1001 Things You Wanted To Know About Visual FoxPro

Marcia Akins
Andy Kramek
Rick Schummer

Hentzenwerke Publishing
Dedications

Andy Kramek
This work is dedicated to my father, who was so proud when I began to have some of my writing published, but died shortly before completion of this, latest, book which I know would have made him even more happy.

Marcia Akins
To my sister Nancy who taught me that it's never too late to try and without whose help and support I would not have been able to do all that I have in the past year. Thank you, I would not have been able to write this book without you.

Rick Schummer
This book is dedicated to the memory of my Grandpa, Richard Holden. Grandpa gave me a pep talk on the darkest day of my college career. I was actually considering giving up my path to a degree in Computer Science from Oakland University. This man never finished high school, yet is one of the wisest people I have met in this lifetime. Had it not been for his perspective, I might not be the computer geek that I am today. For this direction I am eternally grateful.
List of Chapters

Introduction 1
Chapter 1: Controlling the VFP Environment 3
Chapter 2: Functions and Procedures 31
Chapter 3: Design, Design and Nothing Else 55
Chapter 4: Basic Controls 85
Chapter 5: Combos and Lists 127
Chapter 6: Grids: The Misunderstood Controls 159
Chapter 7: Working with Data 197
Chapter 8: Data Buffering and Transactions 239
Chapter 9: Views in Particular, SQL in General 277
Chapter 10: Non-Visual Classes 313
Chapter 11: Forms and Other Visual Classes 363
Chapter 12: Developer Productivity Tools 391
Chapter 13: Miscellaneous Things 423
Chapter 14: Project Management 461
Chapter 15: Project Objects and Projecthooks 479
Chapter 16: Creating Reports 505
Chapter 17: Managing Reports 529
# Table of Contents

Our Contract With You, The Reader                                          xix  
Acknowledgements                                                                   xxii  
About Us                                                                                   xxivii  
How to Download the Files                                                      xxvi  

## Introduction 1
What is this book? 1  
A word about the code in this book 1  
So who is this book for? 1  
What is in this book? 2  
What is not in this book? 2  
Where do you start? 2

## Chapter 1: Controlling the VFP Environment 3
Starting Visual FoxPro 3  
Configuration files 3  
  How to specify a config.fpw file 4  
  How VFP locates its configuration file 4  
  How VFP starts up when no configuration file is found 4  
  Including a configuration file in the project 4  
  How to suppress a configuration file 5  
  How to determine which configuration file is being used 5  
What goes into the configuration file? 5  
  Special settings 5  
  SET Commands 6  
  Commands 6  
Giving VFP a path 7  
  How VFP looks for files 7  
  Setting the default directory 7  
  Using the SET PATH command 8  
  Where am I? 9  
  How to set a path programmatically 9  
Making sure VFP is only started once 10  
  Using a 'semaphore' file 10  
  Using the Windows API 11  
  Combination of semaphore and Windows API 12  
SET Commands and DataSessions 13  
  What exactly does “Default DataSession” mean? 13
So can I have a “public” Datasession? 14
How can I ensure SET commands apply to a private data session? 14
Adding code to BeforeOpenTables() 15
Suppressing auto-open tables 16
Creating an environment setting class 16
How do I get rid of the system toolbars? 18
The system toolbar "Gotcha!" 18
Can I make use of keyboard macros in VFP? 19
How can I construct a more complex macro? 19
What is a "Macro Set"? 20
What's the difference between a macro and an On Key Label? 21
How do I create a 'Splash' screen? 22
How do I run my splash screen? 22
An alternative to the splash screen 23
How to wallpaper your desktop 23
So how can I get the size of the current Screen area? 24
Do I really need to create all these bitmaps? 24
A toolbar 'gotcha!' 25
Tidying up your development environment 25
Closing VFP down 27
What is an On ShutDown procedure? 27
What triggers an On Shutdown procedure? 27
What goes into an On Shutdown procedure? 28

Chapter 2: Functions and Procedures 31
How shall we proceed? 31
Parameters (an aside) 32
By reference, by value? 32
How do I know what was passed? 33
How should I position my parameters? 33
How can I return multiple values from a function? 33
What about using named parameters? 34
Passing parameters optionally 35
Date and time functions 35
Elapsed time 35
Date in words 36
Calculating Age 37
What date is the second Tuesday in October of 2000? 37
Setting up a payment schedule 38
What date is ten business days from today? 40
Gotcha! Strict date format and parameterized views 41
Working with numbers 41
Converting numbers to strings 41
Gotcha! calculations that involve money 42
Chapter 4: Basic Controls

What do we mean by 'basic'?

Text boxes
  - Text box label class
  - Date text box
  - Incremental search text box
  - Numeric text box

Handling time
  - A time entry text box
  - A time entry composite class
  - The true time spinner

Blinking labels

The expanding edit box

Calendar combo

Command buttons

Gotcha! Programming the logical controls

Pages and page frames

Chapter 5: Combos and Lists

Combo and list box basics
  - List and ListItem collections
  - When do the events fire?
  - How do I bind my combo and list boxes?
  - How do I refer to the items in my combo and list boxes?
  - What is the difference between DisplayValue and Value?
  - What's the difference between RowSourceTypes "alias" and "fields"?

How do I make my combo and list boxes point to a particular item?

Quickfill combos

How do I add new items to my combo and list boxes?

How do I filter the items displayed in a second combo or list box based on the selection made in the first?

A word about lookup tables

Generic lookup combos and lists

So what if I want to bind my combo to a value that isn't in the list?

How do I disable individual items in a combo or list?

How do I create a list box with check boxes like the one displayed by Visual FoxPro when I select "View Toolbars" from the menu?

A mover list class

What if I need to display hundreds of items in my combo box?

Chapter 6: Grids: The Misunderstood Controls

When do the events fire?
  - Gotcha! Grid's valid fires before the valid of the current control
What is the difference between ActiveRow and RelativeRow? 161
ActiveColumn does not really tell you which is the active column 162
How do I highlight the current grid row? 162
Keeping a grid from scrolling when the user tabs off the last column 164
How do I create multiline headers? 165
Gotcha! Scroll event does not fire when cursor keys scroll the grid 169
Using tool tip text instead of multiline headers 171
How do I change the grid's display order? 172
How do I control the cursor? 173
How do I display the last full page of a grid? 173
How do I use a grid to select one or more rows? 175
How do I give my multiselect grid incremental search capability? 176
How do I use DynamicCurrentControl? 178
How do I filter the contents of a grid? 180
So what about data entry grids? 182
How do I add new records to my grid? 182
How do I handle row level validation in my data entry grid? 184
How do I delete records in my grid? 187
How do I add a combo box to my grid? 190
Conclusion 195

Chapter 7: Working with Data 197
Tables in Visual FoxPro 197
Some basics 197
How to open the specific table you want to use 197
How to get the structure of a table 199
How to compare the structures of two tables? 200
How to test for the presence of a field in a table 202
How to check if a table is being used by another user 204
What is exactly is a cursor? 206
Indexes in Visual FoxPro 208
Types of indexes 209
How to get information about an index 209
How to test for the existence of an index tag 210
Using candidate (and primary) keys 211
What is a "surrogate key"? 211
Managing user-entered keys 215
Using indexes with bound tables 217
How to index mixed data types when creating a compound key 218
How to index a buffered table 218
Working with the database container 220
Using long table names 220
Using long field names - don't!! 221
Using database containers 222
How to validate a database container 223
How to pack a database container 223
Moving a database container 224
Renaming a database container 226
Managing referential integrity in Visual FoxPro 227
Limitations of the generated RI Code 228
Using compound keys in relationships 229
What about other RI options? 231
Using triggers and rules in Visual FoxPro 231
So what's the practical difference between a 'trigger' and a 'rule'?
Why, when adding a trigger to a table, does VFP sometimes reject it?
Can I temporarily disable a trigger or rule then?
How do I actually create my trigger and rule procedures?
How do I add a trigger to a table?
So when should I use a trigger?
And when should I use a rule?
Must a trigger or rule always refer to a single function?
Chapter 8: Data Buffering and Transactions 239
Using data buffering 239
Where are we coming from?
What do we mean by 'buffering' anyway?
What does all this mean when creating data-bound forms?
So just how do I set up buffering in a form?
So what mode of buffering should I use in my forms?
Changing the buffer mode of a table
IsChanged() - another function that FoxPro forgot?
Using TableUpdate() and TableRevert() 249
Managing the scope of updates
TableUpdate()'s second (force) parameter
Specifying the table to be updated or reverted
Conclusion
How can I handle 'save' and 'undo' functionality generically?
The design of the form class
Using the new form class
Detecting and resolving conflicts 256
The role of OldVal() and CurVal()
So how do I actually detect conflicts?
OK then, having detected an update conflict, what can I do about it?
Conflict resolution sounds fine in theory, how does it work in practice?
Using transactions 269
When do I need a transaction?
What effect does rollback have on my data?
How does a transaction impact on FoxPro's locking mechanisms?
### Can I use multiple transactions simultaneously? 272
### Some things to watch for when using buffering in applications 274
- Cannot use OLDVAL() to revert a field under table buffering 274
- Gotcha! Row buffering and commands that move the record pointer 275

### Chapter 9: Views in Particular, SQL in General 277
- Visual FoxPro views 277
  - What exactly is a view? 277
  - How do I create a view? 278
  - When should I use a view instead of a table? 279
  - Hang on! What is a parameterized view? 280
- How do I control the contents of a view when it is opened? 283
- Why do changes made in a view not get into the underlying table? 284
- Why create a view that is simply a copy of an existing table? 285
- What is the best way to index a view? 288
- More on using views 289
- SQL in Visual FoxPro 294
  - Joining tables 295
  - Constructing SQL queries 296
  - How to Check the results of a query 297
  - How to extend a SQL generated cursor 299
  - How to check your query's optimization 306
- Which is better for updating tables, SQL or native FoxPro commands? 311

#### Conclusion 311

### Chapter 10: Non-Visual Classes 313
- How can I make use of INI files? 313
  - Overview 313
  - Initializing the manager 314
  - The default file 314
  - File registration 316
  - The public interface 316
  - Using the INI file manager 320
- How to select a different work area, OOP style! 322
  - Overview 322
  - A word on creating the selector object 323
  - How the selector class is constructed 323
  - Using the selector class 325
- How can I manage paths in a form's dataenvironment? 327
  - The data path manager class 328
  - The path management table 329
  - The path management class 330
  - Using the data path manager 333
<table>
<thead>
<tr>
<th>How can I manage forms and toolbars in my application?</th>
<th>335</th>
</tr>
</thead>
<tbody>
<tr>
<td>The managed form class</td>
<td>335</td>
</tr>
<tr>
<td>The managed toolbar class</td>
<td>338</td>
</tr>
<tr>
<td>The form manager class</td>
<td>339</td>
</tr>
<tr>
<td>Using the form manager</td>
<td>350</td>
</tr>
<tr>
<td>How can I track and handle errors?</td>
<td>352</td>
</tr>
<tr>
<td>Classifying Visual FoxPro's errors</td>
<td>353</td>
</tr>
<tr>
<td>Logging errors</td>
<td>354</td>
</tr>
<tr>
<td>How can I simplify getting messages to my users?</td>
<td>357</td>
</tr>
<tr>
<td>The standard message table</td>
<td>357</td>
</tr>
<tr>
<td>The message handling class</td>
<td>358</td>
</tr>
<tr>
<td>Using the message handler</td>
<td>361</td>
</tr>
<tr>
<td>Conclusion</td>
<td>362</td>
</tr>
</tbody>
</table>

**Chapter 11: Forms and Other Visual Classes** 363

<table>
<thead>
<tr>
<th>How do I make forms fill the screen regardless of screen resolution?</th>
<th>363</th>
</tr>
</thead>
<tbody>
<tr>
<td>How do I create resizable forms?</td>
<td>366</td>
</tr>
<tr>
<td>How do I search for particular records?</td>
<td>370</td>
</tr>
<tr>
<td>How do I build SQL on the fly?</td>
<td>374</td>
</tr>
<tr>
<td>How can I simulate the Command Window in my executable?</td>
<td>377</td>
</tr>
<tr>
<td>Wrappers for common Visual FoxPro functions</td>
<td>379</td>
</tr>
<tr>
<td>Presentation classes</td>
<td>381</td>
</tr>
<tr>
<td>Postal code lookup class</td>
<td>381</td>
</tr>
<tr>
<td>Generic log-in form</td>
<td>385</td>
</tr>
<tr>
<td>The lookup text box class</td>
<td>387</td>
</tr>
<tr>
<td>Conclusion</td>
<td>389</td>
</tr>
</tbody>
</table>

**Chapter 12: Developer Productivity Tools** 391

| Form/class library editor                                          | 391 |
| Using the SCX/VCX editor                                           | 392 |
| Form Inspector                                                     | 393 |
| The table information page                                         | 394 |
| The form information page                                          | 394 |
| The object list page                                               | 396 |
| Construction of the inspection form                                | 397 |
| Our industrial strength grid builder                              | 403 |
| Resizing grid columns properly                                     | 405 |
| Renaming columns and their controls appropriately                 | 406 |
| The Custom Controls page of the grid builder                       | 407 |
| Adding method code to grid columns                                 | 408 |
| Class cataloger                                                    | 409 |
| What VCXList does                                                  | 410 |
### A wrapper for ‘modify class’ 410
### A form/class library documentation utility 412
### Share.prg - a class browser add-in 415
### A kinder finder 416
### Conclusion 421

**Chapter 13: Miscellaneous Things** 423

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using the Visual FoxPro debugger</td>
<td>423</td>
</tr>
<tr>
<td>Features of the debugger windows</td>
<td>423</td>
</tr>
<tr>
<td>Configuring the debugger</td>
<td>425</td>
</tr>
<tr>
<td>Setting breakpoints</td>
<td>426</td>
</tr>
<tr>
<td>Useful breakpoint expressions</td>
<td>428</td>
</tr>
<tr>
<td>Writing code for ease of debugging and maintenance</td>
<td>430</td>
</tr>
<tr>
<td>Working with datasessions</td>
<td>433</td>
</tr>
<tr>
<td>How do I share datasessions between forms?</td>
<td>433</td>
</tr>
<tr>
<td>How do I change datasessions?</td>
<td>434</td>
</tr>
<tr>
<td>How do I get a list of all active datasessions?</td>
<td>435</td>
</tr>
<tr>
<td>Miscellaneous items</td>
<td>435</td>
</tr>
<tr>
<td>What is the event sequence when a form is instantiated or destroyed?</td>
<td>435</td>
</tr>
<tr>
<td>How do I get a reference to a form's parent form?</td>
<td>437</td>
</tr>
<tr>
<td>How do I get a list of all objects on a form?</td>
<td>437</td>
</tr>
<tr>
<td>How can I set focus to the first control in the tab order?</td>
<td>441</td>
</tr>
<tr>
<td>How do I return a value from a modal form?</td>
<td>441</td>
</tr>
<tr>
<td>How do I change the mouse pointer while a process is running?</td>
<td>443</td>
</tr>
<tr>
<td>How can I create a 'global' property for my application?</td>
<td>445</td>
</tr>
<tr>
<td>How can I 'browse' an array?</td>
<td>445</td>
</tr>
<tr>
<td>Windows API Calls</td>
<td>447</td>
</tr>
<tr>
<td>How do I find the file associated with a file type?</td>
<td>447</td>
</tr>
<tr>
<td>How can I open a file using Windows file associations?</td>
<td>449</td>
</tr>
<tr>
<td>How can I get the user's Windows log-in name?</td>
<td>450</td>
</tr>
<tr>
<td>How can I get directory information?</td>
<td>450</td>
</tr>
<tr>
<td>How can I get the number of colors available?</td>
<td>452</td>
</tr>
<tr>
<td>How do I get the values for Windows color settings?</td>
<td>453</td>
</tr>
<tr>
<td>How do I change the cursor?</td>
<td>454</td>
</tr>
<tr>
<td>How do I customize my beeps?</td>
<td>455</td>
</tr>
<tr>
<td>How do I find out if a specific application is running?</td>
<td>456</td>
</tr>
<tr>
<td>Using the DECLARE command</td>
<td>458</td>
</tr>
</tbody>
</table>
## Chapter 14: Project Management

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>What happens when building an executable?</td>
<td>461</td>
</tr>
<tr>
<td>How to use the project options to your advantage</td>
<td>462</td>
</tr>
<tr>
<td>How do you use a project’s Debug Info setting?</td>
<td>462</td>
</tr>
<tr>
<td>How do you use a project’s Encrypted setting?</td>
<td>463</td>
</tr>
<tr>
<td>How do you set a custom icon for an executable?</td>
<td>463</td>
</tr>
<tr>
<td>How do you manage files in the Project Manager?</td>
<td>464</td>
</tr>
<tr>
<td>How do you manage Servers from the Project Manager?</td>
<td>464</td>
</tr>
<tr>
<td>How do you set the project's object description?</td>
<td>464</td>
</tr>
<tr>
<td>How to set the executable version information</td>
<td>465</td>
</tr>
<tr>
<td>What are the advantages of including the Config.fpw in the project?</td>
<td>467</td>
</tr>
<tr>
<td>How can we include non-VFP objects in the project?</td>
<td>468</td>
</tr>
<tr>
<td>How to reduce screen real estate taken by the Project Manager</td>
<td>470</td>
</tr>
<tr>
<td>How to tear off tabs from the Project Manager</td>
<td>471</td>
</tr>
<tr>
<td>What problems exist when opening a database in the project?</td>
<td>472</td>
</tr>
<tr>
<td>Project dragging and dropping tricks</td>
<td>473</td>
</tr>
<tr>
<td>What happens when dragging from one project to another?</td>
<td>473</td>
</tr>
<tr>
<td>How do I drag objects from a project to a designer?</td>
<td>473</td>
</tr>
<tr>
<td>What happens when dragging from project to program code?</td>
<td>474</td>
</tr>
<tr>
<td>What happens when dragging from Class Browser or Component Gallery to a project?</td>
<td>474</td>
</tr>
<tr>
<td>What happens when dragging from a non-VFP application to a project?</td>
<td>475</td>
</tr>
<tr>
<td>How to take advantage of the project User field</td>
<td>475</td>
</tr>
<tr>
<td>How to go about documenting the project file</td>
<td>476</td>
</tr>
<tr>
<td>Conclusion</td>
<td>477</td>
</tr>
</tbody>
</table>

## Chapter 15: Project Objects and Projecthooks

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>How to use ProjectHooks to catch a big fish</td>
<td>479</td>
</tr>
<tr>
<td>How to set up a global ProjectHook for all projects</td>
<td>481</td>
</tr>
<tr>
<td>What happens when a ProjectHook is lost or deleted?</td>
<td>481</td>
</tr>
<tr>
<td>What did Microsoft leave out of the first release of ProjectHooks?</td>
<td>482</td>
</tr>
<tr>
<td>How to access information in the Project and Files object</td>
<td>483</td>
</tr>
<tr>
<td>How to use Project Objects in development</td>
<td>483</td>
</tr>
<tr>
<td>How to build a basic Application Wizard</td>
<td>484</td>
</tr>
<tr>
<td>ProjectHook and Project Object tricks</td>
<td>488</td>
</tr>
<tr>
<td>How to enhance the base projecthook</td>
<td>488</td>
</tr>
<tr>
<td>How to create a useful development projecthook</td>
<td>489</td>
</tr>
<tr>
<td>How to have the projecthook set the current directory and path</td>
<td>492</td>
</tr>
<tr>
<td>How to programmatically control the VFP IntelliDrop settings</td>
<td>484</td>
</tr>
<tr>
<td>How to remove the printer information from VFP reports</td>
<td>496</td>
</tr>
<tr>
<td>How to track what is done within the Project Manager</td>
<td>499</td>
</tr>
<tr>
<td>How to generate automatic backups of metadata</td>
<td>500</td>
</tr>
</tbody>
</table>
Chapter 16: Creating Reports

Report Rule #1
Formatting techniques
  How to speed printing with printer fonts
  How to generate the "greenbar" effect
  How to generate CheckBoxes in reports
  How to reduce/increase white space when printing memo fields
  How to display a field in preview but not print mode
  How to minimize pain using lines and boxes
  How to use float to your advantage
  How to print bullet symbols with stretchable fields
  How to build a mailing address without gaps
  How to use DBC field captions in reports

Band problems
  How to avoid orphaned headers and widowed footers
  How to have second summary bands with EOF()
  How to create flexible control breaks
  How to build two (or more) sets of detail lines
  How to simulate a detail line longer than one page
  How to fix the location of footer
  How to print "Continued" when detail overflows

Other report techniques
  How to avoid hard coded printer problems
  Using Expression Builder for undefined fields in the DataEnvironment

How to get the label formats to be available

Report Metadata Manipulation
  How do you change fonts programmatically?
  How to convert paper sizes (Letter -> A4 Print)
Conclusion 528

Chapter 17: Managing Reports 529
How to leverage reports and datasessions 529
How to create a report template for a project 530
How to print a range of pages 531
How to print 'Page x of y' on a report 531
How to allow users to select number of copies 532
How to find "Variable not found" errors in a report 533
How to avoid having a report disable the system menu 534
How to collate pages from different reports 534
How to display a custom 'Printing Dialog' 536
How to change the title of Print Preview window 538
How to show a report preview as a Top-Level form 539
How to preview multiple reports at once 540
How to remove printer info. in production reports 542
How to allow end users to modify report layouts 543
How to print a memo field with Rich Text Format 545
How to select the paper tray 547
Other alternatives to the native Report Designer 548
  How to use OLE Automation to Word 548
  How to output to other file types 550
  How to create HTML and ASCII text output 551
  How to generate PDF output 553
How to review code from the report/label designer 555
Conclusion 557
Our Contract with You,
The Reader

In which we, the folks that make up Hentzenwerke Publishing, describe what you, the reader, can expect from this book and from us.

Hi there!

I’ve been writing professionally (in other words, eventually getting a paycheck for my scribbles) since 1974 and writing about software development since 1992. As an author, I’ve worked with a half-dozen different publishers and corresponded with thousands of readers over the years. As a software developer and all-around geek, I’ve also acquired a library of over a hundred computer and software-related books.

Thus, when I donned the publisher’s cap four years ago to produce the 1997 Developer’s Guide, I had some pretty good ideas of what I liked (and didn’t like) from publishers, what readers liked and didn’t like and what I, as a reader, liked and didn’t like.

Now, with our new titles for the spring and summer of 2000, we’re entering our third season. (For those keeping track, the ’97 DevGuide was our first, albeit abbreviated, season and the batch of six "Essentials" for Visual FoxPro 6.0 in 1999 was our second.)

John Wooden, the famed UCLA basketball coach, had posited that teams aren’t consistent – they’re always getting better – or worse. We’d like to get better… One of my goals for this season is to build a closer relationship with you, the reader.

In order to do this, you’ve got to know what you should expect from us.

• You have the right to expect that your order will be processed quickly and correctly and that your book will be delivered to you in new condition.
• You have the right to expect that the content of your book is technically accurate, up to date, that the explanations are clear and that the layout is easy to read and follow without a lot of fluff or nonsense.
• You have the right to expect access to source code, errata, FAQs and other information that’s relevant to the book via our website.
• You have the right to expect an electronic version of your printed book (in compiled HTML Help format) to be available via our website.
• You have the right to expect that, if you report errors to us, your report will be responded to promptly and that the appropriate notice will be included in the errata and/or FAQs for the book.

Naturally, there are some limits that we bump up against. There are humans involved and they make mistakes. A book of 500 pages contains, on average, 150,000 words and several megabytes of source code. It’s not possible to edit and re-edit multiple times to catch every last misspelling and typo, nor is it possible to test the source code on every permutation of development environment and operating system – and still price the book affordably.
Once printed, bindings break, ink gets smears, signatures get missed during binding. On the delivery side, websites go down, packages get lost in the mail.

Nonetheless, we’ll make our best effort to correct these problems – once you let us know about them.

And, thus, in return, when you have a question or run into a problem, we ask that you first consult the errata and/or FAQs for your book on our website. If you don’t find the answer there, please email us at books@hentzenwerke.com with as much information and detail, including (1) the steps to reproduce the problem, (2) what happened and (3) what you expected to happen, together with (4) any other relevant information.

I’d like to stress that we need you to communicate questions and problems clearly. For example…

"Your downloads don’t work" isn’t enough information for us to help you. "I get a 404 error when I click on the Download Source Code link on www.hentzenwerke.com/book/downloads.html." is something we can help you with.

"The code in chapter 14 caused an error" again isn’t enough information. "I performed the following steps to run the source code program DisplayTest.PRG in chapter 14 and receive an error that said "Variable m.liCounter" not found" is something we can help you with.

We’ll do our best to get back to you within a couple of days either with an answer, or at least an acknowledgment that we’ve received your inquiry and that we’re working on it.

On behalf of the authors, technical editors, copy editors, layout artists, graphic artists, indexers and all the other folks who have worked to put this book in your hands, I’d like to thank you for purchasing this book and hope that it will prove to be a valuable addition to your technical library. Please let us know what you think about this book – we’re looking forward to hearing from you.

As Groucho Marx once observed, "Outside of a dog, a book is a man’s best friend. Inside of a dog, it’s too dark to read."

Whil Hentzen
Hentzenwerke Publishing
May, 2000
Acknowledgements

If we were to try and acknowledge, individually, all of those who had contributed, even indirectly, to this book we would have a list of acknowledgements longer than the book itself. But there are some whose contributions have been so significant that we must acknowledge specifically.

First, we'd like to recognize our Technical Editor, John Hosier. Without John the book would never have been in such good shape. Not only did he correct us when we were wrong but his suggestions and improvements were invaluable to us all. The job of Technical Editor is, in many ways, harder than actually writing (and even more thankless) but he has managed it wonderfully - thank you so much, John.

Next, of course, comes our friend and gallant publisher, Whil Hentzen. He was the inspiration behind this book, (though we are still not sure that what he got was what he originally wanted) and his support and assistance has been invaluable. Thanks are also due to all of the team at Hentzenwerke for taking our random scribbings and creating this wonderful book from them. We really appreciate it.

Now we must tackle the most difficult group, the FoxPro Community. We consider ourselves very fortunate to be members, however humble, of this wonderful, multi-national, community. Without you, this book could not have been written and certainly would never have sold a single copy.

The FoxPro Community really IS a community and it sustains itself physically through the many Fox-based User Groups in all parts of the world, electronically through the CompuServe Forums, News Groups, Universal Thread, FoxForum, the Wiki and so on. The comradeship and mutual support is, we believe, unrivalled and long may it continue to be so. Putting faces to names has always been part of the fun of attending DevCon, WhilFest, SoCal, Frankfurt, Amsterdam or any of the many other FoxPro conferences and meetings all over the world. That so many of those “faces” have also become friends is a wonderful bonus and we look forward to renewing old friendships and forging new ones over the years to come.

While it is true that everyone in the community has contributed, in some way, to this book, there are a few individuals whose contributions have been very direct and very specific and we want to take this opportunity to thank them publicly.

- Steven Black (for his "Share" and "MC" utilities)
- Gary DeWitt (for mining the Windows API constants)
- Tamar Granor and Ted Roche (for the indispensable "Hacker's Guide to Visual FoxPro 6.0")
- Doug Hennig (for sharing his work with the Visual FoxPro Builders)
- Christof Lange (for his method of making a FoxPro application "single instance")
- John Petersen (for his contribution of OptUtility)

Last, but by no means least, comes the most important person to us authors, you, our Reader. Thank you for buying the book. We hope that it both pleases you and is useful to you.
Maybe, if we are ever crazy enough to tackle another, you will remember us and give us a look then too.

Andy Kramek
Marcia Akins
Rick Schummer

February 2000
About Us

Andy Kramek

Andy is an independent consultant and long-standing FoxPro developer based, at the time of writing, in England. As well as being a Microsoft Most Valuable Professional, he is also a Microsoft Certified Professional for Visual FoxPro in both Desktop and Distributed applications. Andy is a long time member of the FoxPro support forums on CompuServe, where he is also a SysOp.

Andy's published work includes "The Revolutionary Guide to Visual FoxPro OOP" (Wrox Press, 1996) and, together with his friend and colleague Paul Maskens, the monthly "Kitbox" column in FoxTalk (Pinnacle Publications). Andy has spoken at conferences and user group meetings in England, mainland Europe and the USA.

In the little spare time that he has, Andy enjoys playing squash and golf (though not necessarily at the same time), traveling and listening to Marcia.
You can reach Andy at: AndyKr@Compuserve.com

Marcia Akins

Marcia is an experienced developer in several languages who has been working mainly in Visual FoxPro for the past eight years. She is an independent consultant and, at the time of writing, had deserted her native Ohio to live and work (with Andy) for a year or so in England. She is a Microsoft Most Valuable Professional, and holds Microsoft Certified Professional qualifications for both Visual FoxPro Desktop and Distributed applications.

She has several articles in FoxPro Advisor to her credit and is widely, and at least half-seriously, known as the "Queen 'o' the Grids". She has spoken at conferences and user group meetings in the USA, England and mainland Europe and is a frequent contributor to CompuServe, the Universal Thread and FoxForum.com.

When she is not busy developing software, Marcia enjoys golfing, skiing, playing tennis, working out at the gym, traveling, and harassing Andy.
You can reach Marcia at: MarciaGAkins@Compuserve.com

Rick Schummer

Rick is the Director of Development for Kirtland Associates, Inc. in Troy MI, USA. Kirtland Associates writes custom database applications for a rapidly expanding customer base. He not only directs the development of this fun organization, but also participates in the education of new and experienced Visual FoxPro developers. It is a great way to further your own skills. After hours he enjoys writing developer tools that improve his team's productivity and occasionally pens articles for FoxTalk, FoxPro Advisor, and several user group newsletters.
Rick recently became a Microsoft Certified Professional by passing both the VFP Desktop and Distributed exams.
He spends his free time with his family, cheers the kids as they play soccer, has a volunteer role with the Boy Scouts, and loves spending time camping, cycling, coin collecting, photographing and reading. Rick is a founding member and secretary of both the Detroit Area Fox User Group (DAFUG – http://www.dafug.org) and Sterling Heights Computer Club (http://member.apcug.org/shcc).
You can reach Rick at:
ruschummer@compuserve.com
ras@kirtlandsys.com
http://my.voyager.net/rschummer

John Hosier

John has been active in the FoxPro community since 1987 and has been a developer, consultant, author, conference speaker and trainer. John was also a founding board member of the Mid-Atlantic FoxPro User Group and has served as its president and treasurer. As a consultant, John has worked with both large and small clients in Eastern and Western Europe, the Middle East, the Caribbean and all over the United States. John’s publishing credits include FoxPro Advisor, FoxTalk, FoxPro User’s Journal and a German magazine called “Data Base: Das Fachmagazin für Datenbankentwickler.” No, John does not speak German, but he thinks it is pretty funny that he wrote an article that he was unable read in the final publication. As a Microsoft Certified Professional in Visual FoxPro, John has worked on a wide variety of projects including client server, internet/intranet (including an XML parser written in VFP) and distributed applications. John currently makes his home in the Chicago area.
You can reach John at:
JHosier@earthlink.net
How to Download the Files

There are two sets of files that accompany this book. The first is the source code referenced throughout the text, and noted by the spider web icon; the second is the e-book version of this book – the compiled HTML Help (.CHM) file. Here’s how to get them.

Both the source code and the CHM file are available for download from the Hentzenwerke website. In order to do so, following these instructions:


2. Look for the link that says "Download Source Code & .CHM Files." (The text for this link may change over time – if it does, look for a link that references Books or Downloads.)

3. A page describing the download process will appear. This page has two sections.

Section 1: If you were issued a username/password from Hentzenwerke Publishing, you can enter them into this page.

Section 2: If you did not receive a username/password from Hentzenwerke Publishing, don’t worry! Just enter your email alias and look for the question about your book. Note that you’ll need your book when you answer the question.

4. A page that lists the hyperlinks for the appropriate downloads will appear.

Note that the .CHM file is covered by the same copyright laws as the printed book. Reproduction and/or distribution of the .CHM file is against the law.

If you have questions or problems, the fastest way to get a response is to email us at books@hentzenwerke.com.
Introduction

Thank you for buying this book. We hope that you have as much fun reading it as we did in writing it and that you manage to find as many useful things as we did when compiling it. The objectives behind this book underwent considerable change between its inception (at the Dutch User Group "Conference to the Max" held in Arnhem, Holland in May 1999) and the version of the text which you now hold in your hands. However, the paragraphs below describe what we finally hoped to achieve with the book.

What is this book?
First, it must be stated that this is not a book that will teach you how to use Visual FoxPro. Our primary objective has been to try and distill some of the (often painful) experiences which we, and many others, have accumulated over the years so that you can avoid falling into the same traps that we did and maybe even find some alternative ways of doing things. This is not to say that there is always a 'best' or even a 'right' way of doing things in FoxPro. The language is so rich and powerful that there are usually several ways of tackling any given problem, however, there are also many traps for the unwary, and many techniques that have proven useful. The problem which we have tried to tackle is to collect such tricks and traps together, to group them into some logical order and to try and provide the one thing that almost every developer we know has been asking for – concise and 'relevant' example code.

A word about the code in this book
The code samples in this book have been consciously written to make them easy to follow – at times this has meant that we have forgone some obvious optimizations. Thus you will find many places where you might say 'Why didn't they do it like this? It would have saved a dozen lines of code!' Please bear with us, and remember that not everyone is as perceptive as you are.

You will also note that, for similar reasons, we have not repeated, in every code snippet, method or function, the "standard" tests and error handling code that you would normally expect to find (like checking the type of parameters passed to a function). We have assumed that you know how to do this and, if you want to use the code from this book, will add it yourself where necessary.

So who is this book for?
As we have already said, this book will not teach you how to use Visual FoxPro – it assumes you have a reasonable degree of comfort with the basic operation of the VFP Database and Command Language and with the basic principles of Object Oriented Programming. We would expect that you will have read and used such excellent and useful references as Whil Hentzen's 'Programming VFP,' 'The Revolutionary Guide to VFP OOP' by Will Phelps, Andy Kramek and Bob Grommes and, of course, the indispensable 'Hacker's Guide to VFP,' by Tamar Granor and Ted Roche.

If you are looking for alternative ways of tackling problems, code improvement hints, workarounds for common traps and 'war stories' of those who have been there and done it (yes, we even have the tee-shirts), then this book is for you.
What is in this book?
This book includes tried and tested solutions to common problems in Visual FoxPro together with some basic techniques for building Visual FoxPro tools and components. The book is organized into chapters that attempt to group subjects under logical headings. Each chapter consists, essentially, of a series of 'How Do I …?' questions. Each question includes a working example, and each chapter's example code may be downloaded individually.

All example code was written and tested using Visual FoxPro Version 6.0 (with Service Pack 3). While much of it should run in any version of Visual FoxPro, there are obviously some things that are version specific. (Each new version of Visual FoxPro has introduced some entirely new commands and functions to the language.)

What is not in this book?
An awful lot! In order to keep this book to a manageable size we have left out a lot of things. Since this is essentially a 'How To' book for Visual FoxPro, we have not even attempted to cover such topics as building COM components, or Internet web pages (there are excellent books on these subjects available). Nor have we covered ActiveX controls or Automation (another book would be needed for this topic alone). We recognise that there are significant omissions but felt that since we could not possibly cover everything, we should concentrate on the 'pure' Visual FoxPro issues – and we make no apology for doing so.

Where do you start?
The short answer is wherever you want to! While it has been one of our main concerns to make this a "readable" book, we recognize that you are probably looking at this book because you have a specific problem (or maybe more than one) to deal with and are looking for inspiration if not an actual solution. We cannot hope to provide "solutions" for everyone but if we can offer you a little inspiration, backed up with sample code to get you started, then we will have succeeded in our aims – and you can relax in the knowledge that your modest expenditure on this tome has already proven a worthwhile investment.
Chapter 1
Controlling the VFP Environment

"To begin at the beginning" (Narrator, "Under Milk Wood" by Dylan Thomas)

One of the major benefits of developing in Visual FoxPro is that you have almost complete control over the environment in which your code will run. However, like many benefits this can be a double-edged sword and there are many things to be aware of when establishing and controlling both your development and production environments. In this chapter we will cover some of the techniques we have found to work well.

Starting Visual FoxPro
Visual FoxPro, like most applications, supports several 'command line switches'. Most of the time these tend to get forgotten, but they do exist and are all documented in the online Help files in the 'Customizing Visual FoxPro Startup Options' topic. Probably the most useful ones to remember are:

- \-C which specifies the configuration file to use
- \-T which suppresses the VFP sign-on screen
- \-R which refreshes the VFP registry settings (Note, the settings that get refreshed are those relating to information about VFP, such as file associations. The \-R switch does not update the settings controlled through Visual FoxPro's Options Dialog. This is only done when 'Set As Default' is used to exit the dialog).

So to start Visual FoxPro without the sign-on screen display and with a refresh of the registry settings the required command line would be:

G:\VFP60\VFP6.EXE \-R \-T

Configuration files
There are several ways of handling the initialization of Visual FoxPro, but the easiest, and most flexible, is still to use a configuration file. Visual FoxPro uses a simple formatted text file, called "CONFIG.FPW" by default, as the source for a number of environmental values which can be set as the system starts up.
How to specify a config.fpw file
The actual file name does not matter as you can specify the configuration file which Visual FoxPro is to use as a command line parameter by using the '-c' switch in the command line which is used to start Visual FoxPro. So to set up your own configuration file (for example for a specific application) use the following command line:

G:\VFP60\APPS\MYAPP.EXE -cG:\VFP60\myconfig.txt

How VFP locates its configuration file
The default behavior of Visual FoxPro, in the absence of a specific configuration file, is to search the following locations for a file named 'config.fpw' in this order:

- The current working directory
- The directory from which Visual FoxPro is being started
- All directories in the DOS path

If you are using the '-c' switch to specify a file named other than the default, or in a specific location, you must include the fully qualified path and file name. This provides a simple method of handling the initialization of different applications installed on the same machine.

How VFP starts up when no configuration file is found
If no configuration file is found or specified, then Visual FoxPro will be started with only those settings that are specified in the Options Dialog (located on the TOOLS pad of the main Visual FoxPro menu).

Why these settings in particular?
The answer is simply that all of the settings from this dialog are actually stored in the Windows Registry and can be found under the Registry Key:

HKEY_CURRENT_USER\Software\Microsoft\VisualFoxPro\6.0\Options

Including a configuration file in the project
One little "trap" to watch out for – if you add a configuration file named 'config.fpw' to your project as a text file, it will be INCLUDED in the project by default. When you build an .exe from the project, the config.fpw file will be built into the resulting file. Since Visual FoxPro looks for a file named 'config.fpw' during startup, it will always find the built-in version first and will not look any further. This would apply even if you were to explicitly specify a different configuration file using the '-C' switch! Your specified file would be ignored, and the built-in configuration file would be executed. The best solution is NOT to add your configuration file to the project at all, but if you do, to make sure that it is marked as 'excluded' from the build.
How to suppress a configuration file
Starting Visual FoxPro with the command line parameter '-c' alone suppresses the default behavior and prevents any configuration file that may be found from being run. The result is that you can force Visual FoxPro to start up with its default settings only.

How to determine which configuration file is being used
One of the commonest problems with configuration files is failing to ensure that Visual FoxPro is reading the correct CONFIG.FPW. As noted above, if Visual FoxPro can't find a configuration file, it will search the DOS path and simply use the first one it finds. This could be anywhere on a network. The `SYS(2019)` function will return the full path and file name of the configuration file that Visual FoxPro actually used. If no configuration file was found, the function merely returns an empty string.

What goes into the configuration file?
Now that we know something about how the configuration file is used, the next question is what can we put into it? The answer is quite a lot! Essentially there are three categories of things that can be specified in the configuration file as follows:

Special settings
There are a number of settings that can ONLY be made in a configuration file. (For full details see the "Special Terms for Configuration Files" topic in the Visual FoxPro online Help and the entry under "Configuring Visual FoxPro" in the online documentation.) Notice that the ability to set the location for temporary files is also available in the Options Dialog. Specifying the TMPFILES location in the configuration file will override any setting that is made there and can be useful when you need to differentiate between development and run time locations for temporary files.

<table>
<thead>
<tr>
<th>KeyWord</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVCOUNT = nn</td>
<td>Sets the maximum number of variables that Visual FoxPro can maintain. This value can range from 128 to 65,000; default is 1024.</td>
</tr>
<tr>
<td>TMPFILES = drive:</td>
<td>Specifies where temporary EDITWORK, SORTWORK, and PROGWORK work files are stored if they have not been specified with any of the other options. Because work files can become very large, specify a location with plenty of free space. For faster performance, especially in a multiuser environment, specify a fast disk (such as a local disk). Default is the startup directory.</td>
</tr>
<tr>
<td>OUTSHOW = OFF</td>
<td>Disables the ability to hide all windows in front of the current output by pressing SHIFT+CTRL+ALT. Default is ON.</td>
</tr>
</tbody>
</table>

One other setting that can be used in the configuration file only, but which is not included in the Help file list is "SCREEN = OFF". This prevents the Visual FoxPro main screen from being displayed when your application starts and prevents the annoying 'flash' of the VFP screen that still occurs even if your startup program turns off the main screen with the
command _Screen.Visible = .F. (which enables you to present your application's initial form, or a "splash" screen, without displaying the main VFP window first).

**SET Commands**

Virtually all of the standard SET commands can be issued in a configuration file. The only thing to watch out for is that some settings are actually scoped to the datasession. (See the "Set DataSession" topic in the online Help for a full listing.) So, there is little point in specifying them in the configuration file if you plan to use Private DataSessions for your forms. The syntax for specifying SET commands in the configuration file is a simple assignment in which the keyword 'SET' is omitted:

```plaintext
DEFAULT = C:\VFP60\TIPSBOOK
DATE = BRITISH
```

**Commands**

Well actually you can only specify one (count 'em!) command, and it must be the last line of the configuration file. Like other configuration file entries, it is entered as a simple assignment:

```plaintext
COMMAND = DO setupfox
```

What is the use of just ONE command? Well, quite a lot really because that one command can call a FoxPro program, and that can do a lot of things!

One of the main limitations of the configuration file is that you cannot actually set things that are internal to Visual FoxPro (e.g. system variables) because, when the configuration file runs, Visual FoxPro hasn't actually started. Using this setting allows you to specify a program file to run immediately after Visual FoxPro has started up – even before the command window is displayed.

This program can then be used to set up your development environment the way that you really want it. For example, here are some of the things that are in our standard setup file:

```plaintext
*** Standard 'SET' options (These could be entered directly into the Config File)
SET TALK OFF
SET BELL OFF
SET SAFETY OFF
SET STATUS OFF
SET STATUS BAR ON
SET DATE BRITISH
SET CENTURY ON

*** Re-define Function Keys
SET FUNCTION 2 TO "CLOSE TABLES ALL;CLEAR WINDOWS;"
SET FUNCTION 3 TO "CANCEL;SET SYSMENU TO DEFA;ACTIVATE WINDOW COMMAND;"
SET FUNCTION 4 TO "CLEAR ALL;SET CLASSLIB TO;SET PROC TO;"
SET FUNCTION 5 TO "DISP STRU;"
SET FUNCTION 6 TO "DISP STAT;"
SET FUNCTION 7 TO "DISP MEMO LIKE *"
SET FUNCTION 8 TO "CLEAR;CLEAR WINDOWS;"
SET FUNCTION 9 TO "MODI FORM "
```
SET FUNCTION 10 TO "MODI COMM "
SET FUNCTION 11 TO "DO setpath WITH "
SET FUNCTION 12 TO "="CHGDEFA(); "
***Set up the Screen properties
_SCREEN.CAPTION = "VFP 6.0 (Development Mode)"
_SCREEN.CLOSABLE = .F.
_SCREEN.FONTNAME = "Arial"
_SCREEN.FONTSIZE = 10
_SCREEN.FONTBOLD = .F.
*** Run Cobb Editor Extensions
DO G:\VFP60\CEE6\CEE6.APP
*** Set up some On Key Labels
ON KEY LABEL CTRL+F10 suspend
ON KEY LABEL CTRL+F11 o=SYS(1270)
ON KEY LABEL CTRL+F12 RELEASE o
*** Set up any system variables required
#include = HOME() + "Foxpro.h"
_throttle = 0.1
*** Set up any standard "Public" variables
PUBLIC gcUserName, gcAppPath, gcDataPath
STORE "" TO gcUserName, gcAppPath, gcDataPath
*** Run Standard Path Set-up
DO setpath WITH 1

As you can see, apart from setting up the basic VFP system environment and handling our own particular requirements, this 'one' command available in the configuration file has now been used to run a couple of other programs (CEE and our own SetPath procedure) so we get three for the price of one. By putting these settings in a program, we also have a simple way of re-initialising the environment by re-running this program at any time.

**Giving VFP a path**

Visual FoxPro has the ability to use its own search path and, as a general rule, you should always specify a path for Visual FoxPro for both development and production environments - although they may be very different (see above for one way of handling this requirement). Setting a path for Visual FoxPro does not change the normal DOS search path but can significantly speed up your application by limiting the places that Visual FoxPro has to search in order to find its files – especially in a network environment.

**How VFP looks for files**

By default Visual FoxPro uses the current directory as its 'path' and you can always restore this setting by simply issuing: SET PATH TO

However for more sophisticated applications, and in development, you will normally have some sort of directory structure and you should always set a proper search path to include all required directories.

**Setting the default directory**

Normally you still will want to set a default (or "working") directory – this is where Visual FoxPro will look first for any file that it requires. This can be done in a number of ways, depending on your requirements:
• Specify a default in the Configuration File using DEFAULT = <path to directory>
• Set the default directly in code using SET DEFAULT TO <path to directory>
• Change to a directory using the CD <path to directory> and issue SET PATH TO

Note that using the Get, Put or Locate functions (e.g. GetDir()) does not change either the
default directory or the path. To change the default directory interactively use: SET DEFAULT
TO (GetDir()). (The 'CD' (or 'CHDIR') command can also be used to change both drive and
directory to the specified location).

Using the SET PATH command
Setting the path is simplicity itself. Just issue the SET PATH command followed by a list of
the directories that you wish to include. You do not need to fully qualify subdirectories –
separating them with either commas or semi-colons is sufficient. The example shows a typical
Visual FoxPro search path:

```
SET PATH TO G:\VFP60;C:\VFP60\TIPSBOOK\;DATA;FORMS;LIBS;PROGS;UTILS;
```

To retrieve the current path setting you can use the SET() function (which will work with
most of the Visual FoxPro SET commands) as shown below. You can assign the result directly
to a variable or, as shown below, directly to the clipboard so that you can then paste the current
path into a program or documentation file:

```
_clipText = SET( 'PATH' )
```

Visual FoxPro allows the use of both UNC path names, like:

```
\SERVERNAME\DIRECTORYNAME\.
```

and allows the use of embedded spaces (when enclosed in quotation marks) in directory
names like:

```
"..\COMMON DIRECTORY"
```

However, while the latter may be allowed, we tend to subscribe to the principle that
'although you can use embedded spaces, arsenic is quicker.' (The same, by the way, applies to
file names with spaces!) While improving readability, spaces can also cause problems when
trying to handle files and directories programmatically and we still feel that the best advice is
to avoid them wherever possible in applications. For example, the following code works
perfectly well for conventional directory names, but will fail if the selected directory contains
embedded spaces:

```
LOCAL lcDir
lcDir = GETDIR()
IF ! EMPTY(lcDir)
   SET DEFAULT TO &lcDir
ENDIF
```
Where am I?
Fortunately, Visual FoxPro provides us with a number of functions that will help us locate where we are at any time:

- **SYS(2004)** returns the directory from which Visual FoxPro was started but in a distributed run time application, this will always be the location of the `VFP6R.DLL` (which is normally the appropriate version of the Windows 'System' directory).
- **HOME()** returns the directory from which Visual FoxPro was started by default, but has a number of additional useful options in VFP6 which return information about Visual Studio Components.
- **_VFP.FULLNAME** accesses a property of the Visual FoxPro application object that contains the full path and file name which was used to start VFP.
- **FULLPATH(' ') or FULLPATH( CURDIR() )** returns the full drive and directory of the current working directory (including the terminal ")
- **SYS(5)** returns the default drive (including the ':')
- **CD** shows the current drive and directory in the current output window – but will also change drive and directory in one single command.
- **CHDIR** will change to the specified Drive/Directory (just like CD) but will not report the current status (and so does not mess up your forms).
- **CURDIR()** returns just the current directory (with terminal ") but not the drive.

How to set a path programmatically
As an alternative to hard coding the paths in your setup file, it is possible to derive the path (assuming that you use standard directory structures) using the native Visual FoxPro functions. The little function below shows how this might be done to account for both development and run time structures. It uses the **PROGRAM()** function to determine how it was called and returns a different path when called from a Form or a Program than when called from a compiled file:

```
FUNCTION CalcPath()
LOCAL lcSys16, lcProgram, lcPath, lcOldDir
*** Get the name of program that called this one.
lcSys16 = SYS(16, 1)
*** Get current working directory
lcOldDir = (SYS(5)+CURDIR())
*** Make the directory from which it was run current
lcProgram = SUBSTR(lcSys16, AT(":", lcSys16) - 1)
CD LEFT(lcProgram, RAT(\", lcProgram))
IF INLIST( JUSTEXT( lcProgram ), "FXP", "SCX" )
```

```
*** If we are running a PRG/Form directly, then find the parent directory
CD ..
*** Set up path to include VFP Home plus the standard DEV directory tree
lcPath = (HOME()+';'+SYS(5)+CURDIR()+";DATA;FORMS;LIBS;PROGS;UTILS")
ELSE
*** We are using an EXE/App! Adjust path for DISTRIBUTION directory tree
lcPath = (HOME()+';'+SYS(5)+CURDIR()+";DATA")
ENDIF
*** Restore original directory
CD (lcOldDir)
*** Return the calculated path
RETURN lcPath
ENDFUNC

Making sure VFP is only started once
So far so good! We have managed to cover the process of starting VFP and setting up the basic environments for both development and production. At this point one of the things we all come across is the absent-minded user who minimizes an application and then, ten minutes later, starts a fresh copy from their desktop. Within the hour they have six copies of the application running and are complaining that their machine is slowing down. What to do? (Apart from shooting the user which, appealing though the idea may be, is generally frowned upon and, depending on their seniority, may also be a career-limiting move.) There are actually several approaches that can be taken, as described below.

Using a 'semaphore' file
This is probably the simplest approach of all. It relies on your application creating, on first launch, a zero-byte file whose sole purpose is to indicate that the application is running. Any time the application is launched, it looks for this file and, if it finds it, simply shuts down again:

*******************************************************************************
* Program....: ChkSFile.prg
* Compiler....: Visual FoxPro 06.00.8492.00 for Windows
* Abstract....: Checks for a Semaphore File, creates one if not found and
* ...........: returns a flag
*******************************************************************************
FUNCTION ChkSFile( tcAppName )
LOCAL lcAppName, lcFile, lnHnd, llRetVal
*** Default lcAppName if nothing passed
lcAppName = IIF( EMPTY(tcAppName) OR VARTYPE( tcAppName ) # "C", ;
'apprun', LOWER( ALLTRIM( tcAppName ) ) )

*** Force a TXT extension for the semaphore file in current directory
lcFile = (SYS(5) + CURDIR() + FORCExT( lcAppName, 'txt' ) )

*** Now check for the file?
IF ! FILE( lcFile )
*** File not found, so create it
lnHnd = FCREATE( lcFile )
IF lnHnd < 0
*** Cannot create the file, some sort of error! Set Return Flag
llRetVal = .T.
ELSE
*** Close the file
FCLOSE( lnHnd )
ENDIF
ELSE
*** Set Return Flag
llRetVal = .T.
ENDIF
*** Return Status Flag
RETURN llRetVal

The function returns a logical value that can be used to determine whether to allow the current application to continue, as the following snippet illustrates:

*** Check for a second instance of the application
IF ChkSFile()
    QUIT
ENDIF

Of course the catch with this approach is that your application must delete the file as part of its close-down routine (unless you really want a once-only application). That raises the question of what can you do if, heaven forbid, the application terminates abnormally (euphemism for 'crash') or the user makes an improper exit (disconnecting the power supply for example!). The answer is "Not much." Typically this will require the intervention of 'System Support' to physically delete the file.

But it is a nice, easy approach with no other real drawbacks providing that the semaphore file is always created on the end-user's machine, or in a specific user directory.

**Using the Windows API**
The function below makes use of three Windows API functions to look for a window that is named the same as the application (FindWindow), to make an existing window the uppermost window (BringWindowToTop) and to maximize it (ShowWindow):

*******************************************************************************
* Program....: OnceOnly.prg
* Compiler...: Visual FoxPro 06.00.8492.00 for Windows
* Abstract....: Checks for an existing instance of the application window
* ...........: and, if found, activates the original and returns a flag.
*******************************************************************************
FUNCTION OnceOnly
LOCAL lnHWND, lcTitle, llRetVal

*** Set UP API Calls
Declare Integer FindWindow IN Win32Api AS FindApp String, String
Declare BringWindowToTop IN Win32API AS MakeTop Integer
Declare ShowWindow IN Win32Api AS ShowWin Integer, Integer

*** Get the current Screen Caption
lcTitle = _Screen.Caption
*** Change it to avoid finding the current instance
_Screen.Caption = SYS(3)

*** Now locate another instance
lnHWND = FindApp( NULL, lcTitle )

*** And restore the original caption
_Screen.Caption = lcTitle

*** Check the results
IF lnHWND > 0
*** We have found something!
*** So make it uppermost and maximize it (ShowWin => 3)
MakeTop( lnHWND )
ShowWin( lnHWND, 3 )
*** Set the Return Value
llRetVal = .T.
ENDIF

*** Return Status for action
RETURN llRetVal

The function returns a logical value that can be used to determine whether to allow the
current application to continue, as the following snippet illustrates:

*** Check for a second instance of the application
IF OnceOnly()
QUIT
ENDIF

There is one major drawback to watch out for here. The API functions used check for the
name of a window – in this case we are using _Screen.Caption. If your application changes
the Caption of the screen at run time (as many do), this approach will not work.

Combination of semaphore and Windows API
The last example here shows how combining the principles of the other two examples gives a
better all-round approach:

**************************************************************************
* Program....: IsAppRun.prg
* Compiler...: Visual FoxPro 06.00.8492.00 for Windows
* Abstract....: Checks for a window which is created with a Unique ID by and
* ............: in the application. Combination of Semaphore and API.
* ............: Based on code originally posted into the Public Domain
* ............: by Christof Lange
**************************************************************************
FUNCTION IsAppRun( tcUniqueID )
LOCAL llRetVal, lcUniqueID

*** MUST pass an application ID to this function!
IF EMPTY(tcUniqueID) OR VARTYPE( tcUniqueID ) # "C"
   MESSAGEBOX( 'An Application Specific Character ID is mandatory' + CHR(13) ;
      + 'when calling the IsAppRun() function', 16, 'Developer Error' )
RETURN .T.
ELSE
*** Strip out any spaces
    lcUniqueID = STRTRAN( tcUniqueID, " " )
ENDIF
*** First check for the existence of the Semaphore window
IF WEXIST("_Semaphore_")
    RETURN .T.
ENDIF
*** Look for an occurrence of this ID as a Window Name
DECLARE INTEGER FindWindow IN Win32Api AS FindApp String, String
IF FindApp( NULL, lcUniqueID ) > 0
    *** We found one!  Set Return Value
    llRetVal = .T.
ELSE
*** Create a new window with this ID
    DEFINE WINDOW _Semaphore_ IN DESKTOP FROM 1,1 TO 2,2 TITLE lcUniqueID
ENDIF
*** Return Status Flag
RETURN llRetVal

To use this function, it is simplest to include a #DEFINE in your standard startup file so that you can specify a new unique ID for each application:

#DEFINE APPID "App0001-99"
IF IsAppRun( APPID )
    QUIT
ENDIF

This very neat solution avoids both the problem of 'dangling files' in the semaphore method and that of changing the caption in the API method because the window can only be created and maintained within an instance of the application. As soon as it terminates in any way – even as a result of a crash - the window is destroyed and there is nothing to clean up. Since the window gets its name explicitly from the application, it does not rely on the caption being constant either. Cool stuff Christof, thank you!

SET Commands and DataSessions
OK – we've got Visual FoxPro up and running (and made sure that we can only start one instance of our application) so now what? There are, in case you hadn't noticed, an awful lot of SET commands in Visual FoxPro that allow you to configure the environment in detail. Many of these affect the environment globally, but some are scoped to the currently active datasession (see the "Set DataSession" topic in the on-line help for a full listing of these). When you start Visual FoxPro you are always in the DEFAULT datasession.

What exactly does "Default DataSession" mean?
The Oxford English Dictionary (Ninth Edition) offers a definition of "default" as:

"A pre-selected option adopted by a computer program when no alternative is specified by the user or programmer"

Unfortunately Visual FoxPro seems to prefer to define the word according to the rules proffered by Humpty-Dumpty in Lewis Carroll's 'Alice Through the Looking Glass':
"When I use a word," Humpty Dumpty said in rather a scornful tone, "it means just what I
choose it to mean – neither more nor less."

In fact the Default DataSession is actually DataSession #1 – neither more nor less. It has
no special significance other than that when you start Visual FoxPro, it is selected (just as
Work Area #1 is always selected as the first available work area in any DataSession). This is
easily demonstrated using the SET DATASESSION window and the command window.

When you open the DataSession window, it will display the name of the current
DataSession as 'Default(1)', however the command:

```
SET DATASESSION TO DEFAULT
```

results in a 'Variable Default Not Found' error, while

```
SET DATASESSION TO 1
```

is accepted without comment.

However, when you run a form whose DataSession property is set to "1 Default Data
Session", VFP interprets the term "default" as meaning 'CURRENT' – in other words, a Form
which is designed using this setting will use whichever data session is active when the form is
initialized. This does not appear very logical at first sight since one might reasonably expect
that because Data Session #1 is named 'Default', setting a form's DataSession property to '1
Default Data Session' would ensure that the form would actually use that Data Session and no
other. Not so!

The actual behavior makes sense when you wish forms to share a data session. By setting
the child form's DataSession property to 1 (Default Data Session), it will use whatever
DataSession its parent form was using – whether that DataSession is Private or not.

**So can I have a "public" Datosession?**

The short answer is NO! Visual FoxPro does support the concept of a truly 'default' (or 'public')
Datosession. In other words, if a specified table is not found in the current Data Session VFP
will not look for it elsewhere. All Datosessions are effectively "Private" – even Datosession
#1.

**How can I ensure SET commands apply to a private data session?**

This is actually a complex question and the answer, as so often in VFP, is 'it depends.'
Normally you will be using datosessions that are being created by a form (or formset) as
'Private'. It also matters whether you are using the Form's native DataEnvironment or not. In
any case it is important to understand the order in which things happen – the sequence below
shows how a form with a Private Datosession, and a table in its native DE, is initialized:

```
METHOD: DATAENVIRONMENT.OPENTABLES ()
DATASESSION: 2
ALIAS(): <None>

METHOD: DATAENVIRONMENT.BEFOREOPENTABLES ()
DATASESSION: 2
```