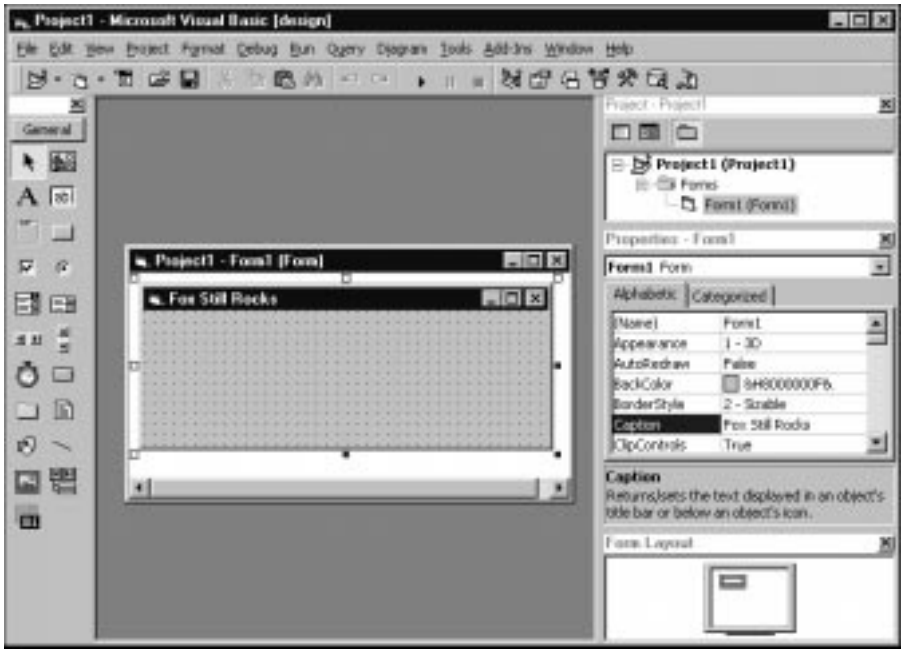

Figure 1. VB 6.0 sports an MDI interface by default.

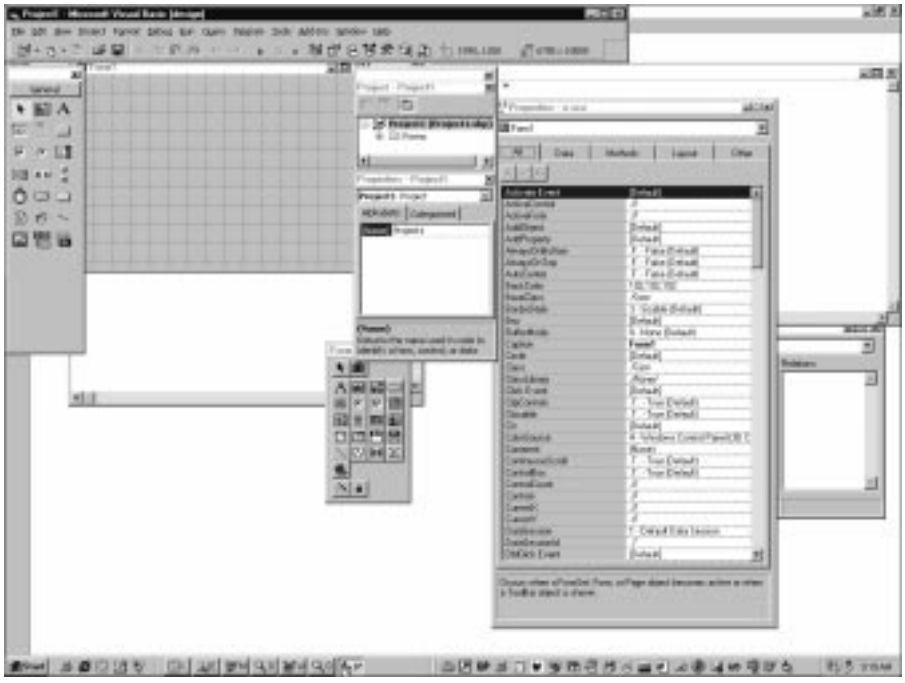

Figure 2. Use the Tools, Options, Advanced tab to change VB 6.0's interface to SDI—if you can stomach the mess!

What are all these windows?

What are all of these windows? On the left side is a long, narrow window with about a dozen controls in it—that's your Controls toolbar, just like Fox. If you let your mouse icon hover over a control, you'll get a ToolTip describing the control, just like every other Microsoft product on the planet. (I was going to say "Windows product" but figured it's pretty close to the same thing.) The bar across the top that's captioned "General" is called a tab, even though you and I think it looks like a "bar." Hmmph, those VB guys . . . However, the toolbar can contain lots of controls, and I mean *lots*—the native controls that come with VBE (Visual Basic Enterprise) number close to 125.

However, instead of segregating controls into different toolbars like Fox does (Standard, ActiveX, and your own libraries), you can separate the Controls toolbar by adding tabs to the toolbar, each of which will contain a certain type (see **Figure 3**).

VB uses a project metaphor to build applications just like Fox, and it's heavily dependent on forms, again, just like Fox. When you create a new project (either through the New Project startup dialog box or the File, New Project menu command), you'll be faced with a plethora of options. Pick Standard EXE for the time being, and you'll get a project named Project1 in the upper right-hand window that originally didn't have a caption in the title bar. This, then, is the Project Explorer.

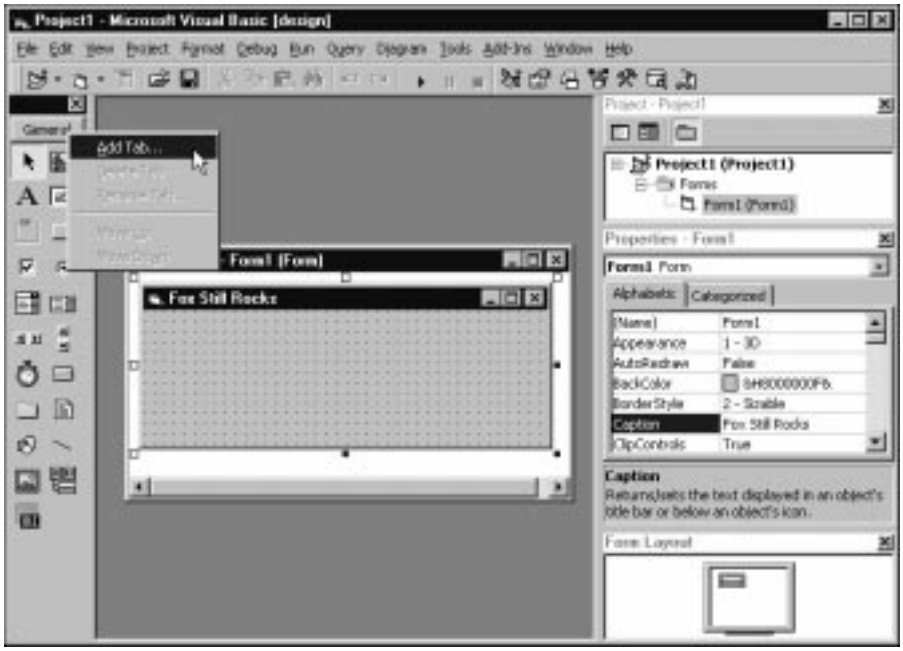

Figure 3. Right-click on the General tab in the Controls toolbar to add, edit, delete, and manipulate tabs.

Underneath the Project Explorer is the Properties window, and it provides the same function as our own Properties window, although the functionality is a bit different. You'll already see that you can sort the properties alphabetically or by category—Appearance, Behavior, Font, Misc, and so on. I've kept the properties sorted alphabetically because it's hard enough keeping track of which properties belong to VFP and which to VB—and what the minute differences are. Trying to remember whether "Visible" is an Appearance property or a Behavior property is just too hard. You'll see I've already changed the caption of Form1 to something more aesthetically pleasing—and I didn't even have to read the manual to find out how!

New to you is the Form Layout window in the lower right—it shows you a bird's-eye view of what the form is going to look like on the desktop. As you resize a form, you'll see the corresponding view in the Form Layout window change as well.

Your first VB form

The form that's created when you create a project is named "Form1" by default. By itself, it isn't very interesting, but you can run it just as you can run an empty VFP form. In order to do so, you can click on the Start button in the Standard VB toolbar (it's the button with the arrowhead pointing to the right, under the "m" in the Diagram menu if you've got both the menu and Standard toolbar docked and nudged up to the left). Alternatively, you can issue the Run, Start menu command or simply press F5 (see **Figure 4**).

You know what's cool? If you task-switch with Alt-Tab, you'll see that the VB form isn't running inside like VFP—it's an SDI form automatically.

Okay, "How do I stop this thing?" You can click on the close box of the form, or press the ever-so-intuitive Alt-F4, or just click on the toolbar button with the square box (two buttons to the right of the Start toolbar button).

Adding code to a form

I know that example is less involved than the typical "Hello World" program you first write in C, but, hey, this is *BASIC*. Let's put some code in this form.

Code windows are one significant departure in behavior from "the Fox way." First, in order to get to them, you can't use the Properties window. It's for *properties*, after all. It doesn't say "Properties and Code Window," does it?

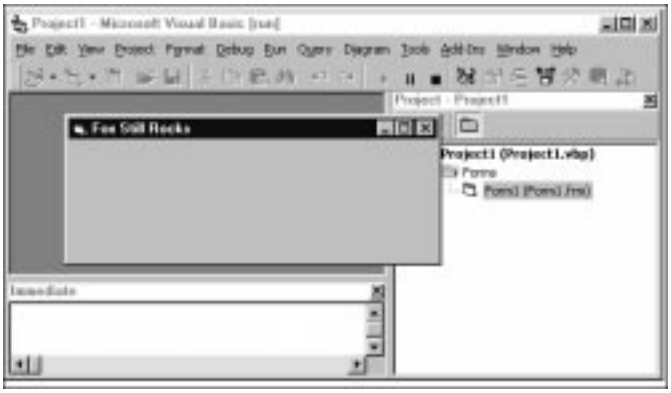

Figure 4. The Fox Still Rocks form in, er, action.

To open up the code window, you again have nine thousand ways of doing so. You can highlight an object in the Project Explorer and click on the View Code button in the project, as shown in **Figure 5**.

You can also issue the View, Code menu command, but the easiest way is to just double-click on the form. After any of these operations, the code window displays as shown in **Figure 6**. The left dropdown shows the various objects of the form, including "General" (I'll get to that later), "Form," and every control on the form. The right dropdown contains all of the events into which you can stuff code—and you're familiar with many of these, such as Activate, Load, Resize, and Unload.

In Figure 6, I got ahead of myself and started writing some code. Just when you got used to typing "MessageBox" in VFP, VB comes along and uses "MsgBox" to perform the same function. You'll notice in Figure 6 that VB has "Code Completion"—a ToolTip will display prompting you regarding the parameters that can be used with the function you're typing.

You'll also see that code for a particular procedure is bracketed by "Private Sub <name>" and "End Sub" statements, just like we use FUNC and ENDFUNC in Fox. "Sub" is a holdover from the olden days when programmers used to write "subroutines," and VB puts them in for you automatically. If you delete one of those lines, "unexpected results may occur."

I suppose I could have put a message box in the form's Click method, but that would have been going from the excruciatingly trivial to the nearly excruciatingly trivial. Instead, I added a command button from the Controls toolbar, double-clicked on it, and entered the MsgBox in there. Obviously, running the form will generate a form with a command button on it, and clicking on it will display a message box.

The VB code window

But let's talk about the code window for a minute, because there are some new things showing up. In **Figure 7**, I've shown the code window with pieces from several methods. As you can imagine, this is pretty handy—being able to see code for more than one procedure in the same window (as long as those procedures are small).

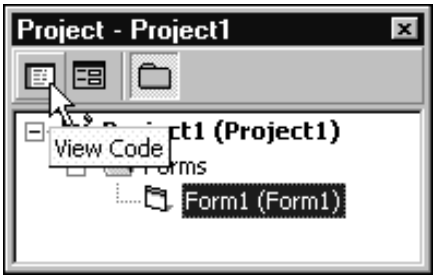

Figure 5. The left button in the Project Explorer toolbar opens a code window.

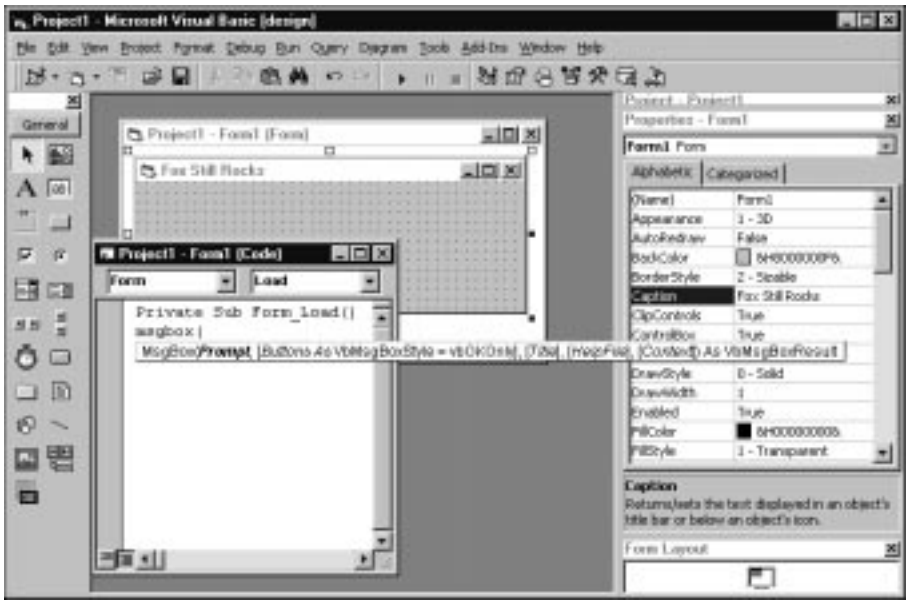

Figure 6. The VB code window with a command in Code Completion mode.

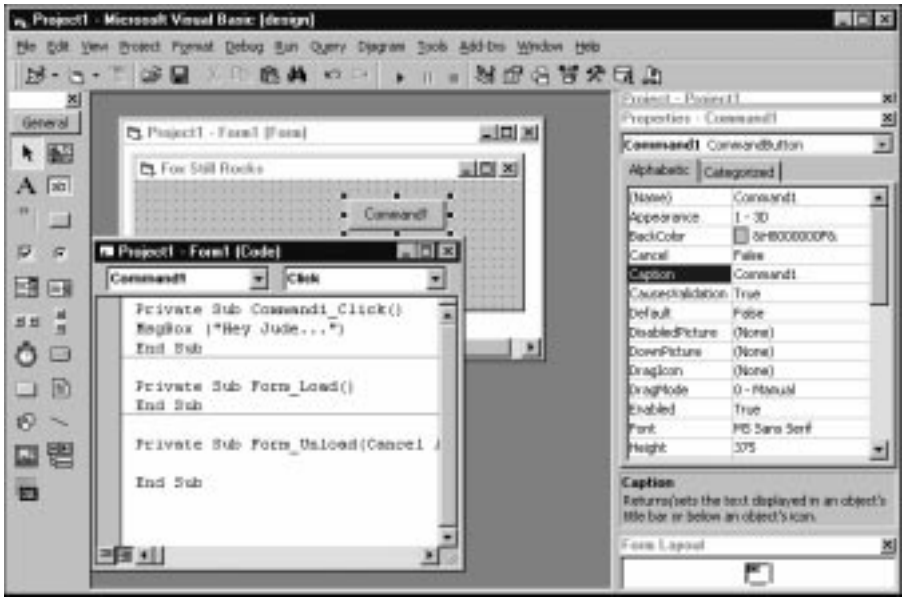

Figure 7. Press the Method View button next to the horizontal scroll bar in a code window to see all of the procedures in the same window.

Notice two buttons to the left of the horizontal scroll bar in the bottom of the code window. The right button is selected in Figure 7 and indicates that you want to see more than one procedure at a time. This button is called the "Method View" button. The left button, Procedure View, limits the display to one procedure at a time.

Saving and printing projects

A project is a file on disk, just like in Fox, but data in it isn't automatically saved like that in a PJX, so you have to do it yourself. However, when you save a project, it will prompt you to save all of the unsaved components, and if they're unnamed, you'll be prompted to name them along the way.

You can also print the contents of a project simply by selecting the File, Print menu. The Print dialog box contains a number of options that correspond directly to VB projects (see **Figure 8**).

Figure 8. You can customize which parts of a VB project print.

Project components and files

Before I wrap up this month, I want to mention what the structure of a project looks like and what the files on disk are. The project is stored with a ".VBP" extension and is simply a text file. A VB form is also just a text file, with an extension of ".FRM." I've shown both of these opened up on Notepad windows in **Figure 9**. A third type of component that I haven't discussed yet, but that you've probably seen here or there, is the Module. It has an extension of .BAS, and it contains code that isn't tied to a form—much like a separate FoxPro procedure file.

There you go. I've been through six or eight VB 6.0 books, and they've each taken 40 to 50 pages to cover this material. You now know more than what readers of those

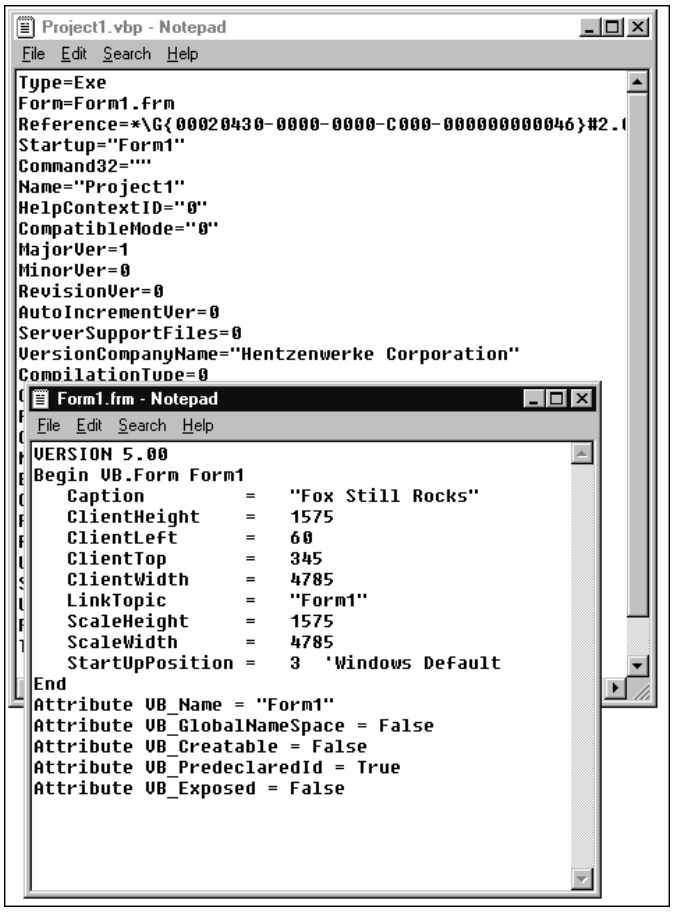

Figure 9. VB projects and forms are simply text files.

Stupid VB Tricks

You know how there are certain behaviors in VFP that just don't seem right? The ones that make you ask yourself, "What *were* they thinking?" And when the explanation offered is, "It's that way *by design*," you just roll your eyes. Take heart; VB has its share of incredulous behaviors. Each month, I'll point out a couple of gotchas that could bite you when you're not expecting it.

Stupid VB trick #1

Add a command button to the form, double-click to open up the code snippet for that button's click event, and enter some code. Then delete the button. The code snippet stays put—it's not deleted.

Stupid VB trick #2

Add a command button to a form. Double-click to open up the code snippet for that button's click event, and enter some code. Change the name of the button from "Command1" to "MyButton." The code snippet for "Command1" stays there—with the code you entered. Meanwhile, running the form and clicking on MyButton won't do anything, because the code is still attached to Command1 (even though it's nonexistent), not MyButton.

Stupid VB trick #3

Repeat the steps in #2, but then add another command button. It will be named Command1—and the previously orphaned code will now be attached to the new button!

books do, and you're probably not done with that can of Jolt. Not bad, eh? Repeat after me: "Thank you, thank you, Sam-I-Am, I *like* VB6, Sam-I-Am!"

I've been pretty flip about the use of some terminology—events, methods, procedures, code snippets, windows, and so on. Next month, I'll tighten up on the lingo and cover the big commands and functions in VB 6.0. Stay tuned! ▲

Whil Hentzen is the editor of *FoxTalk*.