PRE201: Introduction to Visual FoxPro

This Workshop introduces you to Visual FoxPro and the basics of how to use it. Plan to attend this session if you're a beginning Visual FoxPro developer or you haven't worked in Visual FoxPro before, and you would like to get more out of the DevCon that follows.

In the beginning, was the dot

What is Visual FoxPro?

- It’s a standalone tool for data manipulation
- It’s a development tool for standalone, LAN, client-server, COM and Web applications
- It’s a database engine
- It’s a programming language
- It’s part of Visual Studio
- It’s an integral part of Microsoft Windows
- It’s a religion

Well, yes and no. The first four are probably true. The last three are probably not, although you may find adherents who believe some, all, or none, of those statements. They also say there’s one born every minute. Let’s dig a little, starting with a little history lesson.

Wayne Ratliff was the programmer, working for Martin Marietta and subcontracting for the Jet Propulsion Laboratory, who started to create a natural-language-style database engine and manipulation language on his IMSAI 8080 computer, in assembler, in his spare time, in order to improve his chances in the football pool. One thing lead to another, and he was soon marketing the product as dBASE. It was purchased by Ashton-Tate, then Borland, and is now owned by dBASE, Inc. It was one the key products in the making of the “PC Revolution” of the 1980s that lead to a PC on every desk. In its heyday, a number of language-compatible “clones” such as FoxBase, Clipper, dbMAN and many others competed fiercely for the hearts and minds and wallets of developers.

Fox Software, based in Perrysburg, Ohio, was formed and run by Dr. David Fulton. (“Dr. Dave” as he was affectionately known, was a great showman, who delighted in meeting and presenting to his customers. He is a major reason that DevCon continues to this day.) Fox Software created a fast, interpreted version of the dBASE runtime and then broke the mold in going beyond the standard to introduce many additional features. FoxBase ran on Mac, DOS and Unix platforms. FoxPro, starting with version 2.5, supported Windows as well. Fox Software was acquired by Microsoft in March of 1992. While there was a Macintosh version of Visual FoxPro 3.0, subsequent versions run only on the Windows platforms.

In this paper, I look at how to learn Visual FoxPro. Mastering a computer language is similar to mastering another skill. You need to progress through the levels of novice, apprentice, and journeyman to reach the master level. Achieving mastery is not a subject that can be taught in a morning, nor covered in a short paper. But mastery starts with a good understanding of the fundamentals, and that is what I try to cover here. First, I look at data, as is it the data that is really what it is all about – the application is just a way to better manage the data. Second, I look at the language itself, how to interact with the data, read it in and display it. The third section goes beyond the basic procedural parts of the language into the power tools, control and objects that build applications. Finally, the fourth section tries to pull together all of the previous sections, and provide a perspective and philosophy of how an entire application should be put together.

Part I - It's the Data

Before we can plunge headfirst into developing FoxPro applications, a glossary and a bit of abstract theory will make the applications work much better.

Terminology

A **field** is a single piece of information, such a person's first name or an item's price. Divide up your information so that fields can stand on their own, and don't need to be sub-divided when you are processing. For example, if your part number is composed of a part type, a region code, a rating and a sub-part number, such as AN-33-X4-1234, it can often be better to break that information into separate fields and combine it when needed, rather than try to be constantly splitting the field when looking for all parts from one region.

Each field has a single **datatype**. Data types hold a particular kind of information; have upper and lower limits on their capacity; and are restricted on what information they can hold. Fields may be character, integer, date, datetime (a combination of date and time), numeric, double, float, currency logical (true or false), memo (very long freeform text or binary data). Specialized datatypes exist to hold OLE information (general fields), Macintosh binary (picture fields), but are rarely used.

A collection of fields which hold a single piece of information are gathered together to form a **record**. For example, you might record a check received from a customer as:

Field name	**Type**	**Data**
Customer Number	Integer	4321
Check Number	Integer	5678
Amount	Numeric (9,2)	$910.11
Date Received	Date	22 July 2001

A collection of these records would form a **table**. The table of data can be viewed in many ways, but the standard form used by the Fox BROWSE command looks like this:

Custom er	Chec k	Amou nt	Date

1243	3121	232	01/01/1901
3232	43243	3343	02/02/2002
23232	42	43.34	03/03/1903

This geometry leads to other names for the items, records are often called **rows** and fields **columns.**

In FoxPro, there are usually three ways to do anything, or no way at all. To create this table we might type:

```
CREATE
```

And let the user interface guide us to what we wanted to create. Or we could type:

```
CREATE TABLE CheckRec (Customer I, Check I, Amount N(9,2), Date D)
```

Or:

```
DIMENSION laField[4,5]
dimension laField[4,5]
laField[1,1] = "Customer"
laField[1,2] = "I"
laField[1,3] = 4
laField[1,4] = 0
laField[1,5] = .F.
laField[2,1] = "CheckNo"
laField[2,2] = "I"
laField[2,3] = 4
laField[2,4] = 0
laField[2,5] = .F.
laField[3,1] = "Amount"
laField[3,2] = "N"
laField[3,3] = 9
laField[3,4] = 2
laField[3,5] = .F.
laField[4,1] = "ChkDate"
laField[4,2] = "D"
laField[4,3] = 8
laField[4,4] = 0
laField[4,5] = .F.
CREATE TABLE CheckRec FROM ARRAY laField
```

Each of these ways of creating a table has advantages and disadvantages. The first is easiest for a beginner; he or she is guided in the choices to be made. The second is quicker; one line of typing and you're done. The third form, though, leads to the most flexibility and control.

Design and Normalization

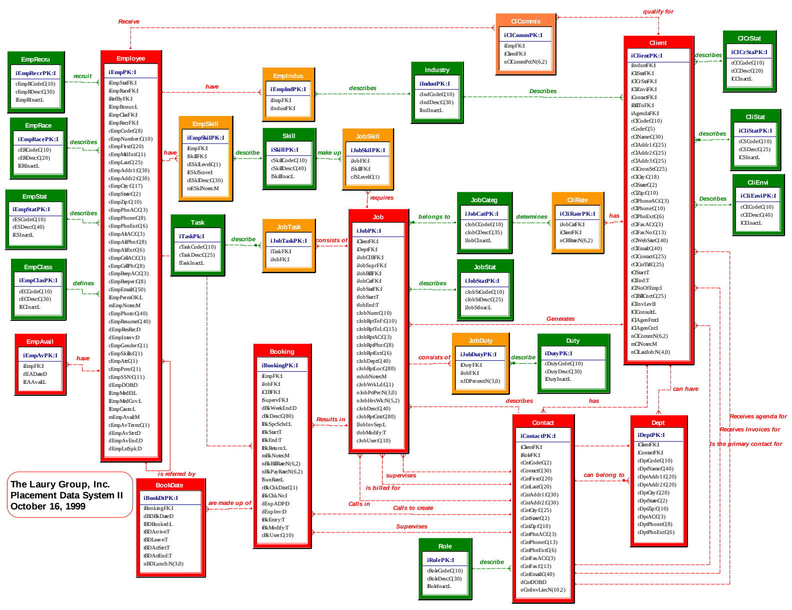
Now that you have a basic grasp on how to create tables of data, you face the task of determining what goes where. Should all of the data be on one large table or should the information be sorted into several smaller tables?

The answer is nearly always the latter – use multiple tables to separate different items. There are exceptions, of course. If you are creating a quick and dirty set of data to use for a week, and then throw away, stuffing everything into one table will make the processing – your work – easier. But be careful! I once created a set of labels to invite some customers to a golf tournament. Five years later, the outgrowth of that system was charting the financial course for a 3000-person company.

The rules for splitting up items between tables are called *normalization*. Each table (also called an *entity*) contains all of the attributes (fields) for one item. If an entity can have more than a quantity of one attribute (for example, an order may have a series of line items), then those items go into a second table. Tables are *related* to one another by a description of how information in one table is associated with information in another. In the order example, the order number is stored in both tables, and we can say that there is a *one-to-many* relationship between the tables: one order may have many line items. The typical relations are:

- one-to-many: parent to many children, such as an order to order item
- zero-or-one-to many: a lookup table may be referenced in none, some or many other records
- one-to-one: data split across several tables for performance

These relationships defined in the design stage will not be of much use if the people using the database system can add, subtract or modify data in all of the tables without regard for the design. For that reason, FoxPro provides *relational integrity* to the database, through the use of triggers (code automatically fired when a data change takes place) and stored procedures (code stored within the database container) to enforce the relationships defined on the data.



Much more information on data design and normalization can be found on the internet or a good introductory text on database design.

Reading and writing data

Interactive

REPLACE, APPEND BLANK, BROWSE, EDIT

VFP Commands

REPLACE, APPEND, COPY, DELETE

SQL Commands

INSERT, UPDATE, DELETE

Transactions and buffering

There are some times in a relationship when you are not ready to commit. There are times when changes to a relational database should not be committed, either, perhaps because a change involves multiple tables and the changes are not yet complete, or because one of a set of changes failed to update properly. Enter buffering and transactions.

Buffering allows data to be stored on the local machine until all changes are ready to be made at once. This prevents tying up shared data resources until the last possible minute, and allows local processing to examine old and new values of data (each stored in a separate buffer) to determine how a change should be handled.

Transactions are the other end of the process, once you are ready to commit the data. A transaction locks data records as the changes are committed, and allows the entire set of updates ("a single transaction") to either be completed successfully or rolled back completely. Visual FoxPro provides support both for local (DBF-based) transaction processing and for transaction processing in a client-server arrangement.

Client server data

Why client-server?

There are three practical reasons to move a DBF-based application to a client-server architecture:

Too much data: VFP has a physical limit of 2Gb for a single table or memo file, but the limit can often be hit earlier, when the amount of time to PACK or REINDEX a VFP table exceeds the recovery time of a client server system following a crash. As more operations move towards 24x7, this factor has become increasingly important.

High-security information: While it is possible to crack a client-server database, the likelihood is far greater in a VFP system where clients must have access to the underlying tables

Low bandwidth: VFP was designed to thrive in a network environment, and it takes advantge of the large bandwidth to locally cache file headers, indexes and records. While this results in remarkable Rushmore performance on a LAN, it can result in unacceptably slow performance in a WAN situation.

There are numerous other, more pragmatic reasons to move to a client-server architecture, including the need for flexibility in distributing processing, and the political necessities of some environments.

The ACID test – Atomicity, Consistency, Isolation, Durability

Many organizations justify their need for a client-server architecture with the ACID test. A well-designed client server system meets all of the following criteria:

Atomicity: all data changes within a transaction are treated as a single indivisible unit, where all are completed or all are rolled back as a unit.

Consistency: the integrity rules for the database are enforced at all times, so that the database is always in a consistent and valid state.

Isolation: results of a transaction are invisible until they are complete, and one transaction's intermediate results should not affect another transaction. If one transaction causes the initial conditions of a second transactions to fail, the second transaction should fail.

Durability: once the data has been committed, it will be stored and retrieved even if the system suffers the loss of a hard disk or a processor or any other component that doesn't destroy the system. Transactions cannot be "lost."

Part B – It's the coding

VFP: commands, functions, object, PEMs and design surfaces

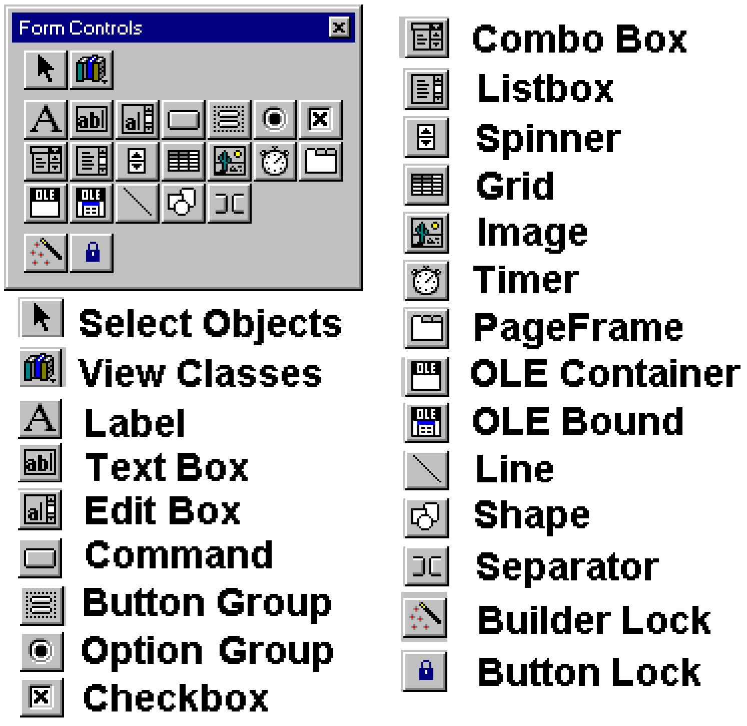
XBASE commands: USE, BROWSE, REPLACE, APPEND, LOCK, DO, IF, CASE

SQL commands: SELECT, INSERT, DELETE, UPDATE

Output: ? REPORT FORM, @ …SAY, CALCULATE, LIST/DISPLAY

Text Manipulation: Textmerge, LLFF, String functions

Controls



Text-based controls: Label, TextBox, EditBox, Spinner

Pick-list controls: ComboBox, ListBox

Buttons: CommandButtons, OptionButtonsGroups, CheckBoxes

Containers; Grids, PageFrames, CommandButtonGroups

Graphical elements: Images, Lines, Shapes, Separators (toolbars only)

OLE: both OLEBound (associated with data) and Container (OCX) controls

Invisible control: timer

Working with controls

Controls, in their simplest form, are simply the new versions of SAYs and GETs. But controls are so much more! They offer much finer control of the individual objects properties, the ability to change these properties at run-time, and the ability to define the code which should run when an event happens. Events are also much more numerous, giving us the chance to create interfaces more responsive to the user.

The controls supplied with Visual FoxPro are the starting point. These controls can (and should!) be subclassed to create our own custom controls. Multiple controls may be combined to form complex controls, better reflecting the complexity and business rules of the particular application. Custom controls can be created, saved and reused.

Properties

Properties describe a characteristic of a control. Most are available both at design and run-time, one exception being the class properties, which are read-only when the control is created. These properties are the data of the control which is "encapsulated" with the control. Most controls share a number of common properties:

Common Property	**Purpose**
Top, Height, Left, Width	The location on the form of the object, if a visual control
Comment	Probably the single most important property; let's you figure out what you were doing when you return to it
BaseClass, Class, ClassLibrary	The pedigree of the control
Name	How the control is named by all code within the form which refers to it
Visible	Whether the control should appear
DragMode, DragIcon	Behavior during mouse drag operations
HelpContextID	Your hook from the control into your custom help file
FontName, FontSize, FontItalic, FontBold, FontOutline, FontStrikethrough, FontUnderline	For all text-based controls

ColorScheme, ColorSource, BorderColor, ForeColor, DisabledForeColor, BackColor, DisabledBackColor	Colors.

Then, there are properties specific to an individual control or two: Interval: how often a Timer control fires,

SpinnerHighValue, SpinnerLowValue, and a slew of others!

Events

Events occur when the user has taken some action, or programmatically when a control changes status. It is not possible to define new events. Common events include:

Event	**Purpose**
Init	Code run once when the control is created
Destroy	Code run when the control is released
DragDrop, DragOver	How to behave when a dragged object is over and dropped
MouseMove	Mouse movement over a control
Click, RightClick, DblClick, MouseDown, MouseUp	Both mouse buttons!
Error	How errors are handled
GotFocus, LostFocus	Code when control is tabbed to or clicked on
When, Valid	Our old friends.

Methods

Methods tend to be more individual to the controls, as they describe the unique behavior of the control. The most common are:

Method	**Purpose**
Drag	What to do if control is dragged
Move	Moves controls within a container
SetFocus	Programmatically "send" focus to a control
Refresh	Each time the control needs

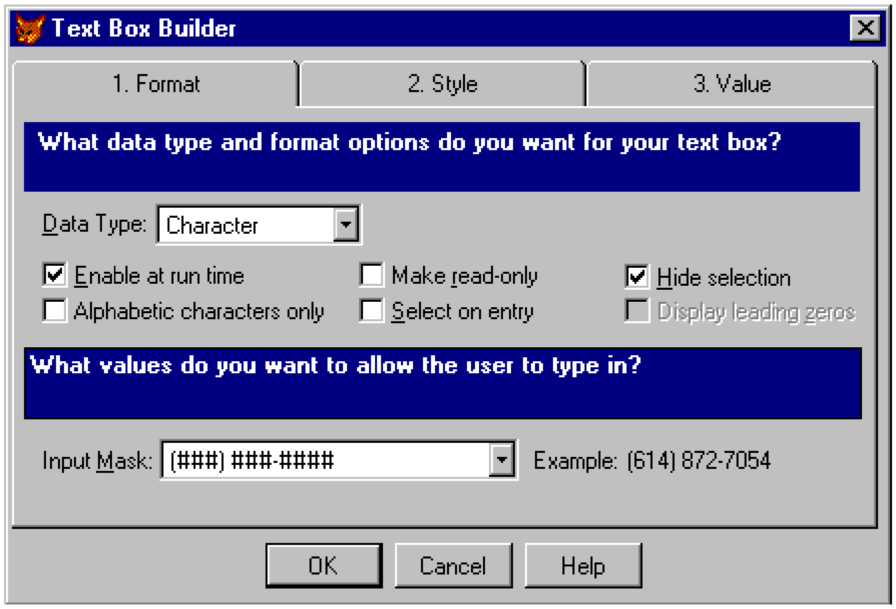
	to be redisplayed, the code runs.

Individual methods include the Reset method for the Timer control, and the DoVerb methods for the two OLE controls.

Manipulating controls: Property Sheet & Builders

Property sheets, like our demo program, give us access to the various programmable and fixed properties, events and methods. Builders give an alternative view into a limited number of properties, simplifying the construction of objects.

Property sheets are tools that let us climb under the hood and tweak all the levers and dials. Builders don't have that depth, but don't have the complexity, either.



What data type and format options do you want for your text box?

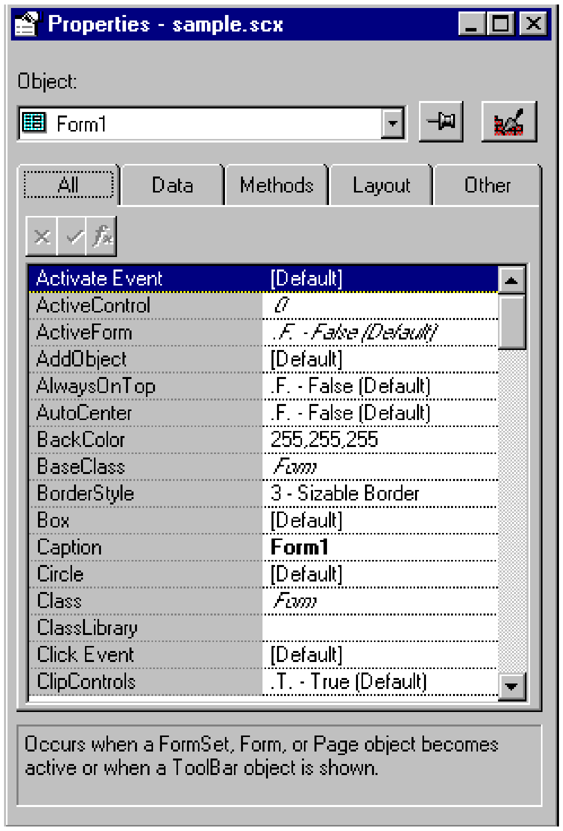
Data Type: Character

- [x] Enable at run time
- [ ] Make read-only
- [x] Hide selection
- [ ] Alphabetic characters only
- [ ] Select on entry
- [ ] Display leading zeros

What values do you want to allow the user to type in?

Input Mask: (###) ###-#### Example: (614) 872-7054

A Typical Builder



Activate Event	[Default]
ActiveControl	*0*
ActiveForm	*.F. - False (Default)*
AddObject	[Default]
AlwaysOnTop	.F. - False (Default)
AutoCenter	.F. - False (Default)
BackColor	255,255,255
BaseClass	*Form*
BorderStyle	3 - Sizable Border
Box	[Default]
Caption	**Form1**
Circle	[Default]
Class	*Form*
ClassLibrary	
Click Event	[Default]
ClipControls	.T. - True (Default)

Occurs when a FormSet, Form, or Page object becomes active or when a ToolBar object is shown.

A Property Sheet

Creating your own custom controls

You will want to create your own set of custom controls so that you can manipulate the base classes and modify their properties and behaviors. It is a good idea to create a set of 'generic' but subclassed controls and use these, rather than the standard toolbar, as the basis for prototyping screens. Create a custom control by placing a control on a form, highlighting it, then selecting 'Save as class..." from the "File" menu.

Most of the literature on the subject of OOP states that it is a learning process, and a different way of looking at problems. Anticipate that your first solution may not always be the best one. If possible, prototype your first development in Visual FoxPro on a system you can throw away. In subsequent systems, beware of the problem of over-engineering a solution.

Don't be afraid to throw out your work.

Don't subclass it to death - three or four levels is about all you'll comprehend.

Event model

In years past, FoxPro was based on a procedural model, when code started at the top of a procedure and executed line-by-line until it was done. Message loops and “Get-less READs” could be used to simulate an event-driven system, but there was a tricky series of obscure behaviors to get it to work. With the advent of Visual FoxPro, a new, clean event model was introduced that allows a much simpler and more comprehensible approach to providing a truly event-driven system.

What are Events?

"An action, recognized by an object, for which you can write code to respond. Events can be generated by a user action, such as clicking the mouse or pressing a key, by program code, or by the system, as with timers."

- FoxPro Help File

Event firing sequences

Probably some of the most difficult functionality to understand. Typically, a container cannot perform its action until its contents exist, therefore, objects are created from the inside out: textbox–›column–›grid–›page–›pageframe, and destroyed in the opposite fashion, from the outside in, imploding.

How to program for events?

The Foundation read is no more, except for legacy code we attempt to migrate to Visual FoxPro. Forms are it. The concept behind Visual FoxPro is that the initial application environment can be set up, a menu can be hoisted to the top of the screen, and READ EVENTS will hold the entire application together until the user chooses to quit. Time will tell if this model proves robust enough for commercial applications.

So what are the new events?

Event	Applies to:	What it does / what to put there
INIT	All but Column, Header, Page, Separator	Fires when the object is created, optionally accepts parameters. If it returns .F., object is not created. Contained objects fire before containers, in the order added.
ERROR	Same	Fires when an error occurs in the method of an object - passes error #, method name and line number. Fires before ON ERROR.
DESTROY	Above, plus CommandGroup and OptionGroup	Code runs just before an object is released. Containers fire before contents.
DRAGOVER, DRAGDROP	Above, plus Cursor, Custom, DataEnvironment, FormSet, Relation, Timer	Fires during and upon completion respectively of a drag & drop operation. Code must include parameters statement to accept the dragged object reference and mouse coordinates.
MOUSEMOVE	Column, but not to above, plus OLEControl, OLEBoundControl	Tracks mouse movements over an object. Also passes status of Ctrl-Alt-Shift keys, as well as left, middle and right mouse button statuses.
CLICK, MOUSEDOWN, MOUSEUP	Not to Column, otherwise same as above	Mouse click

UIENABLE	CheckBox, ComboBox, CommandButton, CommandGroup, Container, Control, EditBox, Grid, Image, Label, Line, ListBox, OLEBoundControl, OLEControl, OptionGroup, PageFrame, Shape, Spinner, TextBox	Fires when control becomes visible because of activation of container, such as PageFrame.
RIGHTCLICK	Above, plus Form, Header, OptionButton, OptionGroup, but NOT OLEBoundControl, OLEControl	Right mouse click on control.
GOTFOCUS, LOSTFOCUS	CheckBox, ComboBox, CommandButton, Container, Control, EditBox, Form, ListBox, OLEBoundControl, OLEControl, OptionButton, Spinner, TextBox	Occurs when the control is tabbed to, or clicked on.
VALID, WHEN	CheckBox, ComboBox, CommandButton, CommandGroup, EditBox, Grid, ListBox, OptionButton, OptionGroup, Spinner, TextBox	Good old WHEN and VALID, fire before accepting a change (after receiving focus) and after a change is made.
ERRORMESSAGE	CheckBox, ComboBox, CommandButton, CommandGroup, EditBox, ListBox, OptionButton, OptionGroup, Spinner, TextBox	When VALID returns a .F., allows display of an error message. "Included for backward compatibility"
MESSAGE	same as above	Displays status bar text. Another " backward compatibility." Property StatusBarText provides similar capabilities.
KEYPRESS	CheckBox, ComboBox, CommandButton, EditBox, Form, ListBox, OptionButton, Spinner, TextBox	Allows processing of input keystroke-by-keystroke, rather than waiting for input to be completed.
MOVED	Column, Container, Control, Form, Grid, OLEBoundControl, OLEControl, PageFrame, Toolbar	Fires when the object has been moved.
RESIZE	same	Fires when the object has been resized.

InteractiveChange, ProgrammaticChange	CheckBox, ComboBox, , CommandGroup, EditBox, ListBox, OptionGroup, Spinner, TextBox	What UPDATED() always should have been, but at a finer level. Fires each time a change is made via mouse or keyboard, even before focus has shifted from the control. INTERACTIVE detects user changes, PROGRAMMATIC changes performed in code.
ACTIVATE, DEACTIVATE	Form, FormSet, Page, Toolbar	Similar to the 2.x Screen's show clause. Occurs when container gets the focus or Show() method runs. Toolbar.Hide() also runs DEACTIVATE
RANGEHIGH, RANGELOW	ComboBox, Listbox, Spinner, TextBox	Dual functions. For ComboBox and ListBox, returns the initially selected element when the control gets the focus. For Spinners & TextBoxes acts as a RANGE test, returning a numeric when focus to the control is lost.
DOWNCLICK	ComboBox, ListBox, Spinner	Not to be confused with MOUSEDOWN, fires when the down- or up-ward-pointing arrow is pressed.
LOAD, UNLOAD	Form, FormSet	Load occurs after Init, but before Activate and GotFocus. UnLoad is the last event to fire.
PAINT	Form, Toolbar	When the item re-paints. CAUTION: don't RESIZE or refresh() objects within PAINT or a "cascading" series may occur!
BEFOREOPENTABLES, AFTERCLOSETABLES	Data Environment	Wrappers around the automatic behavior of the Data Environment. Occurs before OpenTables() method and after CloseTables() methods.
AFTERDOCK, BEFOREDOCK, UNDOCK,	Toolbar	Code which can run while user is manipulating a toolbar.
BeforeRowColChange, AfterRowColChange	Grid	Before the Valid of the row or column of the cell being left, and after the When of the cell being moved to.
DELETED	Grid	When user marks or unmarks a row for deletion.
SCROLLED	Grid	User movement, parameter will return whether by cursor keys or scroll bars and which one.
DROPDOWN	ComboBox	Fires after DOWNCLICK, to allow interactive changes to the contents of the drop down list.
TIMER	Timer	Fires when Timer is enabled and Interval has passed.
QUERYUNLOAD	Form	Allows testing the ReleaseType property to determine if a form is being released using the close box or programmatically.
READACTIVATE, READDEACTIVATE, READSHOW, READVALID, READWHEN	Form	Similar to 2.x READ model, only works in 'Compatibility' modes

What to do now?

Experiment. 90% of the time the standard WHEN and VALID will provide all the functionality needed in data entry fields. Specialized input fields, such as Spinners, have finer control. Click is a more intuitive place to put button-firing code than VALID, but either (though not necessarily both!) work. Add new Events to your arsenal as the need arises. Anticipate some great third party tools that know how to really take advantage of all the new features.

The tools

- Project Managers
- Code Editor
- Form Designer
- Class Designer
- Menu Designer

Part III: Advanced topics

Object oriented analysis and design

COM and n-tier design

User Interface Design

Project Management

Part IV: Putting it all together

RTFM – VFP comes with an excellent set of resources. Browse the help file. There is a tremendous amount of well-organized material there.

Don't fight the tide. Learn to become one with the Fox by thinking the way the Fox thinks.

Get a framework

Get a support group: mailing lists, online communities, magazines & newsletters

Sharpen the saw – books and conferences

Top Ten Mistakes

About the Author

Ted Roche

Ted Roche & Associates, LLC

Ted Roche is president of Ted Roche & Associates, LLC, a consulting firm based in New Hampshire. He is author of *Essential SourceSafe*, co-author with Tamar Granor of the award-winning *Hacker's Guide to Visual FoxPro 6*, and a contributor to 5 other FoxPro books. A former Contributing Editor for FoxPro Advisor magazine, Ted is a Microsoft Certified Solution Developer, Systems Engineer and seven-time Microsoft Support Most Valuable Professional. Contact Ted at tedroche@tedroche.com

References

My 15-year immersion in Fox software and the other dBASE variants would have been a much shallower and weaker experience had it not been for the thriving online communities – CompuServe, the Fox Wiki, the ProFox mailing list – that have brought depth and meaning and humanity to the experience. Thanks to all of those who participated. If you are not yet involved in an online community, find one and, at the least, lurk in the background for a while. The support and education is invaluable, at a very reasonable cost.

Online communities

Nearly the granddaddy of them all, CompuServe has opened their forums to the web. Start at http://www.compuserve.com, and go from there. A direct link that works for me is http://go.compuserve.com/MSDevApps?loc=us - your mileage may vary.

Probably the most active FoxPro community on the web today is http://www.universalthread.com, with many well-known authors and speakers frequenting the site. A close runner-up is the repository of knowledge built up by its members at http://fox.wikis.com - a remarkable, organic site that allows visitors to add, edit and enhance the existing web site. But there are a number of other worthwhile sites. Rod Paddock hosts http://www.foxforum.com . Ed Leafe hosts the ProFox mailing list; you can sign up at http://www.leafe.com, and the Virtual FoxPro User Group is online at http://www.vfug.org/.

Books

I'm a big advocate of learning by reading. There are a lot of fascinating books out there, and I am always in the process of reading a few at a time. If this isn't your style, consider hanging around with people who are keeping up with the latest books – by visiting their web sites, listening to them at conferences and user group meetings, or working with them.

Steven Black has an extensive reading list at http://www.stevenblack.com - look for "bookshelf." Whil Hentzen maintains another list at http://www.hentzenwerke.com - his is called "The Stacks." Between those two lists alone, you will find most, if not all, of the books I would recommend as well.

Magazines

FoxPro devotees should keep up on what's happening in the FoxPro world by reading everything that's available out there. Of course, that would mean that there would be no time left for other activities, like programming, or sleep, or this "life" thing I keep hearing about. My advice is to subscribe to all the magazines that interest you and at least skim the table of contents. That way, you can read the articles that essential for you to know right now, and store away knowledge that there are other articles you can get back to later. FoxTalk (http://www.pinnaclepublishing.com/ft) and FoxPro Advisor (http://www.advisor.com/www/FoxProAdvisor) are the leading magazines in the field.

User Groups

A user group typically meets one evening a week to swap stories, network, demonstrate some code or product, and support each other. These offer a great chance to get out of the house, meet with people of similar interests, find new jobs, locate consultants or consulting opportunities, and keep up with what is going on in the industry. There are lists of user groups at http://fox.wikis.com/wc.dll?Wiki~CategoryUserGroups, http://www.bostonusergroups.com, or hit your favorite search engine to locate a group near you? No groups nearby? Start your own!

Links

An excellent explanation of the ACID principles is available on **http://www.arsdigita.com/books/panda/databases-choosing**

Lammers, Susan, *Programmers at Work,* Microsoft Press, 1986, features a fascinating interview with Wayne Ratliff (while he had an office at Ashton-Tate, and the IBM AT was considered a bargain at $6,000)