

Developer's Guide

Microsoft®
FOXPRO®
Relational Database Management System for MS-DOS®

Information in this document is subject to change without notice. Companies, names, and data used in examples herein are fictitious unless otherwise noted. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Microsoft Corporation.

©1989–1993 Microsoft Corporation. All rights reserved. Printed in the United States of America.

The Fox Head logo, FoxBASE+, FoxPro, Microsoft, MS, MS-DOS, and Multiplan are registered trademarks and Rushmore is a trademark of Microsoft Corporation in the United States of America and other countries.

Paradox is a registered trademark of Ansa Software, a Borland Company.

Macintosh is a registered trademark of Apple Computer, Inc.

dBASE III PLUS and dBASE IV are registered trademarks, and Framework II is a trademark of Ashton-Tate Corporation.

Hayes is a registered trademark of Hayes Microcomputer Products, Inc.

HP is a registered trademark of Hewlett-Packard Company.

Intel is a registered trademark of Intel Corporation.

IBM is a registered trademark of International Business Machines Corporation.

Lotus, Symphony, and 1-2-3 are registered trademarks of Lotus Development Corporation.

Novell is a registered trademark of Novell, Inc.

Epson is a registered trademark of Seiko Epson Corporation, Inc.

Contents

Overview: Putting It All Together

This chapter provides an overview of the power tools used to develop an application. The chapter begins on page 1-1.

Screens

| | |
|---|------|
| Advantages of the Screen Builder | 2-2 |
| Terms Used in this Chapter | 2-3 |
| Utility Screens | 2-4 |
| Other Screens | 2-5 |
| Screen Sets | 2-6 |
| Code Snippets | 2-7 |
| Calling a Screen Program | 2-10 |
| Significance of READ | 2-11 |
| Your Working Environment | 2-14 |
| Design Considerations | 2-15 |
| The Generated Program | 2-18 |
| Screen Layout | 2-26 |
| Setup Code | 2-27 |
| Cleanup and Procedure Code | 2-37 |
| Window Definitions | 2-43 |
| READ Level Clauses | 2-44 |
| Field Objects and Controls | 2-54 |
| Field Objects | 2-55 |
| Push Buttons | 2-65 |
| Radio Buttons | 2-71 |
| Check Boxes | 2-75 |
| Popups | 2-78 |
| Lists | 2-84 |
| Coordinating Browse with Screens | 2-89 |
| Activating Browse Windows | 2-90 |
| Sizing and Positioning Browse Windows | 2-91 |
| Activating Menus During a Modal READ | 2-92 |
| Debugging Screen Code in an Application | 2-93 |
| Using FoxDoc with Screen Programs | 2-95 |

Menus

| | |
|--|------|
| Advantages of the Menu Builder | 3-2 |
| Terms Used in this Chapter | 3-3 |
| Code Snippets | 3-5 |
| Calling a Menu Program | 3-7 |
| Activating the Menu | 3-8 |
| READ and Menus | 3-8 |
| SET SYSMENU | 3-9 |
| PUSH MENU and POP MENU | 3-10 |
| Your Working Environment | 3-11 |
| Design Considerations | 3-12 |
| The Generated Program | 3-16 |
| General Options... | 3-20 |
| Menu Bar Options... | 3-28 |
| Menu Popup Options... | 3-31 |
| Option Check Box | 3-36 |
| Debugging your Menus | 3-40 |
| Additional Tips | 3-42 |

Coordinating Screens and Menus

| | |
|--|-----|
| Managing a Menu System | 4-2 |
| Accessing Menus During a READ | 4-2 |
| Controlling Menus with SET SYSMENU | 4-3 |
| Saving and Restoring Menus | 4-3 |
| Calling Screen and Menu Programs | 4-4 |
| Accessing Screen Controls via a Menu | 4-6 |

Project — The Main Organizing Tool

| | |
|--|-----|
| Advantages of a Project | 5-2 |
| What Can Projects Contain? | 5-3 |
| One Project Versus Multiple Projects | 5-4 |
| Home Directory for Portable Applications | 5-5 |
| Selecting a Main File | 5-7 |
| Including Modifiable Files in Applications | 5-8 |
| Unknown References in Projects | 5-9 |

| | |
|--|------|
| Procedural Code in Projects | 5-11 |
| Error Handling | 5-12 |
| Saving the Current Environment | 5-13 |
| Creating the New Environment | 5-15 |
| Preserving/Restoring the System Menu Bar | 5-15 |
| Testing For Resources | 5-16 |
| Utility Procedures | 5-16 |

Debugging Your Application

| | |
|--|-----|
| Program Errors | 6-2 |
| Compilation Errors | 6-3 |
| Interactive Compilation | 6-3 |
| COMPILE Command | 6-3 |
| Save and Compile | 6-4 |
| Causes of Compilation Errors | 6-4 |
| Runtime Errors | 6-5 |
| Debugging Suggestions | 6-6 |

Using SQL SELECT

| | |
|---------------------------|-----|
| Query Databases | 7-2 |
| Problems | 7-3 |
| Solutions | 7-7 |

Report Variable Hints

| | |
|---|-----|
| Report Variable Do's and Don'ts | 8-2 |
|---|-----|

Arrays

| | |
|---|------|
| Creating Arrays | 9-2 |
| FoxPro Array Functions | 9-4 |
| Manipulating Arrays | 9-5 |
| Initializing Entire Arrays | 9-5 |
| Referencing Array Elements | 9-5 |
| Assigning Values to Array Elements | 9-6 |
| Redimensioning Arrays | 9-7 |
| Public and Private Arrays | 9-8 |
| Public Arrays | 9-8 |
| Private Arrays | 9-8 |
| Array Limitations | 9-9 |
| Passing Entire Arrays to User-Defined Functions | 9-10 |
| Transferring Data Between Arrays and Databases | 9-11 |
| Arrays and SQL SELECT | 9-13 |
| Arrays and FoxPro Controls | 9-14 |

Low-Level File Input/Output

| | |
|--|-------|
| Creating Files | 10-3 |
| Opening Files and Ports | 10-5 |
| Reading From Files and Ports | 10-6 |
| Writing to Files and Ports | 10-8 |
| Closing Files and Ports | 10-8 |
| Commands and Functions for Low-Level I/O | 10-9 |
| Low-Level Access to Communications Ports | 10-10 |

Text Merge

| | |
|---|-------|
| Merging Text with Text Merge Components | 11-2 |
| \ \ \ | 11-5 |
| Directing Output to the Screen, Windows and Files | 11-8 |
| Screen Output | 11-8 |
| Window Output | 11-8 |
| File Output | 11-9 |
| Program Templates and Programs | 11-10 |

Customizing Help

| | |
|---|-------|
| Getting Context-Sensitive Help | 12-1 |
| Understanding FOXHELP | 12-2 |
| Help Database Requirements | 12-3 |
| FOXHELP Topics | 12-3 |
| FOXHELP Details | 12-3 |
| FOXHELP Cross References | 12-4 |
| Tailoring the Help Display | 12-5 |
| Specifying a Help Database | 12-5 |
| Narrowing Displayed Help Topics | 12-5 |
| Grander Schemes | 12-10 |
| Help File Codes | 12-11 |

Documenting Applications with FoxDoc

| | |
|--|-------|
| Overview | 13-2 |
| Getting Started | 13-4 |
| FoxDoc Files | 13-4 |
| Moving Around In FoxDoc | 13-4 |
| Function Key Options | 13-4 |
| A Quick Run Through | 13-6 |
| Status Screen | 13-7 |
| FoxDoc System Screen | 13-9 |
| FoxDoc Report Screen | 13-13 |
| FoxDoc Format/Action Diagram Options Screen | 13-19 |
| FoxDoc Xref (Cross-Reference) Options Screen | 13-26 |
| FoxDoc Headings Options Screen | 13-28 |
| FoxDoc Tree Diagram Screen | 13-31 |
| FoxDoc Printing Options Screen | 13-34 |
| FoxDoc Other Options Screen | 13-38 |
| FoxDoc Commands | 13-40 |
| Macros | 13-40 |
| DOCCODE: Pseudo Program Statements | 13-42 |
| Other FoxDoc Directives | 13-43 |
| Using FoxDoc in a Batch Environment | 13-45 |

| | |
|---|-------|
| Program Limitations and Miscellaneous Notes . . | 13-46 |
| Memory Usage | 13-46 |
| Continuation Lines | 13-46 |
| Multiple Procedure Files | 13-47 |
| Command Line Switches | 13-47 |
| Changing, Saving and Restoring Default Options | 13-49 |
| Default File Names for Report Output | 13-50 |
| FoxDoc File Types Identification | 13-51 |
| Cross-Reference Codes | 13-55 |
| Batch Programs | 13-57 |
| Keyword File List Information | 13-59 |
| Indentation | 13-61 |
| Symbols | 13-62 |
| Sample Reports | 13-64 |
| Sample Main Program/Project File | 13-65 |
| System Summary | 13-66 |
| Tree Diagram | 13-68 |
| Procedure and Function Summary | 13-69 |
| Database Structure Summary | 13-70 |
| Database Summary | 13-72 |
| Index File Summary | 13-74 |
| Report Form File Summary | 13-76 |
| Token Cross-Reference Report | 13-78 |
| Public Variable Summary | 13-79 |
| Macro Summary | 13-80 |
| Array Summary | 13-81 |
| File List | 13-82 |

Optimizing Your Application

| | |
|---|-------|
| The Rushmore Technology | 14-2 |
| Rushmore with Multiple Databases | 14-4 |
| Rushmore with Single Databases | 14-4 |
| Basic Optimizable Expressions | 14-5 |
| Combining Basic Optimizable Expressions . . | 14-6 |
| Combining Complex Expressions | 14-7 |
| When Rushmore Is Not Available | 14-9 |
| Disabling Rushmore | 14-9 |
| General Performance Hints | 14-10 |

Compatibility

| | |
|--|-------|
| FoxBASE+ Compatibility | 15-2 |
| Emulating FoxBASE+ Keystroke Assignments | 15-2 |
| SET Options for FoxBASE+ Emulation | 15-3 |
| Unavoidable Differences | 15-4 |
| SET COMPATIBLE | 15-7 |
| Converting Files from FoxBASE+ 2.10 | 15-8 |
| .NDX Index Files | 15-9 |
| .DBT Memo Files | 15-9 |
| FOX Program Files | 15-10 |
| Compiling Programs | 15-10 |
| Executing Programs | 15-11 |
| Converting Files from FoxPro 1.XX | 15-13 |

FoxPro in a Multi-User Environment

| | |
|--|-------|
| System Configuration | 16-2 |
| Temporary Work Files | 16-2 |
| CONFIG.FP | 16-3 |
| FOXUSER Resource File | 16-4 |
| Programming in a Multi-User Environment | 16-6 |
| Exclusive Use versus Shared Use | 16-6 |
| Commands that Require Exclusive Use | 16-7 |
| Write Access versus Read-only Access | 16-8 |
| Record and File Locking | 16-8 |
| Automatic versus Manual Locking | 16-9 |
| Unlocking Records and Database Files | 16-9 |
| Commands that perform Automatic Locking | 16-10 |
| SET REPROCESS | 16-12 |
| Manual Locking Functions | 16-13 |
| Collision Management | 16-14 |
| Error Handling Routines | 16-14 |
| The Low-Level File Functions | 16-16 |
| Optimizing Performance | 16-17 |
| Place the Temporary Files on a Local Drive | 16-17 |
| Sorted Files versus Indexed Files | 16-17 |
| Exclusive Use of Files | 16-17 |
| Length of Lock | 16-18 |
| Multi-User Commands and Functions | 16-19 |

Printer Drivers

| | |
|---|-------|
| Printer Driver Overview | 17-3 |
| Using FoxPro's Sample Printer Drivers | 17-8 |
| Specifying a Printer Driver | 17-8 |
| Creating a New Printer Driver Setup | 17-9 |
| Modifying an Existing Printer Setup | 17-10 |
| Deleting a Printer Setup | 17-10 |
| Specifying a Default Printer Setup | 17-11 |
| Loading a Printer Driver Setup | 17-11 |
| Clearing the Current Printer Setup | 17-11 |
| Specifying Printer Procedures Interactively | 17-12 |
| Creating Custom Printer Drivers | 17-13 |
| Printer Driver Programs | 17-13 |
| Printer Driver Procedures | 17-14 |
| PDONLOAD | 17-15 |
| PDONUNLOAD | 17-15 |
| PDDOCST | 17-15 |
| PDDOCEND | 17-16 |
| PDPAGEST | 17-16 |
| PDPAGEEND | 17-17 |
| PDLINEST | 17-17 |
| PDLINEEND | 17-17 |
| PDOBJST | 17-17 |
| PDOBJECT | 17-19 |
| PDOBJEND | 17-19 |
| PDADVPRT | 17-20 |
| Printer Procedures Notes | 17-20 |
| _PDPARMS | 17-21 |
| Designating a Printer Driver Program | 17-22 |
| Custom Printer Driver Setup Applications | 17-28 |

Appendix

Error Messages

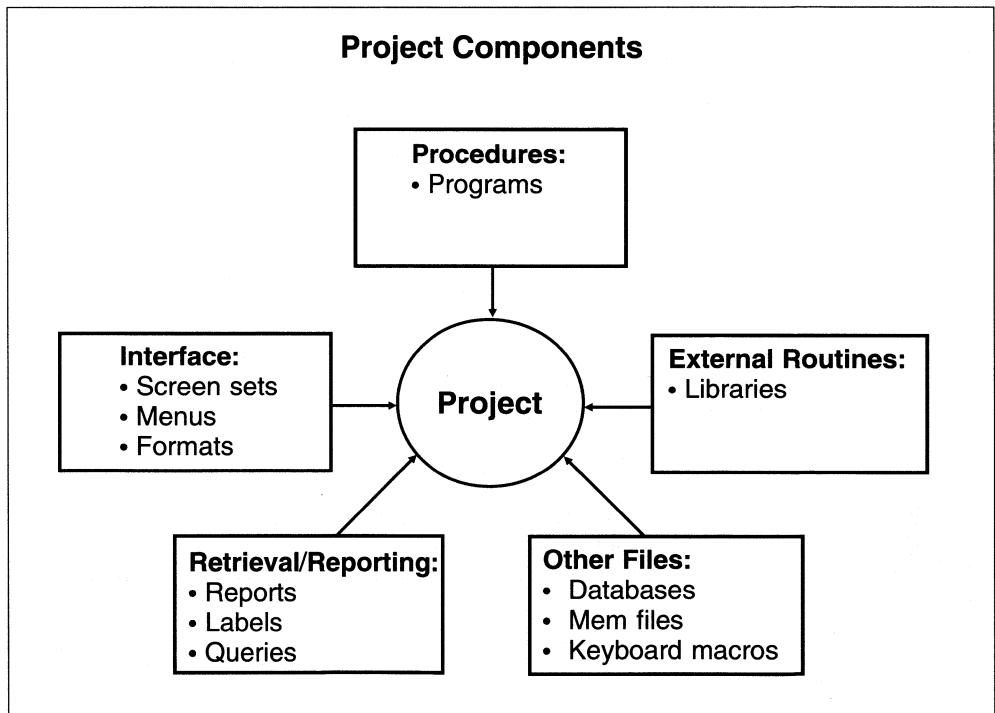
Index

Putting It All Together

1 Overview of Putting It All Together

Chapters in Putting It All Together illustrate how to use FoxPro® power tools to develop an application. FoxPro power tools automate the construction of user interfaces, the retrieval and display of information, the gathering of application components from various locations into an .APP or .EXE file, and the updating of applications when components change.

A project is the unifying mechanism that gathers the pieces of an application together, as shown in the following figure.



Each chapter in Putting It All Together includes examples, explanations and recommended techniques to help you get the most out of FoxPro.

To get the most out of this section, you should know how to operate the power tools and have a basic understanding of the FoxPro language. If you need to learn about the power tools, refer to the *FoxPro User's Guide* or the *FoxPro Getting Started* manual. If you are interested in the details about a particular command or function, refer to the *FoxPro Language Reference*.

Chapters in Putting It All Together contain examples from the ORGANIZER sample application provided with FoxPro version 2.5. This application is in the SAMPLE subdirectory. Feel free to investigate the ORGANIZER in depth.

To use the ORGANIZER, execute ORGANIZE.APP. This application adds two options to the **System** menu: **Organize...** and **Conversions**. The **Conversions** option allows you to convert from one unit of measurement to another. When you choose the **Organize...** option, a submenu appears, displaying the following options:

- **Restaurants** organizes information about restaurants.
- **Client Manager** organizes information about clients.
- **Money Manager...** displays a submenu with the following options:
 - **Credit Cards** organizes credit card information.
 - **Accounts** organizes bank account information.
 - **Transactions** organizes business transaction information.
- **Family & Friends** organizes information about family members and friends.



The ORGANIZER code used in the examples might differ slightly from the ORGANIZER code on your disk.

The ORGANIZER application consists of eight projects: ACCNTS.PJX, TRANS.PJX, CLIENTS.PJX, FAMILY.PJX, CREDIT.PJX, CONVERT.PJX, RESTAURS.PJX and ORGANIZE.PJX.

2 Screens

When you build a screen, you are creating a piece of source code for your application. Information about the screen is saved in an .SCX database. This database has an associated memo field with an .SCT extension. This screen file contains:

- Information to define windows (if the screen is defined as a window)
- Information to define the size, position, and appearance of all fields and controls
- Information about the environment (if it is saved with the screen)
- All underlying code (defined in code snippets) to define the behavior of the screen and the objects within the screen

GENSCRN, the FoxPro screen generator, extracts information from the .SCX database and creates a screen program file with an .SPR extension.



An .SPR screen program file should never be edited. When you need to make changes to a screen, they should be made to the screen itself (using the FoxPro Screen Builder).

This chapter assumes knowledge of the Screen Builder and how to create objects and define code snippets. It also assumes knowledge of FoxPro commands and functions.

Examples throughout this chapter demonstrate how to use the clauses associated with each READ and object level clause to define the behavior of a screen and the objects within the screen. All examples are taken from the ORGANIZER application provided with FoxPro. For more information about the ORGANIZER, see the chapter titled Putting It All Together in this manual.

We recommend that you open the screen (.SCX) files used in the examples to look at the code in the code snippets for different objects in the screen. Compare the code snippets to see how the objects interact with each other and with other screens in a screen set. You should also run the ORGANIZER application to see how objects and screens behave in an application.

Advantages of the Screen Builder

Save Lots of Time

Taking a minimal amount of time learning to use the Screen Builder now will result in a big payoff immediately. The code generator creates all the code to define the physical placement of fields and controls. It also assigns names to procedures, eliminating the possibility of a name collision in an application.

Organization and Clarity

The Screen Builder provides you with a method to encapsulate interface code and separate it from procedural code.

You can unify an object and the procedures that define its action. Procedures are defined in code snippets that are stored with the object.

WYSIWYG

What-you-see-is-what-you-get! There's no more counting rows and columns to define the position of a field. Imagine trying to count pixels in a graphic environment.

When you design a screen with the Screen Builder, you can see how the screen will look and how different objects interact with each other in the generated screen. You can experiment with different layouts, too.

Increased Productivity

You can design "utility" screens that can be combined with other screens in a screen set. One utility screen can be used over and over in an application without code duplication. Also, if you make a change to the utility screen, the change is reflected in every application that uses the screen.

We have supplied several utility screens that you can use as application building blocks. You can also assemble your own library of reusable screens to reflect your own interface style.

Terms Used in this Chapter

The following is a list of terms used throughout this chapter:

Code snippet — A piece of code associated with an object or screen. A code snippet is stored with the screen.

Control — Push buttons, radio buttons, check boxes, popups, lists and invisible buttons are controls that can be defined in a screen.

Generated code — Code created by GENSCRN, the FoxPro screen generator.

Generator-named procedures — Procedures assigned a unique name by the screen generator. Allowing the generator to name procedures prevents name collision in an application.

Object — Any text, field, box, line or control in a screen.

Object level clause — A clause for a specific object in a screen. A code snippet can be defined for each clause. WHEN, VALID and MESSAGE clauses are available for all objects. An ERROR clause is available for GET and EDIT fields.

READ level clause — A clause for a specific screen. A code snippet can be defined for each clause. ACTIVATE, VALID, DEACTIVATE, SHOW and WHEN clauses are available for screens.

Screen set — A screen set can be just one or a combination of several screens. One .SPR program is generated for a screen set.

.SCX file — A screen database file.

.SCT file — Memo file associated with .SCX database.

.SPR file — Generated screen program file.

.SPX file — Compiled .SPR file.

User-named procedures — Procedures that the user names and calls by name in a code snippet or expression. The alternative to a user-named procedure is a generator-named procedure.

Utility screen — A screen designed to work with other screens when combined in a screen set. Utility screens provide consistency in applications. Changes made to a utility screen are reflected in all modules of an application in which the utility screen is used.

Utility Screens

Utility screens are screens designed to be used multiple times throughout an application or in more than one application. They are often combined with other screens in a screen set. Utility screens are usually designed to be independent of the structure and content of a particular database.

Utility screens can be used in a variety of ways. The ORGANIZER application uses utility screens to move through data files (CONTROL1.SCX) and to locate specific records in data files (BROWSER.SCX).



CONTROL1.SCX

When you use utility screens:

- You can share the same screen among different applications.
- You provide consistency throughout an application. When a utility screen is used in several places, the user becomes familiar with its look and functionality.
- Any changes you make to a utility screen are reflected throughout the application. When you modify a utility screen, the Project Manager makes sure that the latest version of the screen is used in every application. All you need to do is rebuild projects that contain the modified utility screen.

Naming Variables in Utility Screens

When you name variables in utility screens, include the “m.” prefix to avoid a variable name conflict with a field name from a database file.

REGIONAL Variables in Utility Screens

Because utility screens are often combined with other screens in a screen set, it is best to define variables as `REGIONAL` when designing utility screens. The `REGIONAL` command lets you create regional memory variables and memory variable arrays. Memory variables or arrays with identical names can be created without interfering with each other — their values are protected within a “region”.

Using `REGIONAL` variables is described later in this chapter and in the *FoxPro Language Reference*.

Other Screens

Many screens are designed to be used with a specific database or application. The “non-utility” screens:

- Are used only once
- Typically reference information that is unique to the current application

Screen Sets

A screen set can consist of one screen or it may consist of many screens. When code is generated, one .SPR program is created for the entire screen set.

Modularity of Interface Pieces

Usually, screens combined in a screen set are defined as windows. It may help to think of “one screen – one window.” For example, the control panel is a separate screen because it occupies its own window. Don’t think of a screen as “everything on the monitor.” Think of a screen as an entity occupying one window.

Creating a Screen Set

You can create a screen set with the Project Manager. When you add a screen to a project, the Generate Screen dialog appears.

Choose the **Add** push button to add the desired screens to the screen set. At the bottom of the Generate Screen dialog you can name the screen set. The name of the first screen in the screen set is displayed in the text box by default.

The order in which screens are placed in a screen set affects the access order of the screens upon execution. It also affects the generation of READ level code snippets in the .SPR program.

Options in the Generate Screen dialog allow you to edit, remove and arrange the screens. You can also suppress the generation of certain code segments. For information on ordering screens in a screen set and other options in the Generate Screen dialog, see the Screen Builder chapter in the *FoxPro User’s Guide*.



You can save the coordinates specified when arranging screens by placing the screens in a project and saving the project. All information specified in the Generate Screen dialog is saved in the project file.

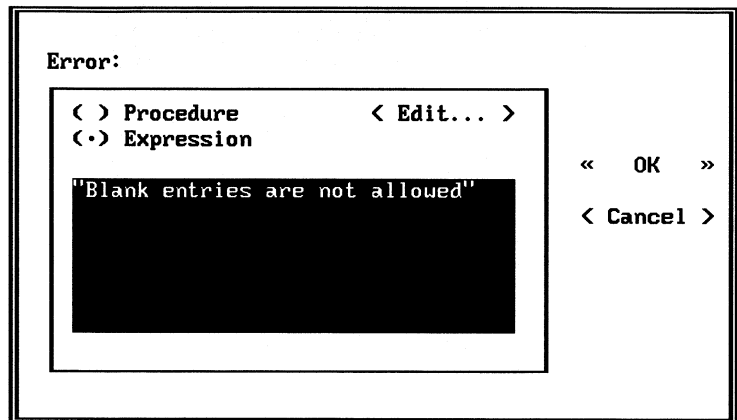


When you define and generate screen sets containing multiple screens, setup code from successive screens is concatenated with the setup code from the first screen. It is possible to write code for one screen that will unintentionally change the desired output of code written for a previous screen. Careful planning and coding will ensure that you obtain the results you intended.

Code Snippets

When you create a screen, you can define code snippets associated with a specific clause for a specific object in a screen. You can also define code snippets that affect the entire screen. Defining code snippets is described in the Screen Builder chapter of the *FoxPro User's Guide*.

When you assign a clause to an object or a screen, you can define an expression or a procedure for the clause.



Defining an Expression

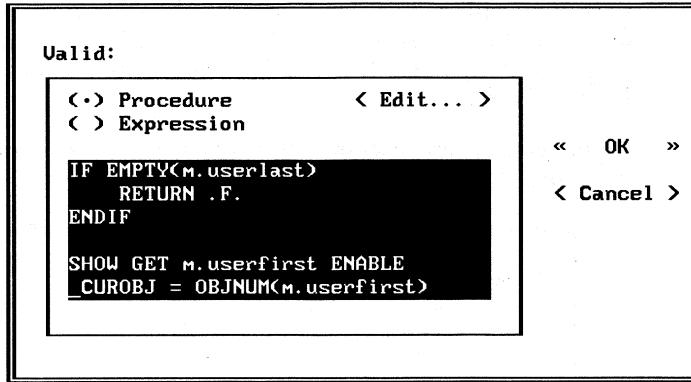
Examples in this section are from the USERLAST field in ADDUSERS.SCX.

If an expression is defined for a clause, the expression is inserted in the generated program with the associated clause.

```
@ 8,41 GET m.userlast ;
.
.
.
ERROR "Blank entries are not allowed" ;
DISABLE
```

If the expression calls another procedure, you can define that procedure in the Cleanup & Procedure code or it can be any procedure located in your path. Cleanup & Procedure code is described later in this chapter.

If a procedure is defined for a clause, the procedure is assigned a unique name by the SYS(2015) function.



Defining a Procedure

This unique name is inserted in the generated program with the clause:

```
@ 8,41 GET m.userlast ;
.
.
.
VALID _px20n0uaf()
```

The procedure is generated at the end of the program file. The unique name associated with the clause becomes part of a FUNCTION command and the code snippet follows.

```
*****
* _PX20NOUAF          m.userlast VALID          *
*                                                           *
* Function Origin:                                          *
*                                                           *
* From Screen:      ADDUSERS,      Record Number:      8  *
* Variable:         m.userlast                                           *
* Called By:        VALID Clause                                          *
* Object Type:      Field                                                 *
* Snippet Number:   6                                                     *
*****
*
FUNCTION _px20n0uaf      && m.userlast VALID
#REGION 1
IF EMPTY(m.userlast)
    RETURN .F.
ENDIF
```

Function
origin comments

```
SHOW GET m.userfirst ENABLE  
_CUROBJ = OBJNUM(m.userfirst)
```

It is not necessary to include a RETURN command in the code snippet. If no RETURN statement is included, a .T. is automatically returned.



The unique name assigned to a code snippet changes every time you generate a screen. When you want to change a screen, you must modify the screen (the .SCX file) and then regenerate the code. If you make modifications in the generated program (the .SPR file) and then regenerate the screen, all changes will be lost.

In the generated code, all procedures are documented with the unique name and function origin comments. These comments describe the screen, object and clause with which the procedure is associated. Comments are also inserted in the generated program so that FoxDoc can document your programs.

Calling a Screen Program

When you call a screen program, you must use the following syntax:

```
DO <filename>.SPR
```

Screen programs (.SPR), menu programs (.MPR), programs (.PRG), queries (.QPR), projects (.PJX) and applications (.APP) are all assigned different extensions. This allows these files to have the same base names, yet not overwrite programs on disk.

Compiled screen programs have an .SPX extension. Be sure to use the .SPR or .SPX extension when calling a screen program.

Significance of READ

READ is the operative command used to animate and coordinate sets of screens, menus and other windows into an interactive session. Other statements in a screen program define the appearance and behavior of objects in the screen. The READ command makes the objects come alive.

READ level clauses and the actions they perform are:

- | | |
|-------------------|---|
| Activate | This routine is used to disable objects in other windows, hide windows, display messages, etc. |
| Deactivate | This routine is used to keep the current READ window active (not allow another window to become the output window). It can also be used to terminate (or not terminate) a READ based on the RETURN value. |
| Show | This routine is used to refresh SAY and GET fields, enable and disable GET objects. |
| Valid | This routine is used to determine if a READ can be exited based on the result of a logical expression. |
| When | This routine is used to determine if the READ is executed based on the result of a logical expression. The WHEN clause can also be used to set the environment for a READ. For example, if you want a menu available in a modal read, you would execute the menu program in the READ WHEN clause. |
| Modal | When a window is defined as modal, the window assumes the behavior of a FoxPro dialog. This means that windows outside of the screen set <i>cannot</i> be brought forward on top of the screen set window and the current menu system is temporarily deactivated. |

With You can include a list of windows that can be activated along with a screen set if an Associated Window list is included in the Generate dialog. This window list is added to the READ command. The syntax for the WITH clause is:

```
READ WITH <window title>
```

The **ACTIVATE** clause can define the behavior of screens in the Associated Window list. It is executed only when the activating window is a READ window. It is not executed when Browse windows, desk accessories or other non-READ windows are included with the screen set.

The **WITH** clause automatically makes the screen set modal. Only those windows defined in the screen set and specified in the Associated Window list can be activated.

To define a screen as Modal or assign an Associated Window list:

1. Choose the Generate option on the **Program** menu popup when the screen is frontmost or choose the **Edit** push button when building a project. The Generate Screen dialog appears.

Defining an Associated Window list automatically makes the screen set modal.

2. Choose the **Modal** or **Associated Windows** check box. Modal defines the screen as modal (as described above).

When you choose the **Associated Windows** check box, the Associated Window dialog appears. This dialog allows you to specify the Associated Window that can be activated with the screen set.

This is a restrictive list. Only those windows specified in the list can be activated. If other windows are present when the screen is executed, they will appear on the monitor but cannot be activated or accessed.

3. Specify the windows you want to include with the screen in the Associated Window list. These windows can be activated with those defined in the Screen Set. It is not necessary to include the windows defined in the Screen Set in the Associated Window list.

Any Browse or memo windows or desk accessories you would like to access with your screens should be included in the Associated Window list.



Rule 1 — to access a memo window while in a modal READ, include the database alias in the associated window list.

Rule 2 — To access a Browse window while in a modal READ, include the Browse window title (by default, the database alias) in the Associated Window list.

For more information and an example of the Associated Window list and the READ WITH clause, see the section on Coordinating Browse with Screens later in this chapter.

Your Working Environment

Working in 50 Line Mode

If your machine supports an extended display mode, use it! Developing in 50 line mode allows you to display many code snippet windows simultaneously. You can see the code for one object while creating code for another object and, at the same time, see the Design window for the screen. You can also have several Screen Design windows open at once.

Cutting, Copying and Pasting Between Windows

You can cut, copy and paste code from one editing window into another even if the code is from different screens. You can also cut, copy and paste objects between several Screen Design windows. When you copy and paste an object from one screen to another, all the information associated with the object (including any code snippets) is copied as well.

Manipulating Code Snippet Editing Windows

You can size, minimize and dock code snippet editing windows just as you can other text editing windows. When you save the screen, the state of windows is saved as well. When you open the screen, all code snippet editing windows appear as they did when you closed the screen.

From the **Screen** menu popup you can choose **Open All Snippets** or **Close All Snippets** to open and close all code snippet editing windows at once. When a code snippet window is open, the clause or option with which the snippet is associated is dimmed in the corresponding dialog.

This provides you with a visual clue as to the state of the window. All open code snippet editing windows are listed at the bottom of the **Window** menu popup. You can make any code snippet editing window the active window by choosing it from this menu popup.

Design Considerations

Window Types

Your applications should have a consistent look. The appearance of a window provides the user with a visual clue about the behavior of the window. For example, make all your input screens one window type, all dialogs another, alerts another type, and so on.

For more information on window types, see the Defining Windows section later in the chapter.

Screen Controls

Placing controls in a screen reduces the need for menu options. If you can make a menu option available through a control in the screen (without making the screen cluttered), do it.

Reserve menu options for seldom used and irreversible options and keyboard shortcuts for screen controls. For more information on designing menus, see the Menus chapter in this manual.

Access Order of Screens and Objects

Mouse users can point and click between screens and between objects in a screen. Keyboard users do not have this option. Design screens with both types of users in mind.

Screens are accessed in the order they appear in a screen set. Objects within a screen are accessed in the order they are defined. This is the order the objects are numbered in the Screen Design window.

Objects should be ordered in a screen in an intuitive manner. Ordering objects by row, column or region is desirable.

For information on ordering screens in a screen set and objects in a screen, see the Screen Builder chapter of the *FoxPro User's Guide*.

Disabled Item

If a field or control has no meaning until another action occurs (for example, you fill in a field to enable a push button), it should be disabled. A code snippet for one object can include code that will enable other fields and controls. Examples in this chapter demonstrate how to achieve this result.

Hot Keys

Mouse users can point and click anywhere in the screen. Keyboard users do not have this option. Hot keys allow the user to move to a desired control with one key press.

For example, when a user edits one field in a screen, a hot key can enable him to access a control that will save the information.

If a user is in a GET field or an EDIT region, he will need to exit the GET field or EDIT region before the hot key is available.

Default and Escape Push Buttons

A default push button is surrounded by « » and is automatically chosen when the user presses Ctrl+Enter. Default buttons usually take an action (for example, save the information in the screen) before exiting the screen.

An escape push button appears as any other push button and is activated when the user chooses the button or presses Escape. Escape buttons usually exit the screen without taking any action (cancel).

SCATTER MEMVAR and GATHER MEMVAR vs. Direct Editing

The SCATTER MEMVAR command allows you to create memory variables for every field in the current database record. When you create these variables and define the fields with an “m.” prefix, the editing of fields takes place on the variables, not directly on the database. This allows the user to cancel out of or escape from a screen without saving any changes.

If variables are not created, when the user exits the READ (by moving to another record or exiting the screen), any modifications to the field are saved immediately. When you edit variables, the changes are not saved until a GATHER MEMVAR command is executed.

Most of the screens in the ORGANIZER application have a SCATTER MEMVAR command in the READ SHOW code snippet. The GATHER command is located in the VALID code snippet for the **Save** push button. Changes are not saved until the user chooses the **Save** push button.

Color

When you assign a color scheme to a screen, every object in the screen takes on the attributes of that color scheme. You can, however, assign a different color scheme to individual objects within a screen.



Because an object can be colored differently at different points (selected, disabled, enabled, hot key), a single color pair is not enough to color the object. For this reason, a color scheme is required.

If an object is assigned a color scheme which is different than the scheme assigned to the screen, the color scheme assigned to the object takes precedence. Every object in a screen can be defined with a different color scheme.

For information on assigning color schemes to objects, see the Screen Builder chapter of the *FoxPro User's Guide*. For information on color schemes, see the Customizing FoxPro chapter in the *FoxPro Installation and Configuration* manual.

Following is a list of tips for using color:

- Use color as a complementary feature to provide extra information for those users who have color capability.
- Colors look best against a background of neutral gray. Studies have shown colored text is harder to read than black text on a white background. Beware of light shades of blue, which are generally the most illegible of all colors.
- If all users of the application have a color monitor, color can be used to distinguish objects.

The Generated Program

GENSCRN, the FoxPro screen generator, extracts information from .SCX databases and creates a program file with an .SPR extension. .SPR programs are generated in the following order:

- Setup Code — Section 1
- Program environment code (opening)
- Open file commands
- Define window commands
- Setup Code — Section 2
- Screen layout commands
- READ command
- Release window commands
- Close file commands
- Program environment code (closing)
- Cleanup and procedure code
- READ and object level code snippets

The example on the following pages is from CONVERT.SPR. This example shows how code appears in the generated program. Examples of code snippets and how they are used to manipulate specific screens are described throughout this chapter.

```

* *****
* * 07/18/91          CONVERT.SPR          11:26:39 *
* *****
* * Fox Software Systems Group
* *
* * Copyright (c) 1991 Fox Software, Inc.
* * 134 W. South Boundary
* * Perrysburg, OH 43551
* *
* * Description:
* * This program was automatically generated by GENSCRN.
* *****

```

Program Header — This code is *always* generated. The author and address information in the program header is taken from the Screen Code Options dialog.

```

* *****
* *          CONVERT Setup Code - SECTION 1          *
* *****
#REGION 1
m.quitting = .F.
IF RDLEVEL()=0
    SET PROCEDURE TO utility
    ON ERROR DO errorhandler WITH MESSAGE(), LINENO()
    CLEAR PROGRAM
    CLEAR GETS

    IF SET("TALK") = "ON"
        SET TALK OFF
        m.talkstat = "ON"
    ELSE
        m.talkstat = "OFF"
    ENDIF

    m.area   = 0
    m.exact  = ""
    .
    .
    .
    m.hidecomm = WVISIBLE("command")

    DO setup
ENDIF

```

Setup Code – Section 1 — You can define code to be inserted and executed at the beginning of the generated program by placing generator directives in the setup code for the screen (defined in a code snippet with the Setup option in the Screen Layout dialog).

The Generated Program

```
#REGION 0
REGIONAL m.currarea, m.talkstat, m.compstat

IF SET("TALK") = "ON"
    SET TALK OFF
    m.talkstat = "ON"
ELSE
    m.talkstat = "OFF"
ENDIF
m.compstat = SET("COMPATIBLE")
SET COMPATIBLE FOXPLUS

m.currarea = SELECT()
```

Program Environment Code — This information is *always* generated. This code defines regional variables and makes environment settings for the entire generated program.

```
*          *****
*          *          CONVERT Databases, Indexes, Relations          *
*          *****

IF USED("units")
    SELECT units
    SET ORDER TO 0
ELSE
    SELECT 0
    USE (LOCFILE("dbfs\units.dbf", "DBF", "Where is units?"));
        AGAIN ALIAS units ;
        ORDER 0
ENDIF

IF USED("factors")
    SELECT factors
    SET ORDER TO 0
ELSE
    SELECT 0
    USE (LOCFILE("dbfs\factors.dbf", "DBF", "Where is factors?"));
        AGAIN ALIAS factors ;
        ORDER 0
ENDIF

SELECT units
```

Open file commands — These commands are generated when environment information has been saved (in the Screen Layout dialog) and Open Files is checked in the Generate Screen dialog. You can suppress the generation of these commands by unchecking the Open Files check box. If you choose to suppress the generation of these commands, you can open files in the setup code for the screen.

```

*          *****
*          *                               *
*          *           Window definitions           *
*          *          *****
*          *
IF NOT WEXIST("convert")
    DEFINE WINDOW convert ;
        FROM INT((SROW()-15)/2),INT((SCOL()-53)/2);
        TO INT((SROW()-15)/2)+14,INT((SCOL()-53)/2)+52;
        TITLE " Conversions " ;
        FLOAT ;
        CLOSE ;
        SHADOW ;
        MINIMIZE ;
        SYSTEM ;
        COLOR SCHEME 8
ENDIF

```

Window Definitions —
DEFINE WINDOW commands are generated when a window is defined (in the Screen Layout dialog) and Define Windows is checked in the Generate Screen dialog. You can suppress the generation of these commands by unchecking this check box. If you choose to suppress the generation of these commands, you can define the windows in the setup code for the screen.

```

*          *****
*          *                               *
*          *   CONVERT Setup Code - SECTION 2   *
*          *          *****

```

```

*#REGION 1
SET UDFPARMS TO REFERENCE
PUSH MENU _MSYSMENU

.
.
.

m.size = ALLEN(fromarry)
DIMENSION toarry[m.size]
FOR m.i = 1 TO m.size
    fromarry[m.i] = ALLTRIM(fromarry[m.i])
    toarry[m.i] = fromarry[m.i]
ENDFOR

m.frompop = fromarry[1]
m.topop   = toarry[1]

```

Setup Code – Section 2 —
This segment of the generated program includes all setup code that follows the #SECTION 2 generator directive. If the setup code contains no generator directives, it is placed in this segment of the screen program.

The Generated Program

```
* *****
*          CONVERT Screen Layout          *
* *****
*#REGION 1
IF WVISIBLE("convert")
    ACTIVATE WINDOW convert SAME
ELSE
    ACTIVATE WINDOW convert NOSHOW
ENDIF
@ 7,28 TO 12,49 ;
    COLOR W/BG
@ 7,1 TO 12,22 ;
    COLOR W/BG
@ 7,3 SAY " From: "
@ 7,30 SAY " To: "
@ 8,2 GET m.fromval ;
    SIZE 1,20 ;
    DEFAULT " " ;
    PICTURE "@TJK" ;
    VALID convrt (toval, fromval, "right")
@ 9,4 GET m.frompop ;
    PICTURE "@^" ;
    FROM fromarry ;
    SIZE 3,16 ;
    DEFAULT 1 ;
    VALID _px80oj5d3() ;
    .
    .
    .
@ 0,17 GET m.unittype ;
    PICTURE "@*RVN Ar<ea;\<Length;Ma\<ss;Spee\<d;\<Temperature;T\<ime;Volu\<me" ;
    SIZE 1,15,0 ;
    DEFAULT 1 ;
    VALID _px80oj6xq()
@ 0,10 SAY " Type: "

IF NOT WVISIBLE("convert")
    ACTIVATE WINDOW convert
ENDIF

READ CYCLE ;
    WHEN _px80oj7hk() ;
    DEACTIVATE _px80oj7hq()
```

The window is activated with a NOSHOW clause so the objects can be drawn and the window shown with all the objects in place. This produces a "snappier" effect.

Screen Layout commands — One command is generated for each object in each screen.

ACTIVATE WINDOW command — The ACTIVATE WINDOW command includes a generated window name or a name you specify in the Screen Layout dialog.

Controlled by options in the Generate Screen dialog, a READ or READ CYCLE command is always generated.


```

*          *****
*          *                                     *
*          *          Closing Databases          *
*          *                                     *
*          *****

```

```

IF USED("units")
    SELECT units
    USE
ENDIF

IF USED("factors")
    SELECT factors
    USE
ENDIF

SELECT (m.currarea)

```

Close file commands — These commands are generated only when the Close Files option is checked in the Generate Screen dialog. You can suppress the generation of these commands by unchecking this check box.

```

#REGION 0
IF m.talkstat = "QN"
    SET TALK ON
ENDIF
IF m.compstat = "QN"
    SET COMPATIBLE ON
ENDIF

```

Restore environment code — This code is *always* generated. This code reverses the settings made with the program environment code at the beginning of the program.

```

*          *****
*          *          CONVERT Cleanup Code          *
*          *****
*#REGION 1
IF m.quitting OR RDLEVEL()=0
    RELEASE WINDOW convert
ENDIF
POP MENU _MSYSMENU
SET UDPPARMS TO VALUE
IF RDLEVEL()=0
    DO cleanup
    SET PROCEDURE TO
ENDIF

```

Cleanup code — This code snippet is defined with the Cleanup & Procs... option in the Screen Layout dialog and is *always* included in the generated program when cleanup code has been defined.

The Generated Program

```

*
* CONVRT - Do the conversion.
*
FUNCTION convrt
PARAMETER m.new, m.old, m.direction
PRIVATE m.toid, m.fromid, m.tounit, m.fromunit

IF (VAL(m.old) = 0 AND m.unittyp<>"Temperature") OR ;
    (m.old = SPACE(19) AND m.unittyp="Temperature") OR ;
    m.topop = m.frompop
    m.new = m.old
    SHOW GETS
    RETURN
ENDIF

.
.
.
m.new = stripzeros(m.new)
m.old = stripzeros(m.old)
SHOW GETS
.
.
.

```

User-named procedures are defined in the Cleanup and Procs... code snippet and appear before the generator-named procedures.

```

* *****
* * _PX800J5D3          m.frompop VALID          *
* * * * * * * * * * * * * * * * * * * * * * * * *
* * Function Origin: *
* * * * * * * * * * * * * * * * * * * * * * * * *
* * From Screen:      CONVERT,      Record Number:  9 *
* * Variable:         m.frompop *
* * Called By:        VALID Clause *
* * Object Type:      Popup *
* * Snippet Number:   1 *
* * *****
*
FUNCTION _px800j5d3      && m.frompop VALID
#REGION 1
IF EMPTY(m.fromval)
    _CUROBJ = OBJNUM(m.fromval)
    SHOW GET m.fromval
    RETURN .F.
ENDIF
= convrt (m.fromval, m.toval, "left")

```

Object level clause

```

*          *****
*          * _PX800J7HK          Read Level When          *
*          *          *          *          *          *
*          * Function Origin:          *
*          *          *          *          *          *
*          * From Screen:          CONVERT          *
*          * Called By:          READ Statement          *
*          * Snippet Number:          6          *
*          *****

```

```

*
FUNCTION _px800j7hk      && Read Level When
*
* When Code from screen: CONVERT
*
#REGION 1
DO convmenu.mpr
IF RDLEVEL()>1
    SET SKIP OF POPUP _MRECORD .T.
    SET SKIP OF POPUP reports .T.
    SET SKIP OF POPUP cardinfo .T.
ENDIF

```

— READ level clause

Screen Layout

Choose **Screen Layout...** on the **Screen** menu popup to bring forward the Screen Layout dialog.

(< >) DeskTop < > Window
 Name: <Type...>
 Title:
 Footer:
 Size: Screen Code:
 Height: 25 [] Setup...
 Width: 80 [] Cleanup & Procs... « OK »
 Position: READ Clauses: < Cancel >
 Row: [] Activate... [] Show...
 Column: [] Valid... [] When...
 [X] Center [] Deactivate...
 Environment: [X] Add alias
 < Save > < Restore > < Clear >

Screen Layout Dialog

This dialog contains options for defining a window including color, sizing and position, defining code snippets for setup and cleanup code, and defining code snippets for READ level clauses.

Setup Code

Code in the Setup code snippet can be used to:

- Save the current environment (to be restored later). This often includes saving the value of certain SET commands.
- Create a new environment.
- Define memory variables and arrays.
- Save the current menu system by pushing it on the menu stack. Pushing menus is described in the Menus and Coordinating Screens and Menus chapters in this manual.
- Call other programs (for example, a program to install a menu system).
- Receive parameters (using generator directives).
- Specify the error handling routine that is called when an error is generated (ON ERROR).
- Open files.
- Define windows.
- Specify a procedure file.

Setup Code Example 1

This example is from FAMILY.SCX. This screen is used in the Family & Friends module of the application. Setup code is used to define variables and push a menu system.

Family/Friends Manager

Last Name:
Gossnergan

First Name:
Ed

Initial:

Spouse: Turan

Birth: 11/10/64

Phone Number: 303-664-6981

Address: 321 Strata Ln.
Boulder, CO 80303

Notes:

[X] Send Holiday Cards

[] Special Diet Needs

[] Exchange Gifts

CTRL+TAB to exit

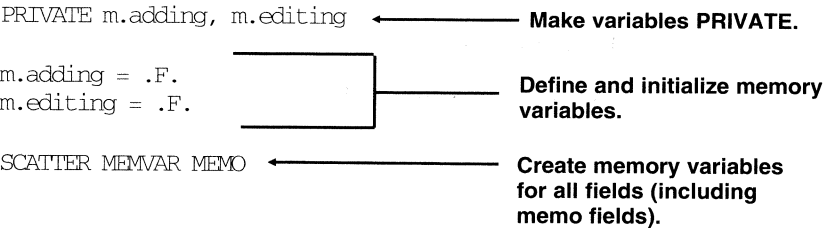
< Help >

< New >

< Save >

< Cancel >

FAMILY.SCX



Regional Variables

Often, a control in a screen uses the same variable name as a control in another screen. When these screens are combined into a screen set and a single program is created, a conflict occurs with the variable common to both screens. REGIONAL variables are used to avoid this type of conflict.

REGIONAL variables are similar to private variables. Memory variables or arrays with identical names can be created without interfering with each other — their values are protected within a “region”.

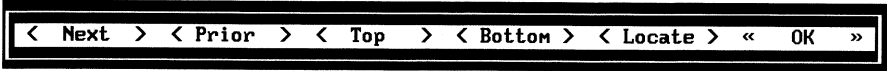
Declaring screen variables as regional in the setup code ensures that a variable is not affected by a variable with the same name used in another screen in the screen set. If you declare variables in screens in a screen set as REGIONAL variables in the setup code, FoxPro automatically resolves the conflicts.

When regional variables are declared in the screen setup code, the GENSCRN screen generating program automatically inserts the necessary #REGION compiler directives needed to resolve memory variable name conflicts. #REGION compiler directives are inserted at the beginning of the setup code for each screen, at the beginning of the screen layout statements for each screen, at the beginning of the cleanup code for each screen and in the READ and Object level code snippets for each object.

For more information on REGIONAL variables, see the REGIONAL command in the FoxPro *Language Reference*.

Setup Code Example 2

This example is from CONTROL1.SCX, a utility screen used throughout the ORGANIZER application. Setup code is used to define regional variables.



CONTROL1.SCX

```
#SECTION 2
REGIONAL m.choice, m.toprec, m.bottomrec, m.saverecno, m.quitting
m.quitting = .F.
m.choice = "OK"
```

Generator directive (described in next section).

Make variables REGIONAL. Initialize the variable for the "OK" push button.

```
IF EOF()
  GO BOTTOM
ENDIF
```

If the file is open and at end of file, position at the bottom record.

```
m.saverecno = RECNO()
GO TOP
m.toprec = RECNO()
GO BOTTOM
m.bottomrec = RECNO()
GO m.saverecno
```

Initialize variables with top, bottom and current RECNO().

Return the record pointer to the current record.

Generator Directives

Generator directives were implemented in FoxPro to allow the user to include code in a generated screen program that would not otherwise be available through the Screen Builder.

Generator directives are commands that communicate solely with GENSCRN, FoxPro screen program code generator. Generator directives do not appear in generated screen program code. Generator directives must be placed in the Setup code for a screen. Only one directive of each type can be included in the code.

#SECTION 1 | 2

These generator directives allow you to split the setup code for a screen into two sections. Setup Code — Section 1 is generated at the beginning of the .SPR program. Setup Code — Section 2 is generated after the DEFINE WINDOW commands and before the Screen Layout commands. You would split the setup code if you wanted to include a PARAMETER statement or an ON ERROR statement in the screen program.

Precede commands to be inserted in section one of the setup code with a #SECTION 1. If you would like other setup code placed in Setup Code — Section 2, a #SECTION 2 generator directive should follow commands in section 1.

```
#SECTION 1
    PARAMETER x, y, z
    <other commands>
#SECTION 2
    <other commands>
```

#READCLAUSES <clauses>

This generator directive allows you to specify clauses to be placed at end of a READ command that are not available through the Screen Builder. Typical clauses to include are TIMEOUT, SAVE, OBJECT, NOMOUSE and COLOR.

```
#READCLAUSE TIMEOUT SAVE OBJECT COLOR
```

- * These READ clause must be
- * placed on one line in the
- * setup code.

#ITSEXPRESSION <char>

This generator directive allows you to specify a single character that can be used to indicate that picture clauses, window titles and window footers are expressions instead of literal strings.

```
#ITSEXPRESSION ~
#SECTION1
varname = "Fred"
* A window title can be defined
* as ~<varname>. When the window is displayed, Fred
* appears as the window title.
```

The #SECTION1 generator directive is included in this example because the windows are defined before Setup Code — Section 2 is executed. #ITSEXPRESSION is applicable only to windows defined in the Screen Layout dialog in the Screen Builder.

#WNAME <string>

This generator directive allows you to substitute the current window name for <string> wherever it occurs in code. This allows you to use generic code which is independent of the unique window name generated by GENSCRN.

The following statement is defined in setup code for screen:

```
#WNAME fred
```

The following statements are defined in the ACTIVATE clause for the window:

```
IF WONTOP('fred')
    <statements>
ELSE
    <statements>
ENDIF
```

A unique name for current window is substituted for “fred”. It is not necessary to know actual window name.

With the above example, the following code appears in the generated screen program:

```
IF WONTOP('_PV60NIAGH') && Unique named window
    <statements>
ELSE
    <statements>
ENDIF
```

#REDEFINE

By default, GENSCRN generates code to check for the existence of a window before defining it. This generator directive allows you to suppress the commands that check for existence of a window and automatically redefine window.

Setup Code Example 3

This example is from ADDUSERS.SCX, a screen called in the Credit Cards module of the ORGANIZER application. Setup code is split into two sections. In section two of the setup code, an array is defined and filled with a SQL SELECT statement.

The screenshot shows a window titled 'ADDUSERS.SCX'. It contains two main sections: 'Selection list:' and 'Authorized users:'. The 'Selection list:' section has a list box containing the names: Jon Jeager, Arden Schwartz, Bonnie Schwartz, Geoffrey Schwartz, Nadine Schwartz, and Pat Schwartz. The 'Authorized users:' section has a list box containing: Geoffrey Schwartz and Nadine Schwartz. Between these two sections are several buttons: '< Move + >', '< + Remove >', '< Remove All >', '< New name >', '<< OK >>', 'Last: < Help >', and 'First:'. Below the 'Last:' and 'First:' labels are input fields.

ADDUSERS.SCX

#SECT1 ← **Generator directive.**
 PARAMETER m.cardid ← **Parameter statement.**

#SECT2 ← **Generator directive.**
 PRIVATE m.mover, m.user, m.allcnt, m.saverec, m.usrcnt, m.limit,;
 allusers, m.status, m.savearea, m.userlast, m.userfirst

SET EXACT ON ← **Environment setting.**
 m.user = 1
 m.userlast = ""
 m.status = .T.
 m.savearea = SELECT()
 DIMENSION allusers[1,3]
 allusers = ""

IF NOT locatedb("carduser",1)
 RETURN
 ENDIF

**Define and initialize variables;
 create an array to prevent an
 error if CARDUSER.DBF is
 empty.**

**Call UDF LOCATEDB() to
 locate .DBF and the associated
 .FPT. LOCATEDB() is defined
 in MAIN.PRG.**

m.saverec = RECNO()
 SELECT DISTINCT lastname, firstname, ;
 ALLTRIM(firstname)+" "+ALLTRIM(lastname) ;
 FROM carduser ;
 INTO ARRAY allusers

**Fill the array using
 SQL SELECT.**

```
m.allcnt = ALEN(allusers,1)
```

```
IF EMPTY(users) ←
```

```
    m.usrcnt = 0
```

```
ELSE
```

```
    m.usrcnt = 1
```

```
    m.limit = ALEN(users,1)
```

```
    DO WHILE m.usrcnt <= m.limit
```

```
        IF EMPTY(users[m.usrcnt,1])
```

```
            EXIT
```

```
        ENDIF
```

```
        m.usrcnt = m.usrcnt + 1
```

```
    ENDDO
```

```
    m.usrcnt = m.usrcnt - 1
```

```
ENDIF
```

(users) is an array that was created in the previous screen.

Check the size of the Authorized Users list so that blank records will not be displayed.

Open Files

When screens are generated, code to open files, set index order, set relations, etc., is generated using the environment information saved with the screen. These commands are generated and executed prior to the `DEFINE WINDOW` commands. The **Open Files** check box in the Generate Screen dialog allows you to suppress the generation of these commands.

We recommend that you allow the generator to create the open file commands; however, you can define these commands in the Setup code snippet.

Defining Windows

When screens are generated, code to define windows is generated using the window definition information saved with the screen. These commands are generated and executed prior to the commands in the Setup Code — Section 2 code snippet. The **Define Windows** check box in the Generate Screen dialog allows you to suppress the generation of these commands.

We recommend that you allow the generator to create the `DEFINE WINDOW` commands; however, you can define these commands in the Setup code snippet.

If you do not name a window in the Screen Layout dialog, a unique name is generated for the window. This unique name changes every time you regenerate the screen.

Cleanup and Procedure Code

Cleanup and Procedure code is generated and executed at the end of the .SPR program. This code snippet can be used to:

- Restore environment settings.
- Release public memory variables (by name).
- Restore the previous menu system.
- Release windows.
- Close files.
- Define user-named procedures.

Cleanup and procedure code is used to restore the environment and release public memory variables. It is also used to pop menu systems.

Cleanup and Procedure Code Example 1

This example is from FAMILY.SCX, a screen called in the Family & Friends module of the ORGANIZER application. This code snippet restores environment settings and pops a menu system.

```
IF m.quitting
    RELEASE WINDOW family
    RELEASE WINDOW controls
ENDIF
```

Release windows.

Releasing Public Variables by Name

You should release public variables by name in the cleanup code for a screen.

Issuing a `RELEASE ALL` command in the cleanup code for a screen will release all variables in the currently executing program.

Close Files

When screens are generated, code to close files is generated automatically. These commands are executed before the user-defined cleanup code. The **Close Files** check box in the Generate Screen dialog allows you to suppress the generation of these commands.

We recommend you allow the generator to create the open file commands, however, you can define commands to close files in the Cleanup and Procedure code snippet.

Releasing Windows

`RELEASE WINDOW <window name>` commands are generated automatically for every window defined in a screen set. The **Release Windows** check box in the Generate Screen dialog allows you to suppress the generation of these commands.

We recommend that you allow the generator to create these commands; however, you can close them in the Cleanup and Procedure code snippet.

If you define `RELEASE WINDOW <window name>` commands, it is best to name the window (in the Screen Layout dialog).

User-Named Procedures

If a procedure is used by more than one object in a screen, it is best to call the procedure with a `DO` command in a code snippet for the associated clause or as a UDF in an expression for the associated clause. This “user-named procedure” can be defined in the cleanup and procedure code or it can be any procedure located in your path.

Using a user-named procedure ensures that when you modify the procedure, the changes are reflected in the behavior of *all* objects with which the procedure is associated. If an identical procedure is defined individually in multiple code snippets, you would need to modify the procedure in *every* code snippet.

When you define a procedure in the code snippet, the generator assigns the procedure a unique name that changes every time the screen is regenerated. The unique name is inserted in a `FUNCTION` command and the code in the code snippet follows.

When you define a user-named procedure, you must use a `PROCEDURE` or `FUNCTION` command (depending on how the procedure is called). If you want a value other than `.T.` returned, you must issue a `RETURN` at the end of the procedure. These commands *are not* generated as they are with procedures defined at the `READ` and object level.

Any procedure, whether it is defined in a code snippet or a user-named procedure, can call another procedure. You can define that procedure in the Cleanup and Procedure code snippet. The procedure can also be any procedure located in your path.

Cleanup and Procedure Code Example 2

This example is from ADDUSERS.SCX., a screen called in the Credit Cards module of the ORGANIZER application. The **Last** and **First** GET fields in ADDUSERS.SCX call a procedure named ESCHANDLER from the WHEN clause code snippet.

The screenshot shows a window titled 'ADDUSERS.SCX'. It contains two main sections: 'Selection list:' and 'Authorized users:'. The 'Selection list:' section has a list box containing the names: Jon Jeager, Arden Schwartz, Bonnie Schwartz, Geoffrey Schwartz, Nadine Schwartz, and Pat Schwartz. The 'Authorized users:' section has a list box containing: Geoffrey Schwartz and Nadine Schwartz. Between these two sections are several buttons: '< Move >', '< + Remove >', '< Remove All >', '< New name >', '<< OK >>', and '< Help >'. Below the 'Authorized users:' list box are two input fields labeled 'Last:' and 'First:', each with a '< Help >' button next to it.

ADDUSERS.SCX

The WHEN clause code snippet contains the following code:

```
ON KEY LABEL esc DO eschandler
```

This procedure is defined in the cleanup code for the screen and contains the following code:

```
*  
* ESCHANDLER - Handle ESC-aping out of a field.  
*  
PROCEDURE eschandler  
ON KEY LABEL esc  
m.userlast = SPACE(22)  
m.userfirst = SPACE(14)  
SHOW GET m.userlast DISABLE  
SHOW GET m.userfirst DISABLE
```

Abort entry of new name.

Cleanup and Procedure Code Example 3

This example is from CONVERT.SCX, a screen called in the Conversions module of the ORGANIZER application. In this screen, the **From** and **To** GET fields use an expression to call a UDF.

The screenshot shows a window titled "Conversions". Inside, there is a list of units: Area, Length, Mass, Speed, Temperature, Time, and Volume. To the right of the list are two buttons: "< Help >" and "<< OK >>". Below the list, there are two input fields labeled "From:" and "To:". The "From:" field contains the value "1" and a dropdown menu showing "Years". The "To:" field contains the value "31,556,925.97" and a dropdown menu showing "Seconds". An equals sign "=" is positioned between the two input fields.

CONVERT.SCX

The following expression is defined for the VALID clause for the **From** GET field:

```
convrt (toval, fromval, "right")
```

The following expression is defined for the VALID clause for the **To** GET field:

```
convrt (toval, fromval, "left")
```

These expressions call CONVRT(), a UDF defined in the cleanup code for the screen. CONVRT() contains the following code:

```
*
* CONVRT - Do the conversion.
*
FUNCTION convrt
PARAMETER m.new, m.old, m.direction
PRIVATE m.toid, m.fromid, m.tounit, m.fromunit

IF (VAL(m.old) = 0 AND m.unitttype="Temperature") OR ;
   (m.old = SPACE(19) AND m.unitttype="Temperature") OR ;
   m.topop = m.frompop
```

Window Definitions

It is not necessary to name windows in the Screen Layout dialog. If you do not specify a window name, a unique name is created during generation. Allowing FoxPro to name the window eliminates the possibility of names colliding with window names in other screens of the application. This unique name changes every time you generate the screen.

Name your windows if you want to reference the windows by name in other procedures or if your application includes context-sensitive help. For information about incorporating context-sensitive help in your applications, see the chapter titled Customizing Help in this manual.

Positioning Windows

By default, all windows are centered on the monitor. You can specify the position of the window in the Screen Layout dialog. These coordinates are saved in the .SCX database.

The **Arrange** push button in the Generate Screen dialog allows you to reposition the windows in a screen set prior to generation. If the screen set is saved in a project, these coordinates are saved in the .PJX database. Arranging windows does not change the coordinates saved in the .SCX database.

Window Types

The following list describes the window types available and suggested uses for each window type:

| | |
|---------------|---|
| User | This window type is used when you want to control window attributes (close, float, zoom, etc.) and window color. |
| System | This window type is used when you want to mimic FoxPro system windows. |
| Dialog | Typically, this window type is specified for windows that are modal in nature. In modal dialogs, information must be completed in the dialog before the dialog can be exited. In the FoxPro interface, the Setup dialog is modal. |
| Alert | This window type is used to display warning and confirmation information. Alerts are typically modal. |

READ Level Clauses

You can define READ level code snippets for the ACTIVATE, VALID, DEACTIVATE, SHOW and WHEN clauses.

The following list is the order of execution for READ events and clauses when the READ is first issued:

- READ level WHEN clause
- ACTIVATE WINDOW
- READ level ACTIVATE
- READ level SHOW¹
- GET level WHEN for the first GET

Following is a list of the order that READ clauses are called when a new window is activated:

- VALID for the field being exited
- DEACTIVATE old window (window name returned by WLAST()²)
- ACTIVATE new window (window name returned by WONTOP())
- READ level DEACTIVATE
- READ level ACTIVATE (if the window brought on top is a READ window)
- WHEN clause for the new field

¹ The SHOW clause is also executed whenever the SHOW GETS command is issued. The SHOW GETS command can be issued in any code snippet in the screen.

² The DEACTIVATE routine is executed when any window is brought forward *and* the deactivating window is a READ window. The DEACTIVATE routine is not executed if the window brought forward is launched from a VALID, WHEN, DEACTIVATE, ACTIVATE or SHOW clause.

READ Level ACTIVATE Example

This example is from CLIENTS.SCX., a screen called in the Client Manager module of the ORGANIZER application. The Client Manager module is an example of displaying Browse windows with screens. This procedure is used to select the CLIENTS database when the Account Details window is active and to prohibit the closing of the Browse windows.

For more information about coordinating Browse windows with screens, see the section titled Coordinating Screens with Browse later in this chapter.

| Client Manager | | | |
|---|--|------------------------------|--|
| Company: Big Masters | | Balance: 2180.10 | |
| Contact: Dorit Springer | | Notes: | |
| Address: 8920 Recsize Drive | | | |
| Fairmont, NJ 07654 | | CTRL+TAB to exit | |
| Area-Phone: 304-428-8807 | | EXT: 2680 | |
| Client Type: (.) Active () Inactive () Prospect | | Cuisine Preference: Japanese | |
| < Help > < New > < Save > < Cancel > < Balance > | | | |

| | | | | | |
|----------|-----------|---------|------------|------------|----------|
| < Next > | < Prior > | < Top > | < Bottom > | < Locate > | << OK >> |
|----------|-----------|---------|------------|------------|----------|

| Client List | | Account Details | | | |
|-----------------------|--|-----------------|------------|--------|---------|
| Company | | Trans_type | Trans_date | Amt | Service |
| Aspen Planning & Inc. | | Billing | 01/02/91 | 622.02 | Memo |
| American Forum | | Expense | 01/06/91 | 125.97 | Memo |

CLIENTS.SCX

```

IF UPPER(WLAST()) = "ACCOUNT DETAILS" OR ;
    UPPER(WLAST()) = "DETAILS.SERVICE"
    SELECT clients
    SHOW GETS
ENDIF

```

If Browse window was the last window on top, reselect the CLIENTS database because activating the Browse window automatically selects the associated database.

```

IF NOT WEXIST("Account")
    SELECT details
    BROWSE LAST NOWAIT NORMAL
    SELECT clients
    SHOW GETS
ENDIF

```

If a Browse window is closed, reopen it.

```

IF NOT WEXIST("Client")
    BROWSE LAST NOWAIT NORMAL
ENDIF

```


READ Level SHOW Example 1

This example is taken from FAMILY.SCX, a screen called in the Family & Friends module of the ORGANIZER application. The SHOW clause is used to create variables for every field in the database and move data from the current record into corresponding variables.

| Family/Friends Manager | | |
|----------------------------|-------------------|--|
| Last Name: Gossnergan | First Name: Ed | Initial: ■ |
| Spouse: Turan | | Birth: 11/10/64 |
| Phone Number: 303-664-6981 | | |
| Address: 321 Strata Ln. | | |
| Boulder | | CO 80303 |
| Notes: <div></div> | | <input checked="" type="checkbox"/> Send Holiday Cards <input type="checkbox"/> Special Diet Needs <input type="checkbox"/> Exchange Gifts |
| CTRL+TAB to exit | | |
| | | < Help > < New > < Save > < Cancel > |

FAMILY.SCX

IF NOT adding ← See if this is a new record.
SCATTER MEMVAR MEMO
ENDIF

Create variables for all fields (including memo fields) and move data from the current record into variables.

Editing takes place on variables, not directly on database fields.

The SHOW clause is executed whenever the SHOW GETS command is issued. The SHOW GETS command can be defined in almost any code snippet in the screen.



Do not place the SHOW GETS command in a SHOW snippet. This will cause FoxPro to make a recursive call and return the error "Insufficient Stack Space."

SHOW GETS redisplayes all GET objects (fields, text, radio or invisible buttons, check boxes, popups, lists and text editing regions). When objects are redisplayed, you can specify whether they are enabled or disabled.

SHOW GETS vs. SHOW GET

All GETS are redisplayed with SHOW GETS. *Individual* objects can be redisplayed with SHOW GET or SHOW OBJECT. SHOW GETS will execute the READ level SHOW routine. SHOW GET and SHOW OBJECT will not.

READ Level SHOW Example 2

This example is taken from CONTROL1.SCX, a utility screen used throughout the ORGANIZER application. The SHOW clause is used to enable and disable buttons depending on the position of the record pointer in the database.

| | | | | | | | | | | | | | | | | | |
|---|------|---|---|-------|---|---|-----|---|---|--------|---|---|--------|---|----|----|----|
| < | Next | > | < | Prior | > | < | Top | > | < | Bottom | > | < | Locate | > | << | OK | >> |
|---|------|---|---|-------|---|---|-----|---|---|--------|---|---|--------|---|----|----|----|

CONTROL1.SCX

```
IF EOF()  
    GO BOTTOM  
ENDIF  
m.saverecno = RECNO()  
GO TOP  
m.toprec = RECNO()  
GO BOTTOM  
m.bottomrec = RECNO()  
GO m.saverecno
```

Save current, top and bottom RECNO() to variables.

```
IF RECNO() = m.bottomrec  
    SHOW GET m.choice, 1 DISABLE  
    SHOW GET m.choice, 2 ENABLE  
    SHOW GET m.choice, 3 ENABLE  
    SHOW GET m.choice, 4 DISABLE  
ELSE
```

Enable/Disable buttons if record pointer is positioned on bottom record.

```
    IF RECNO() = m.toprec  
        SHOW GET m.choice, 1 ENABLE  
        SHOW GET m.choice, 2 DISABLE  
        SHOW GET m.choice, 3 DISABLE  
        SHOW GET m.choice, 4 ENABLE  
    ELSE
```

Enable/Disable buttons if record pointer is positioned on top record.

```
        SHOW GET m.choice ENABLE  
    ENDIF  
ENDIF
```

Enable all buttons if record pointer is on any record except top or bottom record.

READ Level SHOW Example 3

This example is taken from CREDIT.SCX, a screen called in the Credit Cards module of the ORGANIZER application. The SHOW clause is used to SCATTER database fields and create an array for the **Authorized Users** list.

The screenshot shows a window titled "Credit Card Manager". Inside, there are several fields and sections:

- Id:** DC1
- Number:** 7851-7479-7374-4507
- Diner's Club** (in a box)
- Issued By:** Last Federal Savings
- Phone:** 800-067-8627
- Interest** and **Limit** (headers)
- Annual Fee:** 0.00
- Purchase:** 11.00
- Cash Adv:** 12.00
- Expires:** 09/13/92
- Due Date:** / /
- Notes:** (empty box)
- Authorized Users:** (list box containing "Geoffrey Schwartz" and "Nadine Schwartz")
- Balance:** \$-218.39
- Buttons: < Edit Users >, < Balance >, < Help >, < New >, < Save >, < Cancel >, < View Charges >

CREDIT.SCX

```
PRIVATE m.i ← Make this variable PRIVATE.
IF NOT m.adding ← See if this is a new record.
    SCATTER MEMVAR MEMO
    cards = Type ← SCATTER fields to memory variables. (See READ Level SHOW Example 1)

SELECT carduser
COUNT FOR carduser.card_id = m.card_id TO m.usrcnt
IF m.usrcnt <> 0
    DIMENSION users[m.usrcnt,3]
    COPY TO ARRAY users ;
    FIELDS Lastname, Firstname ;
    FOR card_id = m.card_id
    GO TOP

    FOR m.i = 1 TO ALLEN(users,1)
        users[m.i,3] = ALLTRIM(users[m.i,2]) + ;
        " "+ALLTRIM(users[m.i,1])
    ENDFOR
```

If there are users in the database, create an array for the Authorized Users list based on Id number.

```
SELECT credcard
ELSE
  SELECT credcard
  users = ""
  SHOW GET m.user
ENDIF
ELSE
  SHOW GET m.user
ENDIF
```

If there are no card users,
blank the array (users = "")
and refresh the display.

If card users have just been
added, refresh the display.

READ Level WHEN Example 1

This example is from CLIENT.SCX, a screen called in the Client Manager module of the ORGANIZER application. The WHEN routine is executed each time the READ is executed.

In this example, the WHEN routine is used to install the menu system associated with the Client Manager screen. This allows the menu system to be reactivated whenever the user is in a screen with GET fields.

Client Manager

| | |
|---|-------------------------------------|
| Company: Big Masters | Balance: 2180.10 |
| Contact: Dorit Springer | Notes: |
| Address: 8920 Reccize Drive | |
| Fairmont | WU 26554 |
| Area-Phone: 304-428-8807 | EXT: 2680 |
| Client Type: < > Active < > Inactive< > Prospect | Cuisine Preference: Japanese |

CTRL+TAB to exit

< Help > < New > < Save > < Cancel > < Balance >

Navigation

< Next > < Prior > < Top > < Bottom > < Locate > << OK >>

| Client List |
|-----------------------|
| Company |
| Aspen Planning & Inc. |
| American Forum |

| Account Details | | | |
|-----------------|------------|--------|---------|
| Trans_type | Trans_date | Amt | Service |
| Billing | 01/02/91 | 622.02 | Memo |
| Expense | 01/06/91 | 125.97 | Memo |

CLIENTS.SCX

```

DO mainmenu.mpr  ← Install the menu system.
*
* release 'CONVERT' bar.
*
RELEASE BAR 6 OF _MSYSTEM ← Disable the Conversions menu
                             option.

IF VAL(SYS(1001)) < 225000
    SET SKIP OF POPUP reports .T.
    *
    * Labels only if enough memory.
    *
    IF VAL(SYS(1001)) < 213000
        SET SKIP OF BAR 1 OF reports .T.
    ENDIF
ENDIF
ENDIF

```

Check memory and then disable
menus and options accordingly.

READ Level WHEN Example 2

This example is taken from ADDUSERS.SCX, a screen called when you choose the **Edit Users** push button in CREDIT.SCX, a screen used in the Credit Cards module of the ORGANIZER application. The WHEN clause is used to display a message when the user chooses the **Edit Users** push button without entering a card id number in the Credit Card Manager screen.

The screenshot displays the ADDUSERS.SCX screen, which is titled "Credit Card Manager". At the top, there is a menu bar with options: System, Edit, Record, Window, Reports, Card Info. Below the menu bar, a status bar indicates "Blank entries are not allowed". The main area of the screen is divided into two sections: "Selection List:" and "Authorized Users:". The "Selection List:" contains a list of names: Jon Jeager, Arden Schwartz, Bonnie Schwartz, Geoffrey Schwartz, Nadine Schwartz, and Pat Schwartz. The "Authorized Users:" section contains a list of names: Geoffrey Schwartz and Nadine Schwartz. Between these two lists are several buttons: "< Move + >", "< + Remove >", "< Remove All >", and "< New Name >". Below the "Authorized Users:" list, there are input fields for "Last:" and "First:", with a "< Help >" button next to the "Last:" field. At the bottom of the screen, there is a row of buttons: "< Edit Users >", "< Balance >", "< OK >", "< Help >", "< New >", "< Save >", "< Cancel >", and "< View Charges >".

ADDUSERS.SCX

```
IF EMPTY(m.cardid)
    WAIT WINDOW "Blank entries are not allowed" NOWAIT
    m.status = .F.
    RETURN .F.
ENDIF
```

Field Objects and Controls

In a screen, field objects allow you to display and edit data. Controls such as push buttons and check boxes are used to designate, confirm or cancel actions.

Defining field objects and controls is described in the Screen Builder chapter of the FoxPro *User's Guide*.

Every field object and control in a screen can be assigned clauses. WHEN, VALID, and MESSAGE clauses can be assigned to all fields and controls. GET and EDIT fields can also be assigned an ERROR clause.

The following list describes the execution time of each clause:

| | |
|-----------------------|--|
| WHEN Clause | The WHEN clause is executed as you move on to a field or control. In lists, the WHEN clause is executed as you move from item to item in the list. |
| VALID Clause | The VALID clause is executed when the cursor exits a GET or EDIT field and when a control is chosen. |
| MESSAGE Clause | The MESSAGE clause is executed when the cursor is positioned on a GET or EDIT field and when a control is selected. |
| ERROR Clause | The ERROR clause is executed when the cursor exits a GET or EDIT field with invalid data (as defined in a VALID clause). The ERROR clause is available for GET and EDIT fields only. |

This chapter contains examples of the clauses available for field objects and controls and how these clauses can be used.

Field Objects

Choose **Field...** on the **Screen** menu popup to bring forward the Screen Field dialog.

The dialog box is titled "Field:" and contains several sections. At the top, there is a row of three buttons: "< > Say", "<.> Get", and "< > Edit". Below this, there are two rows of controls. The first row has "< Get... >" followed by a text input field, and to the right of the field are the labels "< OK >". The second row has "< Format... >" followed by another text input field, and to the right of the field are the labels "< Cancel >". Below these is a section titled "Range:" which contains two buttons: "[] Upper..." and "[] Lower...". At the bottom of the dialog, there is a grid of nine checkboxes arranged in three rows and three columns. The first row contains "[] When...", "[] Error...", and "[] Scroll bar". The second row contains "[] Valid...", "[] Comment...", and "[] Allow tabs". The third row contains "[] Message...", "[] Disabled", and "[] Refresh".

| | | |
|--|---|-------------------------------------|
| Field: | | |
| <input data-bbox="537 401 645 435" type="button" value=" < > Say "/> <input data-bbox="732 401 826 435" type="button" value=" <.> Get "/> <input data-bbox="913 401 1021 435" type="button" value=" < > Edit "/> | | |
| < Get... > | <input data-bbox="705 469 1034 512" type="text"/> | < OK > |
| < Format... > | <input data-bbox="705 520 1034 563" type="text"/> | < Cancel > |
| Range: | | |
| <input data-bbox="537 614 698 649" type="button" value=" [] Upper... "/> <input data-bbox="866 614 1028 649" type="button" value=" [] Lower... "/> | | |
| <input type="checkbox"/> When... | <input type="checkbox"/> Error... | <input type="checkbox"/> Scroll bar |
| <input type="checkbox"/> Valid... | <input type="checkbox"/> Comment... | <input type="checkbox"/> Allow tabs |
| <input type="checkbox"/> Message... | <input type="checkbox"/> Disabled | <input type="checkbox"/> Refresh |

Screen Field Dialog

When you define a *new* field, the size of the field in the Design window is determined in the following manner:

- If a PICTURE format is defined in the Format text box, the field is the defined width of the picture.
- If no picture is defined, the field in the Design window is the size of the field as defined in the structure of the database unless one of the following is true:
 - If the field is a SAY expression of character type that includes more than one field or variable, the size of the fields is concatenated and the field in the Design window is the total of all the fields in the expression.
 - If the field is a SAY expression of numeric type and includes more than one field or variable, the size of the field in the Design window is the size of the largest numeric field in the expression.
 - If the field is a SAY expression of date type, the field will be eight characters wide (if SET CENTURY is ON the field will be ten characters wide).
 - If the field is a memory variable, its size is determined by its type. Character variables are the length of the defined string, logical are three characters wide and numeric are ten characters wide. Date variables are eight characters wide (if SET CENTURY is ON the date field will be ten characters wide).
 - All fields of memo type are defined as the width specified by SET MEMOWIDTH (the default SET MEMOWIDTH value is 50).
- If the field is not defined in the structure and has no picture clause, the size of the field will default to 10 characters wide.

You can manually change the size of fields after they are placed in the Design window. The size of a field is never automatically changed once it is placed in the Screen Design window.

/@ ... GET/EDIT WHEN Clause Example 1

This example is taken from RESTAURS.SCX, a screen called from the Restaurants module of the ORGANIZER application. The WHEN clause is defined for the Cuisine GET field and is used to execute an escape routine.

When you choose **Other...** in the **Cuisine** popup, code in the VALID clause for the **Cuisine** popup enables the Cuisine GET field.

The escape routine is a user-named procedure defined in the cleanup code for the screen.

Restaurant Manager

Restaurant: A&P Steaks

Speciality: Filet Mignon

Address: 5171 Dorcas Way
Kailua, HI 96734

Cuisine: Rating: Cost:

Other...

II

\$50-100

Maitre'd: Ronald Pecukonis

Phone: 808-340-7002

Notes:

CTRL+TAB to exit

☐ Reservations

☒ Credit Cards

☐ Valet Parking

☒ Handicap Access

☒ Casual Attire

< Help >

< New >

< Edit >

< Cancel >

Order:

Record#

RESTAURS.SCX

ON KEY LABEL esc DO eschandler WITH "newcuis"

Screens

D2-57

/@ ... GET/EDIT WHEN Clause Example 2

This example is taken from RESTAURS.SCX, a screen called from the Restaurants module of the ORGANIZER application. The WHEN clause defined for the State GET field is used to display a list when the cursor enters the field.

The screenshot shows a 'Restaurant Manager' window. It contains several input fields: 'Restaurant:' with 'A&P Steaks', 'Speciality:' with 'Filet Mignon', 'Address:' with '5171 Dorcas Way Kailua', 'Cuisine:' with 'Surf & Tur', and 'Rating:' with 'II'. There is a 'Maitre'd:' field with 'Ronald Pecukonis' and a 'Ph' field with '808-340-7002'. A 'Notes:' section has a text area and a list of checkboxes: 'Reservations', 'Credit Cards', 'Valet Parking', 'Handicap Access', and 'Casual Attire'. A 'State' field has a dropdown menu open, showing a list of states: 'Delaware', 'District of Columbia', 'Florida', 'Georgia', and 'Hawaii'. There are 'Help' and 'Cancel' buttons at the top right, and an 'Order:' field with 'Record#' at the bottom right. A 'CTRL+TAB to exit' message is at the bottom left.

RESTAURS.SCX

```
PRIVATE staflds, m.choice, m.savearea, m.count, m.lastkey

m.lastkey = LASTKEY()
m.savearea = SELECT()

IF NETWORK() AND NOT (m.editing OR m.adding)
    RETURN
ENDIF
IF NOT locatedb("states",0)
    RETURN
ENDIF

COUNT TO m.count
DIMENSION staflds[m.count,2]
COPY TO ARRAY staflds
```

Make variables PRIVATE and initialize variables.

If using FoxPro on a network, prevent the display of the popup, if not editing or adding records.

Call UDF LOCATEDB() (See Setup Code Example 3)

Create and fill array with records for STATES.DBF

```

= ASORT(staflds,2)
DEFI WIND list FROM 7,32 TO 14,52 NONE
ACTI WIND list
IF NOT EMPTY(State)
    m.choice = ASUBSCRIPT(staflds,ASCAN(staflds,State,1),1)
ENDIF
@ 0,0 GET mchoice ;
    FROM staflds ;
    PICTURE '@&T' ;
    RANGE 2 ;
    SIZE 7,20 ;
    DEFAULT 1 ;
    COLOR SCHEME 1
READ
RELE WIND list
USE
SELECT (m.savearea)
REPLACE state WITH staflds[m.choice,1]
SHOW GET state
IF m.lastkey=5 OR m.lastkey=19 OR m.lastkey=15
    *
    * Provide a way to move between the objects with the keyboard.
    *
    _CUROBJ = OBJNUM(m.city)
ELSE
    _CUROBJ = OBJNUM(m.zip)
ENDIF

```

Sort records.

Define and activate window for list.

State stored in field is selected in list.

Define and activate list.

Release window for list.

Store chosen state to database.

Allow user to use keyboard to move between objects.

%@ ... GET/EDIT VALID Clause Example 1

This example is taken from CLIENTS.SCX, a screen called from the Client Manager module of the ORGANIZER application. The VALID clause is defined for the Company GET field and is used to compare values and enable the **Save** push button. The SHOWSAVE() function is located in the cleanup code for the screen.

Client Manager

Company: Big Masters

Balance: 2180.10

Contact: Dorit Springer

Notes:

Address: 8920 Recsize Drive

Fairmont, NJ 26554

CTRL+TAB to exit

Area-Phone: 304-428-8807

EXT: 2680

Cuisine Preference: Japanese

Client Type: (<>) Active (<)> Inactive(<)> Prospect

< Help >

< New >

< Save >

< Cancel >

<Balance>

< Next >

< Prior >

< Top >

< Bottom >

< Locate >

<< OK >>

Client List

| Company |
|-----------------------|
| Aspen Planning & Inc. |
| American Forum |

| Account Details | | | |
|-----------------|------------|--------|---------|
| Trans_type | Trans_date | Amt | Service |
| Billing | 01/02/91 | 622.02 | Memo |
| Expense | 01/06/91 | 125.97 | Memo |

CLIENTS.SCX

```
IF m.company < > clients.company
    = showsave( )
ENDIF
```

%@ ... GET/EDIT VALID Clause Example 2

This example is taken from ADDUSERS.SCX, a screen called when you choose the **Edit Users** push button in CREDIT.SCX, a screen used in the Credit Cards module of the ORGANIZER application. In this example, the VALID clause is defined for the Last GET field and is used to check for data in the field and to enable the First GET field.

The screenshot shows a window titled 'ADDUSERS.SCX'. It contains two main sections: 'Selection list:' and 'Authorized users:'. The 'Selection list:' has a list box with the following names: Jon Jeager, Arden Schuartz, Bonnie Schuartz, Geoffrey Schuartz, Nadine Schuartz, and Pat Schuartz. The 'Authorized users:' has a list box with Geoffrey Schuartz and Nadine Schuartz. Between these two sections are several buttons: '< Move >', '< Remove >', '< Remove All >', '< New name >', '<< OK >>'. Below the 'Authorized users:' list box are two input fields labeled 'Last:' and 'First:'. The 'Last:' field has a '< Help >' button next to it.

ADDUSERS.SCX

```
IF EMPTY(m.userlast)
    RETURN .F.
ENDIF
```

If the field is empty, do not let the user exit until the user enters data or presses Escape.

```
SHOW GET m.userfirst ENABLE
_CUROBJ = OBJNUM(m.userfirst)
```

Enable and select the first GET field.

/@ ... GET/EDIT ERROR Clause Example

This example is taken from ADDUSERS.SCX, a screen called when you choose the **Edit Users** push button in CREDIT.SCX, a screen used in the Credit Cards module of the ORGANIZER application. In this example, the ERROR clause is defined for the Last GET field and is used to display a message if the user tries to tab to another field without entering a last name.

| Selection list: | | Authorized users: |
|--|---|---|
| <div><div>Jon Jeager</div><div>Arden Schuartz</div><div>Bonnie Schuartz</div><div>Geoffrey Schuartz</div><div>Nadine Schuartz</div><div>Pat Schuartz</div></div> | <div>< Move + ></div> <div>< + Remove ></div> <div>< Remove All ></div> <div>< New name ></div> <div><< OK >></div> | <div><div>Geoffrey Schuartz</div><div>Nadine Schuartz</div></div> <div><div>Last: < Help ></div><div></div><div>First:</div><div></div></div> |

ADDUSERS.SCX

"Blank entries are not allowed"

%@ ... SAY Refresh Example

This example is taken from CREDIT.SCX, a screen called from the Credit Cards module of the ORGANIZER application. The SHOW routine is defined for the Balance SAY field and is used to refresh the field when the record changes.

Credit Card Manager

| | | | |
|----------------|------------------------------------|----------------|---------------------|
| Id: DC1 | Number: 7851-7479-7374-4507 | DELETED | Diner's Club |
|----------------|------------------------------------|----------------|---------------------|

Issued By: Last Federal Savings
Phone: 800-067-8627
Annual Fee: 0.00
Expires: 09/13/92
Due Date: / /

Interest: 11.00
Purchase: 11.00
Cash Adv: 12.00
Limit: \$5000.00
\$800.00

Notes:

Authorized Users:

- ▶ Geoffrey Schuartz
- Nadine Schuartz

Balance: \$-218.39

< Edit Users > < Balance >

< Help > < New > < Save > < Cancel > < View Charges >

CREDIT.SCX

```

PRIVATE currwind
STORE WOUTPUT() TO currwind
*
* Show Code from screen: CREDIT
*
#REGION 1
PRIVATE m.i
.
.
.
IF SYS(2016) = "CREDIT" OR SYS(2016) = ""
  ACTIVATE WINDOW credit SAME
  @ 1,24 SAY IIF(DELETED(),"DELETED",SPACE(7)) ;
  SIZE 1,9
ENDIF
  
```

Make this variable PRIVATE and then initialize it.

Generated SHOW routine.

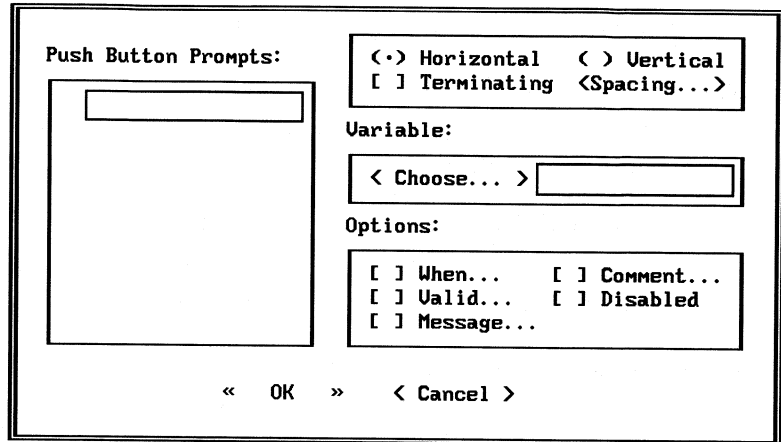
Other Options

- Range** The RANGE clause allows you to specify an upper and lower bound for data in a GET field.
- Comment...** You may assign comments to any object in the Screen Builder. These comments are for reference purposes only and do not affect the generated output in any way.
- Disabled** This option is available for GET and EDIT fields. If this option is checked, you will be unable to access the field upon generation. Data will be displayed, but the cursor will skip the field and editing of the field is prohibited. Code snippets for other objects in the screen can contain commands to enable these objects.
- Scroll Bar** This option is available for EDIT fields only. It is not available for GET and SAY fields. This option places a scroll bar on the right side of all editing regions that are at least two lines deep. After generation, if data for the field is greater than the display area, the scroll bar allows the user to scroll up and down to view and edit all the data in the field.
- Allow Tabs** This option is available for EDIT fields only. It is not available for GET and SAY fields. **Allow Tabs** allows the user to Tab in the EDIT field. To exit the field in the generated screen, the user must press Ctrl+Tab.
- Length...** This option is available for EDIT fields only. A dialog appears allowing you to specify the maximum number of characters the user can enter in the EDIT field.

Push Buttons

Push buttons allow you to get information from the user that typically initiates an action. The user chooses the desired button to perform an action described by the button's prompt.

Choose **Push Button...** on the **Screen** menu popup to bring forward the Push Button dialog.



The Push Button Dialog box is a rectangular window with a double-line border. It contains several sections:

- Push Button Prompts:** A label followed by a large rectangular text area for entering the button's prompt.
- Orientation:** A box containing four options: ☐ Horizontal, ☐ Vertical, ☐ Terminating, and ☐ Spacing... (with an ellipsis).
- Variable:** A label followed by a box containing the text "< Choose... >" and a small rectangular text field for entering a variable name.
- Options:** A box containing six options arranged in three rows: ☐ When..., ☐ Comment..., ☐ Valid..., ☐ Disabled, and ☐ Message... (with an ellipsis).
- Buttons:** At the bottom, centered, are three buttons: "< OK", "> Cancel", and a small ">" button.

Push Button Dialog

Push buttons can be used to invoke another screen or a dialog. You can give the user a visual clue that a push button will bring forward a dialog by placing an ellipsis (...) in the push button prompt.

Group vs. Individual Push Buttons

Push buttons can be defined individually or in groups. Push buttons that perform similar actions (Top, Bottom, Prior, Next) should be defined in a group. A DO CASE statement in the VALID clause can determine which button was selected and take the appropriate action.

Push buttons which perform actions that are not related to any other push buttons should be defined individually.

Terminating vs. Non-Terminating Push Buttons

Push buttons perform actions within a READ and are, by default, non-terminating buttons. When a button is non-terminating, all controls remain active and you can make further selections and enter additional data in the generated screen.

Terminating push buttons execute the VALID (if one is defined), exit the READ and execute the next line in the controlling program.

Default and Escape Push Buttons

Special characters allow you to define default and escape push buttons. For information on defining default and escape push buttons, see the Screen Builder chapter in the *FoxPro User's Guide* or the @ ... GET — Push Buttons command in the *FoxPro Language Reference*.

Push Button VALID Clause Example 1

This example of an individual push button is taken from FAMILY.SCX, a screen called in the Family & Friends module of the ORGANIZER application. The VALID clause is defined for the **Help** push button and is used to bring forward a Help window for the Family & Friends module.

| Family/Friends Manager | | | | |
|------------------------|--|----------|---|----------|
| Last Name: | First Name: | Initial: | < Help > < New > < Save > < Cancel > | |
| Gossnergan | Ed | | | |
| Spouse: | Turan | Birth: | | 11/10/64 |
| Phone Number: | 303-664-6981 | | | |
| Address: | 321 Strata Ln. Boulder, CO 80303 | | | |
| Notes: | <input checked="" type="checkbox"/> Send Holiday Cards <input type="checkbox"/> Special Diet Needs <input type="checkbox"/> Exchange Gifts | | | |
| CTRL+TAB to exit | | | | |

FAMILY.SCX

HELP CHR(254) Family / Friends

Push Button VALID Clause Example 2

This example of group push buttons is taken from CONTROL1.SCX, a utility screen used throughout the ORGANIZER application. The VALID clause is used to enable and disable push buttons depending on the position of the record pointer in the database.



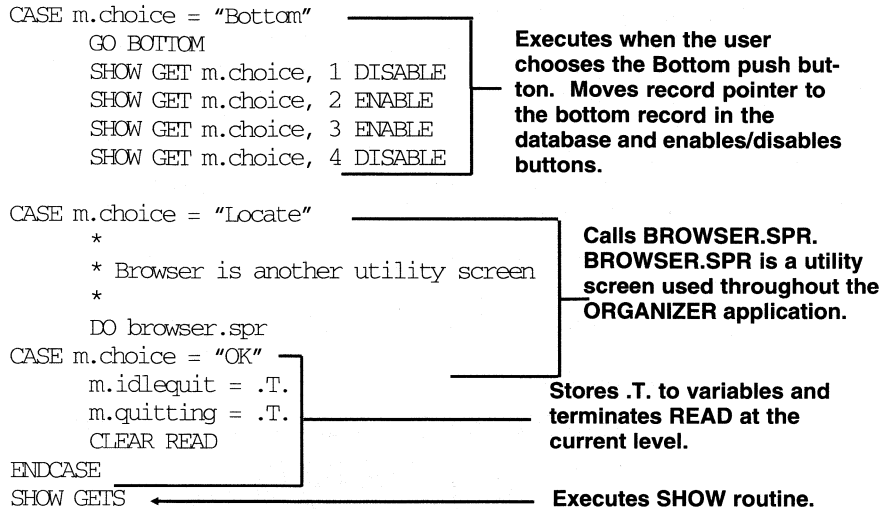
CONTROL1.SCX

```
DO CASE
CASE m.choice = "Next"
    SKIP 1
    IF RECNO() = m.bottomrec
        SHOW GET m.choice, 1 DISABLE
        SHOW GET m.choice, 4 DISABLE
    ELSE
        IF RECNO() > m.toprec
            SHOW GET m.choice, 2 ENABLE
            SHOW GET m.choice, 3 ENABLE
        ENDIF
    ENDIF
CASE m.choice = "Prior"
    SKIP -1
    IF RECNO() = m.toprec
        SHOW GET m.choice, 2 DISABLE
        SHOW GET m.choice, 3 DISABLE
    ELSE
        IF RECNO() < m.bottomrec
            SHOW GET m.choice, 1 ENABLE
            SHOW GET m.choice, 4 ENABLE
        ENDIF
    ENDIF
CASE m.choice = "Top"
    GO TOP
    SHOW GET m.choice, 1 ENABLE
    SHOW GET m.choice, 2 DISABLE
    SHOW GET m.choice, 3 DISABLE
    SHOW GET m.choice, 4 ENABLE
```

Executes when the user chooses the Next push button. Moves record pointer down one record in the database and enables/disables buttons.

Executes when the user chooses the Prior push button. Moves record pointer up one record in the database and enables/disables buttons.

Executes when the user chooses the Top push button. Moves record pointer to the top record in the database and enables/disables buttons.



Variables for push buttons can be of character or numeric type. In this example, the variable `m.choice` is defined as a character variable in the setup code for the screen. This allows you to rearrange the push buttons in the screen without modifying the code snippet.

This technique can also be used for radio buttons and popups.

Push Button WHEN Clause Example

This example is taken from RESTAURS.SCX, a screen called in the Restaurants module of the ORGANIZER application. The Restaurant module is a multi-user application. In a network environment, choosing **Edit** push button locks the record and the **Edit** button changes to **Save**. Choosing the **Save** or **Cancel** push button unlocks the record.

The **Edit** push button is disabled when FoxPro is not being used on a network. You can edit fields directly. The WHEN clause is defined for the **Edit** push button and is used to display a message if the user chooses the **Save** push button without entering a restaurant name.

The screenshot shows a window titled "Restaurant Manager". Inside, there are several input fields and buttons. The fields are: Restaurant (A&P Steaks), Speciality (Filet Mignon), Address (5171 Dorcas Way, Kailua, HI 96734), Cuisine (Surf & Tur), Rating (II), Cost (\$50-100), Maitre'd (Ronald Pecukonis), and Phone (808-340-7002). There is a Notes section with a list of checkboxes: [] Reservations, [X] Credit Cards, [] Valet Parking, [X] Handicap Access, and [X] Casual Attire. On the right side, there are buttons for Help, New, Edit, Cancel, and Record#. The Edit button is highlighted.

RESTAURS.SCX

```
IF NOT (m.adding OR m.editing)
    RETURN .T.
ENDIF
IF EMPTY(restaurant)
    ?? CHR(7)
    WAIT WINDOW "Enter restaurant name" NOWAIT
    _CUROBJ = OBJNUM(restaurant)
    RETURN .F.
ENDIF
```

Disable the button if not adding or editing a record.

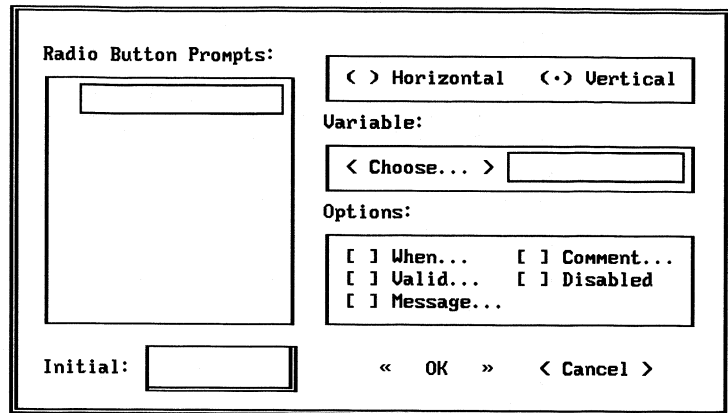
See if the Restaurant field is empty.

Display a message and ring the bell if the user chooses the Save push button without entering a restaurant name.

Radio Buttons

Radio buttons allow the user to choose from a list of mutually exclusive options.

Choose **Radio Button...** on the **Screen** menu popup to bring forward the Radio button dialog.



The dialog box is titled "Radio Button Prompts:". It contains a large rectangular area for a list of prompts, with a smaller rectangular area at the top for a title or label. To the right of this area, there are two radio buttons labeled "< > Horizontal" and "< > Vertical". Below these, there is a section labeled "Variable:" with a button labeled "< Choose... >" and an adjacent text input field. Underneath, there is a section labeled "Options:" containing three rows of checkboxes: "[] When..." and "[] Comment..." on the first row, "[] Valid..." and "[] Disabled" on the second row, and "[] Message..." on the third row. At the bottom left, there is a label "Initial:" followed by a text input field. At the bottom right, there are three buttons: "< OK", "> Cancel", and a ">" button.

Radio Button Dialog

Radio buttons always occur in groups and only one radio button in the group can be selected at any given time. A check box should be used to represent single item options.

Radio Button VALID Clause Example 1

This example is taken from TRANS.SCX, a screen called in the Transactions module of the ORGANIZER application. The VALID clause is used to enable and disable controls in the screen corresponding to the radio button selected.

TRANS.SCX

```
DO CASE
CASE m.decider = 1
    SHOW GET m.cards ENABLE
    SHOW GET m.card_id ENABLE
    SHOW GET m.accnt DISABLE
    SHOW GET m.cleared DISABLE
    SHOW GET m.check_no DISABLE
CASE m.decider = 2
    SHOW GET m.cards DISABLE
    SHOW GET m.card_id DISABLE
    SHOW GET m.accnt ENABLE
    SHOW GET m.cleared ENABLE
    IF NOT (m.trans_type = "Fee" OR m.trans_type = "Interest")
        SHOW GET m.check_no ENABLE
    ENDIF
ENDCASE
IF NOT m.adding
    = showsave()
ENDIF
```

Enable/disable controls when the Credit Cards radio button is selected.

Enable/disable controls when the Accounts radio button is selected.

Enable the Save push button.

Radio Button VALID Clause Example 2

This example is taken from CONVERT.SCX, a screen called in the Conversions module of the ORGANIZER application. The VALID clause is used to fill the arrays for the **From** and **To** popups. The options in the popups correspond to the radio button selected.

CONVERT.SCX

```

PRIVATE m.i, m.size
SELECT DISTINCT units.unit ;
FROM units ;
WHERE units.type = m.unitttype ;
ORDER BY units.type ;
INTO ARRAY fromarray

m.size = ALLEN(fromarray)
DIMENSION toarray[m.size]
FOR m.i = 1 TO m.size
    fromarray[m.i] = ALLTRIM(fromarray[m.i])
    toarray[m.i] = fromarray[m.i]
ENDFOR

m.frompop = fromarray[1]
m.topop   = toarray[1]
m.fromval = SPACE(19)
m.toval   = SPACE(19)

_CUROBJ = OBJNUM(m.fromval)
SHOW GETS

```

Make variables PRIVATE.

Create, sort and fill using an SQL SELECT statement.

Determine the length of array1

Create array2.

Copy elements from array1 to array2.

Initialize variables for popups and GET fields.

Make the From GET field the current object.

Execute SHOW routine.

Initial Popup

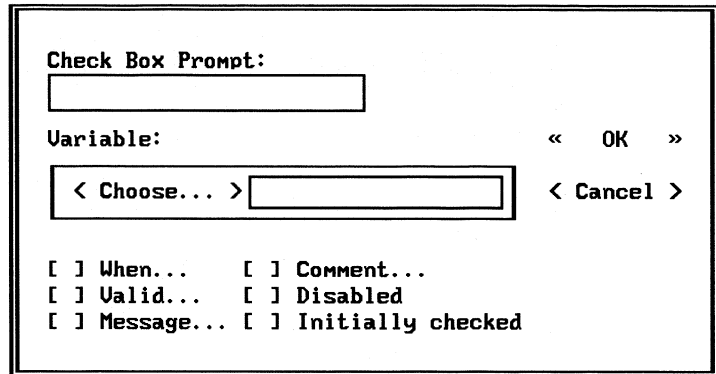
The initial selection is used only for the creation of a variable. By default, the variable is initialized to “1”, the number that corresponds to the first prompt.

If the variable is out of range, no button is selected in the generated screen. Once a button is selected, there is no way to display the buttons with no buttons selected unless the variable value is changed and the GET is refreshed.

Check Boxes

Check boxes act like toggle switches. They are used to indicate a state that is one of two values, such as “on” or “off,” and are frequently used to bring forward a dialog. Check boxes often appear in small groups. Even though they appear as a group, each check box is defined individually.

Choose **Check Box...** on the **Screen** menu popup to bring forward the Check Box dialog.



The image shows a 'Check Box Dialog' box. It has a title bar at the top. Inside, there is a section labeled 'Check Box Prompt:' followed by a text input field. Below that is a section labeled 'Variable:' followed by a text input field. To the right of the 'Variable:' section are two buttons: '< OK >' and '< Cancel >'. Below the 'Variable:' section is a button labeled '< Choose... >' followed by a text input field. At the bottom of the dialog, there are six checkboxes arranged in two columns. The first column contains: '[] When...', '[] Valid...', and '[] Message...'. The second column contains: '[] Comment...', '[] Disabled', and '[] Initially checked'.

| | |
|---|--|
| Check Box Prompt: | |
| <input type="text"/> | |
| Variable: | <input type="text"/> |
| < Choose... > | <input type="text"/> |
| <input type="button" value="OK"/> <input type="button" value="Cancel"/> | |
| <input type="checkbox"/> When... | <input type="checkbox"/> Comment... |
| <input type="checkbox"/> Valid... | <input type="checkbox"/> Disabled |
| <input type="checkbox"/> Message... | <input type="checkbox"/> Initially checked |

Check Box Dialog

Check Box VALID Clause Example 1

This example is taken from TRANS.SCX, a screen called in the Money Manager module of the ORGANIZER application. The VALID clause is defined for the **Deductible** check box and is used to compare values and enable **Save** push button. The SHOWSAVE() function is defined in the cleanup code for the screen.

TRANS.SCX

```
IF details.deductible < > m.deductible
    = showsave()
ENDIF
```

Check to see if value has changed. If so, enable the Save push button.

Check Box VALID Clause Example 2

This example is taken from REPORTS.SCX, a screen called when the user chooses **Reports...** on the **Reports** menu popup in the ORGANIZER application. The VALID clause is defined for the **For...** check box and is used to bring forward the Expression Builder.

REPORTS.SCX

```

IF EMPTY(foexpr)
    GETEXPR "Enter FOR expression:" TO foexpr TYPE 'L'
ELSE
    GETEXPR "Enter FOR expression:" TO foexpr TYPE 'L' DEFAULT
foexpr
ENDIF

```

Display the Expression Builder so the user can enter a FOR expression.

```

m.for = IIF(EMPTY(foexpr), 0, 1)
SHOW GET m.for

```

Check or uncheck the For... check box.

Popups

Popups are used for setting values or choosing from a list of related items. A popup may be defined as either a list popup or an array popup.

Choose **Popup...** on the **Screen** menu popup to bring forward the Popup dialog.

| < > List Popup | | < > Array Popup | | | | | | | |
|--|---|----------------------|---|----------------------------------|-------------------------------------|-----------------------------------|-----------------------------------|-------------------------------------|---|
| <div style="border: 1px solid black; height: 150px; width: 100%;"></div> | | Variable: | <div style="border: 1px solid black; padding: 2px;">< Choose... ></div> | | | | | | |
| | | Options: | <div style="border: 1px solid black; padding: 5px;"> <table> <tr> <td><input type="checkbox"/> When...</td> <td><input type="checkbox"/> Comment...</td> </tr> <tr> <td><input type="checkbox"/> Valid...</td> <td><input type="checkbox"/> Disabled</td> </tr> <tr> <td><input type="checkbox"/> Message...</td> <td><input type="checkbox"/> 1st Element...</td> </tr> <tr> <td></td> <td><input type="checkbox"/> # Elements...</td> </tr> </table> </div> | <input type="checkbox"/> When... | <input type="checkbox"/> Comment... | <input type="checkbox"/> Valid... | <input type="checkbox"/> Disabled | <input type="checkbox"/> Message... | <input type="checkbox"/> 1st Element... |
| <input type="checkbox"/> When... | <input type="checkbox"/> Comment... | | | | | | | | |
| <input type="checkbox"/> Valid... | <input type="checkbox"/> Disabled | | | | | | | | |
| <input type="checkbox"/> Message... | <input type="checkbox"/> 1st Element... | | | | | | | | |
| | <input type="checkbox"/> # Elements... | | | | | | | | |
| Initial: <div style="border: 1px solid black; display: inline-block; width: 100px; height: 20px;"></div> | | « OK » < Cancel > | | | | | | | |

Popup Dialog

With an array popup, you DIMENSION the array in your setup code (described earlier in this chapter). The elements in the array can be defined in a variety of ways. For information on arrays, see the Arrays chapter in this manual.

Popup Example 1

In a list popup, you will define all the items that appear on the popup. These items remain intact every time you use the generated screen.

This example of a list popup is taken from RESTAURS.SCX, a screen called in the Restaurants module of the ORGANIZER application. The VALID clause is defined for the **Order** popup and is used to set the index order of the database.

< > List Popup

- Record#
- Restaurant
- Cuisine
- Rating
- Cost

Initial: **Cost**

< > Array Popup

Variable: **M.setorder**

Options:

- ☐ When...
- ☒ Valid...
- ☐ Comment...
- ☐ Disabled
- ☐ Message...
- ☐ 1st Element...
- ☐ # Elements...

« OK » < Cancel >

Popup Dialog

Restaurant Manager

Restaurant: Arnie's Seafood

Speciality: Orange Almond Trout

Address: 40 Japanese Avenue
Phoenix, AZ 85020

Cuisine: **Seafood** Rating: **I** Cost: **\$ 0-24**

Maitre'd: Phone: 602-314-9485

Notes:

- ☒ Reservations
- ☒ Credit Cards
- ☐ Valet Parking
- ☒ Handicap Access
- ☒ Casual Attire

CTRL+TAB to exit

< Help >

< New >

< Edit >

< Cancel >

0 Record#
Restaurant
Cuisine
Rating
Cost

RESTAURS.SCX

```
DO CASE
CASE m.setorder = 1
    IF NOT EMPTY(ORDER())
        SET ORDER TO
        GO TOP
    ENDIF
CASE m.setorder = 2
    IF LOWER(ORDER()) <> "restaurant"
        SET ORDER TO TAG restaurant
        GO TOP
    ENDIF
CASE m.setorder = 3
    IF LOWER(ORDER()) <> "cuisine"
        SET ORDER TO TAG cuisine
        GO TOP
    ENDIF
CASE m.setorder = 4
    IF LOWER(ORDER()) <> "rating"
        SET ORDER TO TAG rating
        GO TOP
    ENDIF
CASE m.setorder = 5
    IF LOWER(ORDER()) <> "cost"
        SET ORDER TO TAG cost
        GO TOP
    ENDIF
ENDCASE
SHOW GETS
```

Set the index order corresponding to the option chosen and then position the pointer at the top of the file.

Execute SHOW routine.

Popup Example 2

This example is taken from RESTAURS.SCX, a screen called in the Restaurants module of the ORGANIZER application. The VALID clause is defined for the **Cuisine** popup and is used to enable a GET field allowing the user to enter a new choice. The VALID clause for the Cuisine GET field inserts the new value in the array for the popup.

The screenshot shows a 'Restaurant Manager' window. It contains several input fields and a list of options. The 'Cuisine' field has a dropdown menu open, showing 'Other...' as the selected option. The 'Rating' field shows 'II' and the 'Cost' field shows '\$50-100'. The 'Maitre'd' field shows 'Ronald Pecukonis' and the 'Phone' field shows '808-340-7002'. The 'Notes' section has a list of checkboxes: 'Reservations' (unchecked), 'Credit Cards' (checked), 'Valet Parking' (unchecked), 'Handicap Access' (checked), and 'Casual Attire' (checked). The 'Order' field has a 'Record#' button. The 'CTRL+TAB to exit' message is visible at the bottom left.

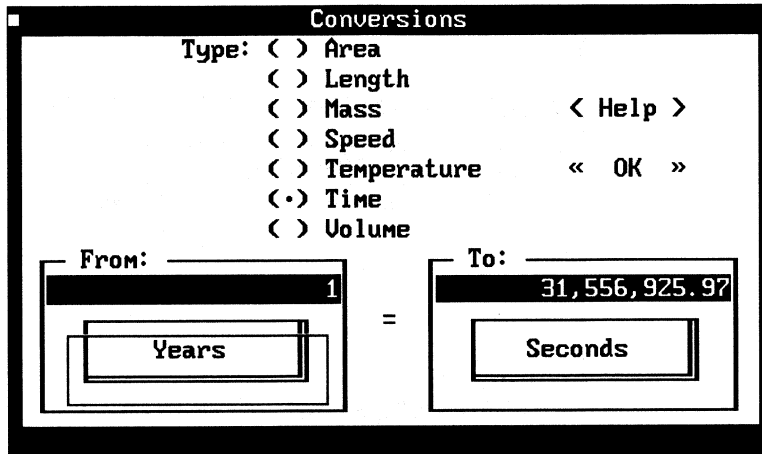
RESTAURS.SCX

```
IF m.cuisine = "Other..."
    popUpEdit = .T.
    SHOW GET m.newcuis ENABLE
    _CUROBJ = OBJNUM(m.newcuis)
ENDIF
```

Enable the GET field when the Other... option is chosen on the Cuisine popup. The GET field is the current object.

Popup Example 3

This example is taken from CONVERT.SCX, a screen called in the Conversion module of the ORGANIZER application. In this example, the SELECT statement is defined in the setup code for the screen.



CONVERT.SCX

Setup Code

```
#SECTION 2
SET UDFPARMS TO REFERENCE
PUSH MENU _MSYSMENU
.
.
.
SELECT DISTINCT units.unit ;
FROM units ;
WHERE ALLTRIM(units.type) = "Area" ;
INTO ARRAY fromarray

= ASORT(fromarray)

m.size = ALLEN(fromarray)
DIMENSION toarray[m.size]
.
.
.
```

Generator directive.

Pass parameters by reference.

Push the menu system onto the stack.

Create and fill an array using a SQL SELECT statement.

Sort the array.

Store the length of the array to a variable.

VALID Clause

```
IF EMPTY(m.fromval)
  _CUROBJ = OBJNUM(m.fromval)
  SHOW GET m.fromval
  RETURN .F.
ENDIF
= convrt (m.fromval, m.toval, "left")
```

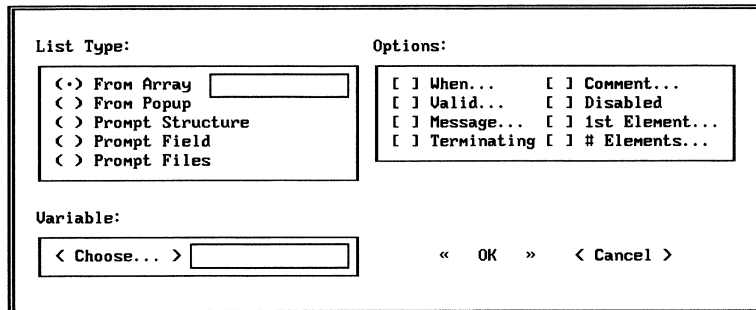
Force the user to enter a value in the From GET field.

Execute UDF CONVRT().
CONVRT() is defined in the cleanup code for the screen.

Lists

Lists are used to display multiple items from which the user may choose an item. If the list contains more items than what will fit in the defined size of the list, a scroll bar appears on the right of the list enabling the user can scroll through a many options.

Choose **List...** on the **Screen** menu popup to bring forward the List dialog.



The List Dialog box is a rectangular window with a double border. It contains three main sections: 'List Type:', 'Options:', and 'Variable:'. The 'List Type:' section has a list of five options: '< > From Array', '< > From Popup', '< > Prompt Structure', '< > Prompt Field', and '< > Prompt Files'. The 'Options:' section has a list of eight options: '[] When...', '[] Comment...', '[] Valid...', '[] Disabled', '[] Message...', '[] 1st Element...', '[] Terminating', and '[] # Elements...'. The 'Variable:' section has a text box with '< Choose... >' and a small rectangular field. At the bottom right, there are three buttons: '< OK >', '< Cancel >', and a small '< >' button.

| List Type: | Options: |
|----------------------|--------------------|
| < > From Array | [] When... |
| < > From Popup | [] Comment... |
| < > Prompt Structure | [] Valid... |
| < > Prompt Field | [] Disabled |
| < > Prompt Files | [] Message... |
| | [] 1st Element... |
| | [] Terminating |
| | [] # Elements... |

Variable:

< Choose... > []

< OK > < Cancel >

List Dialog

Items that are displayed in a list may come from an array, a popup, the structure of a database, the records in a specific field in a database or specific files on a disk.

List Example 1

This example is taken from CREDIT.SCX, a screen called in the Credit Cards module of the ORGANIZER application. The **1st Element** and **#Elements** are defined for the **Authorized Users** list and are used to specify the column in the array to be displayed in the list and to prevent the display of blank records in the list.

The List Dialog box contains the following fields and options:

- List Type:**
 - (>) From Array **users**
 - (>) From Popup
 - (>) Prompt Structure
 - (>) Prompt Field
 - (>) Prompt Files
- Options:**
 - ☐ When... ☐ Comment...
 - ☐ Valid... ☐ Disabled
 - ☐ Message... ☒ 1st Element...
 - ☐ Terminating ☒ # Elements...
- Variable:**
 - < Choose... > **M.user**
- Buttons: « OK » < Cancel >

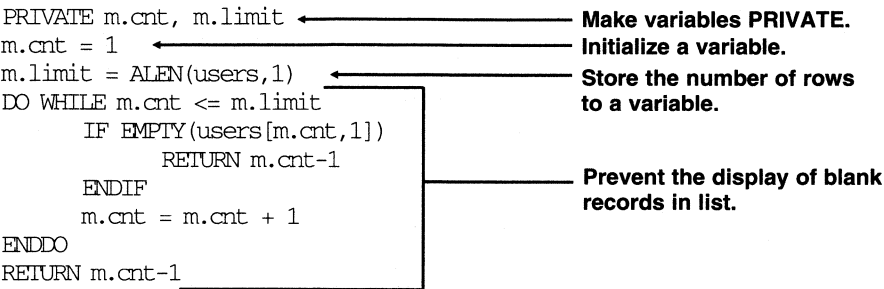
List Dialog

The Credit Card Manager screen displays the following information:

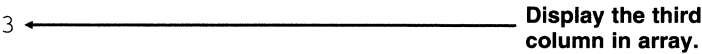
- Id:** DC1
- Number:** 7851-7479-7374-4507
- Diner's Club**
- Issued By:** Last Federal Savings
- Phone:** 800-067-8627
- Interest:**
- Limit:**
- Annual Fee:** 0.00
- Purchase:** 11.00
- Cash Adv:** 12.00
- Expires:** 09/13/92
- Due Date:** / /
- Notes:**
- Authorized Users:**
 - Geoffrey Schuartz
 - Nadine Schuartz
- Balance:** \$-218.39
- Buttons: < Edit Users > < Balance >
- Footer: < Help > < New > < Save > < Cancel > < View Charges >

CREDIT.SCX

#Elements



1st Element



List Example 2

This example is taken from ADDUSERS.SCX, a screen called when you choose the **Edit Users** push button in CREDIT.SCX, a screen used in the Credit Cards module of the ORGANIZER application.

In this example, the VALID clauses are used to move items between two lists.

ADDUSERS.SCX

Selection List VALID

```
IF alreadyin(allusers[m.alluser,3])
    WAIT WINDOW "Duplicate entry" NOWAIT
    RETURN .F.
```

Compare the selection list with the Authorized users list to prevent duplication.

```
ENDIF
```

```
IF m.usrcnt+1 > ALEN(users,1)
    DIMENSION users[m.usrcnt+1,3]
```

Add a row to the array (if necessary).

```
ENDIF
```

```
users[m.usrcnt+1,1] = allusers[m.alluser,1]
users[m.usrcnt+1,2] = allusers[m.alluser,2]
users[m.usrcnt+1,3] = allusers[m.alluser,3]
m.usrcnt = m.usrcnt + 1
m.user = m.usrcnt
```

Copy a selected name to the Authorized users list.

Increment the array row counter.

```
SHOW GET m.mover, 2 ENABLE
IF m.usrcnt > 1
    SHOW GET m.mover, 3 ENABLE
```

Enable the controls.

```
ENDIF
```

```
SHOW GET m.user
```

Display the Authorized users list.

Authorized Users VALID

```
= ADEL(users, m.user)
m.usrcnt = m.usrcnt - 1
m.user = m.usrcnt

    IF m.usrcnt = 0
        SHOW GET m.mover, 2 DISABLE
        SHOW GET m.mover, 3 DISABLE
    ENDIF
SHOW GET m.user
```

← Remove a row from the array.

← Decrement the array row counter.

Disable the controls.

← Display the Authorized users list.

Coordinating Browse with Screens

When a Browse window is displayed with a READ window, the user can activate the Browse window and select a record. The related information in that record can then be displayed in the screen. You can display multiple Browse windows allowing the user to view information from related databases.

The Client Manager module of the ORGANIZER application uses Browse windows to display company names for all records in one Browse window, and transaction information for the current record in another Browse window.

| Client Manager | | | | |
|--|------------------|-------------------------------------|-------------------------|--|
| Company: Big Masters | | | Balance: 2180.10 | |
| Contact: Dorit Springer | | Notes: CTRL+TAB to exit | | |
| Address: 8920 Reccize Drive | | | | |
| Fairmont, WU 26554 | | | | |
| Area-Phone: 304-428-8807 | EXT: 2680 | Cuisine Preference: Japanese | | |
| Client Type: < > Active < > Inactive < > Prospect | | | | |
| < Help > < New > < Save > < Cancel > < Balance > | | | | |

| |
|---|
| < Next > < Prior > < Top > < Bottom > < Locate > « OK » |
|---|

| Client List |
|-----------------------|
| Company |
| Aspen Planning & Inc. |
| American Forum |

| Account Details | | | |
|-----------------|------------|--------|---------|
| Trans_type | Trans_date | Amt | Service |
| Billing | 01/02/91 | 622.02 | Memo |
| Expense | 01/06/91 | 125.97 | Memo |

Browse Windows

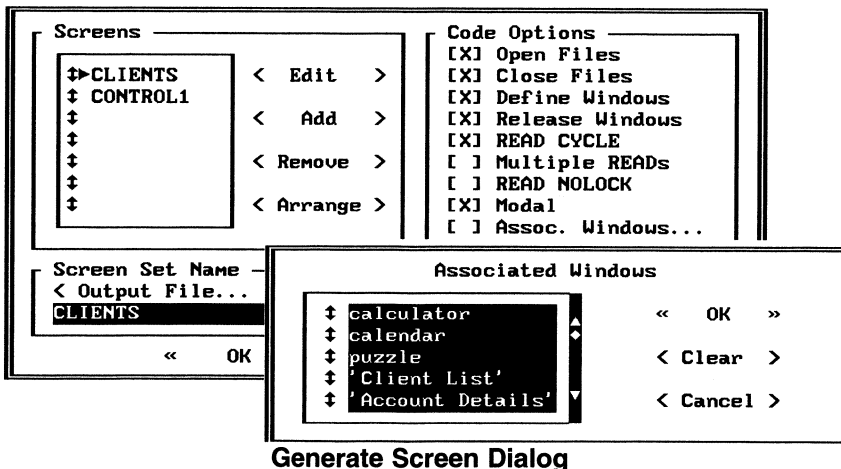
This section describes how these Browse windows are defined and activated, and offers tips on using Browse with screens.

Activating Browse Windows

To combine a Browse window with a READ window, place the commands to define the Browse window in the setup code for the screen. The following commands appear in the setup code for CLIENTS.SCX:

```
IF NOT WISIBLE("Client List")
    BROWSE NORMAL NOWAIT NODELETE LAST TITLE "Client List" ;
    NOAPPEND NOMENU FIELDS company ;
    WHEN showgets()
ENDIF
IF NOT WISIBLE("Account Details")
    SELECT details
    BROWSE NORMAL NOWAIT NODELETE LAST TITLE "Account Details" ;
    NOAPPEND NOEDIT NOMENU ;
    FIELDS ;
    Trans_type:10, ;
    Trans_date:10, ;
    Amt:7, Service
    SELECT clients
ENDIF
```

The Browse windows are allowed to interact with the READ windows by including the <window title> in an Associated Window list. The Associated Window list is part of the WITH clause for the READ command. The list is defined with the **Associated Windows** option in the Generate Screen dialog.



Defining an Associated Window list makes the screen set MODAL and only the windows in the screen set and windows specified in the Associated Window list can be activated. When you define an Associated Window list, the following command is generated in the .SPR program:

```
READ CYCLE MODAL ;
    WHEN _px80shzon() ;
    ACTIVATE _px80shzot() ;
    DEACTIVATE _px80shzoz() ;
    SHOW _px80shzp4() ;
    WITH calculator, calendar, puzzle, 'Client List', ;
        'Account Details', Details
```

The windows in the Associated Window list can be activated before the screen is executed or they can be activated with menu options. Activating windows with menu options is described later in this chapter and in the chapter titled Menus in this manual.

Sizing and Positioning Browse Windows

The first time you run an application that combines a Browse window with a screen, the Browse window opens at the default size and position on the monitor.

You can save the size and position of Browse windows by specifying a resource file for the application and including that resource file in the project. ORGUSER is a resource file created for the Client Manager Application.

The size and position of the Browse windows were saved in this resource file outside of the ORGANIZER application. During development, we set the resource file to ORGUSER and arranged all the windows. These coordinates were saved in the ORGUSER resource file when we exited FoxPro.

The ORGUSER resource file was then included in the CLIENTS project and the following commands were defined in the setup code for CLIENTS.SCX:

```
m.resoset = SET("RESOURCE")
m.oldreso = SET("RESOURCE",1)
SET RESO TO ORGUSER
```

When you include a database in a project, the database is automatically read-only. When the user runs the Client Manager module of the ORGANIZER application, they can move and size the Browse windows, but these new coordinates will *not* be saved in the resource file. Every time the application is executed, the windows will open at the size and position saved in the resource file.

Commands in the cleanup code for CLIENTS.SCX call a procedure in UTILITY.PRG that restores the resource file. This procedure includes the following commands:

```
IF NOT EMPTY(m.oldreso)
    SET RESO TO LOCFILE(m.oldreso, "DEF",;
        "Where is "+m.oldreso+" resource file?")
ENDIF
IF m.hidecomm
    SHOW WINDOW "Command"
ENDIF
IF m.resoset = "OFF"
    SET RESOURCE OFF
ENDIF
```

Activating Menus During a Modal READ

When a modal READ is issued, your menu system is disabled. A modal READ is a READ that includes the MODAL keyword or a WITH <window title list> clause. However, your menu can be reactivated and accessible during the READ by defining a READ WHEN clause.



To make your menu available during a modal READ, execute your menu program in the READ WHEN clause.

The following code is defined in the WHEN clause for the CLIENTS.SCX:

```
DO mainmenu.mpr
```

Debugging Screen Code in an Application

When you run a screen program in the Trace window, what you see is the generated code as it is executed. In order to debug generated code, you must make sure the **Save Generated Code** check box is checked in the Project Options dialog. Options in this dialog allow you to specify where the generated program is saved. For information on the Project Options dialog, see the Project Manager chapter in the *FoxPro User's Guide*.

If you receive errors while running a generated program, suspend or cancel the program and note the location in the generated program where the error occurred. Return to the screen that generated the error and make your changes in the appropriate code snippet.

Generated programs (generated by GENSCRN) are *extremely* well documented. All code snippets are labeled with their unique name (provided by the generator), the screen, the READ or object level clause and the object type with which the code snippet is associated.

```

*          *****
*      * _PX80QJ5D3          m.frompop VALID          *
*      *                                                              *
*      * Function Origin:                                          *
*      *                                                              *
*      * From Screen:      CONVERT,      Record Number:   9      *
*      * Variable:         m.frompop                               *
*      * Called By:        VALID Clause                           *
*      * Object Type:      Popup                                   *
*      * Snippet Number:   1                                       *
*      *****
*
FUNCTION _px80oj5d3      && m.frompop VALID
#REGION 1
IF EMPTY(m.fromval)
    _CUROBJ = OBJNUM(m.fromval)
    SHOW GET m.fromval
    RETURN .F.
ENDIF
= convrt (m.fromval, m.toval, "left")
.
.
.

```



Never make your changes in the generated code. Your *screen* is the source for the application.

If you make changes in the generated code, when the screen is regenerated or a project that includes the screen is rebuilt, all changes made to the generated program are overwritten by new code.

Generated names change every time you regenerate a screen and should never be referenced in a program.

Your screen is the source for the application. Generated code is an intermediate step and should be used for debugging purposes only. Generated code should never be edited. Make all changes with the Screen Builder.

Using FoxDoc with Screen Programs

FoxDoc is an automatic application documenter for FoxPro programs. With FoxDoc, documenting an application becomes a simple matter of entering some basic information and pressing a few keys.

Even though generated programs are *extremely* well documented (with function origin comments), FoxDoc can make them even better. FoxDoc will perform the following documentation tasks on a generated program:

- Cross-reference variables.
- Identify all files used in program.
- Construct tree diagram of generated program.
- Include generated program in system-wide statistics.

Also, when FoxDoc documents snippets in a generated screen or menu program, the snippet is described with the variable name with which it is associated.

For information on using FoxDoc, see the chapter titled Documenting Applications with FoxDoc in this manual.

3 Menus

The Menu Builder is a tool for creating menus. In the Menu Builder's Menu Design window, you create the menu pads that appear across the top of your screen and the menu popups that appear below the menu pads.

The Menu Builder eliminates the most tedious step in creating menus — writing the menu program code. After you design your menu, FoxPro generates the corresponding menu program code for you. Then you can use the Project Manager to integrate the menu program into an executable FoxPro application.

This chapter illustrates some of the menu program code created by the Menu Builder and describes some of the circumstances in which you might want to write your own code. To understand what you read in the chapter, you should understand the FoxPro interface and the Menu Builder. If you aren't familiar with these, consult the Interface Basics and Menu Builder chapters in the *FoxPro User's Guide*.

The menus illustrated in this chapter are from the ORGANIZER sample application included with FoxPro. The menu files are CONVMENU.MNX, ORGANIZE.MNX and MAINMENU.MNX, which are in the MENUS subdirectory of the SAMPLE directory. In the Menu Design window you can open and examine these menu files and generate and modify their menu program code.

Advantages of the Menu Builder

Organization and Clarity

The Menu Builder unifies menu definition code and menu procedures. When you create a menu with the Menu Builder, the procedures that FoxPro executes when you choose a menu option are integrated into the menu definition code.

Increased Productivity

Designing a menu interactively with the Menu Builder is much faster than writing a program to create the same menu.

Without leaving the Menu Design window, you can test the appearance and operation of a menu you're designing by choosing the **Try It** option. If you're not satisfied with the menu, you can continue modifying it.

After you design your menu, FoxPro can generate the corresponding menu program code for you.

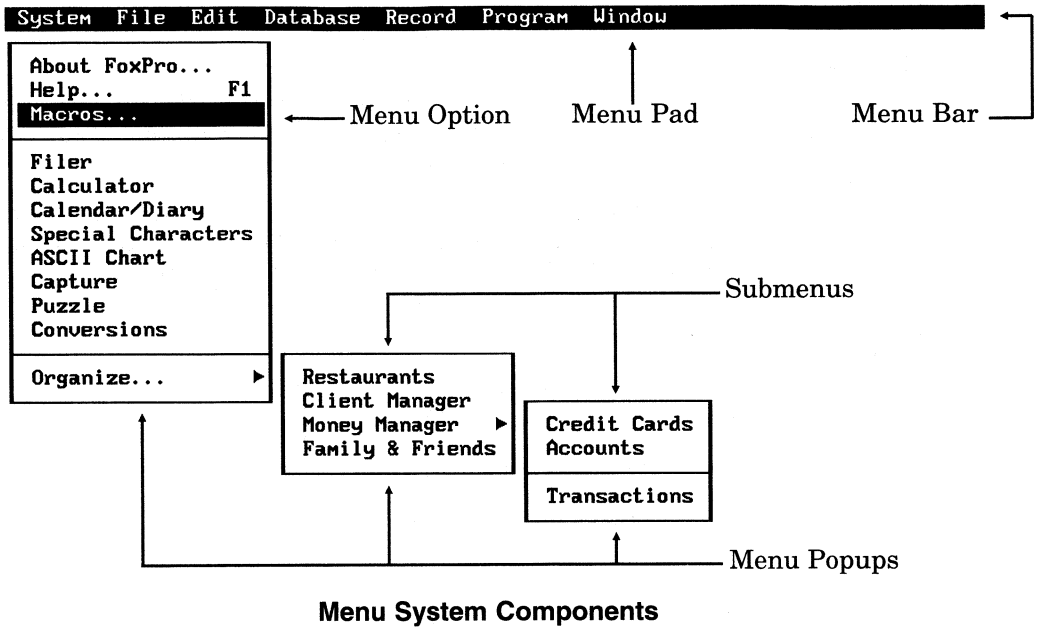
FoxPro System Menu Bar

The Menu Builder uses the FoxPro system menu bar, which has the following advantages:

- FoxPro automatically activates and deactivates the system menu bar in your application as necessary. Whenever your application waits for keyboard input, such as during a READ operation, the system menu bar is activated.
- You can include options from FoxPro menu popups on your popups, and these options are automatically enabled and disabled as needed.
- You don't have to explicitly activate your menu using the `ACTIVATE MENU` command.

Menu Terms Used in This Chapter

A menu consists of the following parts: menu bar, menu pads, menu popups and menu options.



Menu Bar

The *menu bar* appears at the top of the screen and contains the menu pads.

Menus created with the Menu Builder automatically use the FoxPro system menu bar, `_MSYSMENU`.

Menu Pads

The names on the menu bar are *menu pads*. When you choose a menu pad, a menu popup appears below the menu pad, or a procedure or command executes.

With the Menu Builder, you create menu pads in the Menu Design window.

Menu Popups

Menu popups contain a set of related options. You can choose an option from a menu popup to perform an action.

To create menu popups for a menu pad, choose the **Submenu** option from the **Result** popup in the Menu Design window.

Menu Options

A menu popup contains a set of related options. When you choose an option from a menu popup, a procedure or command executes or another menu popup, called a *submenu*, appears.

To create options for a menu popup, choose the **Create** push button to the right of the **Result** popup. The **Create** push button appears when you choose the **Submenu** option from the **Result** popup.

File Extensions

Here are the file extensions used for FoxPro menu databases and menu programs:

- .MNX – menu database
- .MNT – menu database memo file
- .MPR – menu program
- .MPX – compiled menu program

Creating Code Snippets

When you design a menu with the Menu Builder, you can create code snippets that are assigned to menu pads, popups or options. A *code snippet* is a procedure or expression associated with a specific menu pad, menu popup or menu option.

When a code snippet is only one line long, it is included with the command that executes the snippet. Typically, this command is ON SELECTION PAD or ON SELECTION BAR. The command and its code snippet are part of the menu definition code.

If a code snippet is longer than one line, the snippet becomes a separate procedure, with a unique name created by the menu code generator. This procedure is placed at the end of the generated menu program, and the command that executes the snippet calls it by its generated name.



Never reference procedures by their generated names, because the names change each time you regenerate a menu program.

In the generated menu code, comment boxes precede procedures and document the procedures. Comment boxes provide debugging information and information for FoxDoc, the program documentor included with FoxPro.

A comment box includes the following information:

- The procedure name
- The command that calls the procedure
- The number of the .MNX database record that contains the procedure
- The menu pad or popup option prompt that calls the procedure
- The number of the code snippet

Following is the code snippet for the **Clock** option on the **System** menu popup in the MAINMENU file. Because the code snippet is longer than one line, the Menu Builder creates a procedure for it. The procedure has the generated name `_PV00WTOVO`, and the comment box provides information about the procedure.

```

MAINMENU Clock Procedure
IF MRKBAR("environmen",BAR())
  SET CLOCK OFF
  SET MARK OF BAR BAR() OF environmen TO .F.
ELSE
  SET CLOCK ON
  SET MARK OF BAR BAR() OF environmen TO .T.
ENDIF
  
```

**Comment
box.**

```

* *****
* *
* * _PV00WTOVO ON SELECTION BAR 1 OF POPUP environmen *
* *
* * Procedure Origin: *
* *
* * From Menu: MAINMENU.MPR, Record: 12 *
* * Called By: ON SELECTION BAR 1 OF POPUP environmen *
* * Prompt: Clock *
* * Snippet: 1 *
* *
* *****
*
  
```

← **Generated procedure name.**

```

PROCEDURE _pv00wtovo
IF MRKBAR("environmen",BAR( ))
  SET CLOCK OFF
  SET MARK OF BAR BAR( ) OF environmen TO .F.
ELSE
  SET CLOCK ON
  SET MARK OF BAR BAR( ) OF environmen TO .T.
ENDIF
  
```

**Multiple-line
code snippet.**

Calling a Menu Program

To call a menu program, issue a command with the following syntax:

```
DO <menu name>.MPR
```

You must include the .MPR extension because other types of executable files might have the same name.

Activating the Menu

Menus created with the Menu Builder automatically use the FoxPro menu system. If you create a menu that uses the FoxPro menu system, you don't have to explicitly activate your menu using `ACTIVATE MENU`.

READ and Menus

The `READ` command activates controls in FoxPro screens. When a `READ` command has activated a control, your menu might or might not be accessible, depending upon the type of `READ` command issued.

When you issue a modal `READ` command, your menu is disabled. A modal `READ` command is a `READ` command that includes the `MODAL` keyword or a `WITH <window title list>` clause. However, you can reactivate your menu and enable it during a `READ` by including a `WHEN` clause with the `READ` command.

When you issue a `READ` command, access to your menu also depends on the `SET SYSMENU` command you use, as discussed in the following section.

SET SYSMENU

SET SYSMENU is a powerful command for manipulating menus that use the FoxPro menu system. Using SET SYSMENU, you can disable your menus, add and remove items from menus, restore the default FoxPro menus, and control access to your menus during program execution. Here are some of the forms SET SYSMENU can take:

- SET SYSMENU ON – The menu bar is enabled during program execution of commands such as BROWSE, MODIFY MEMO, and non-modal READ, while FoxPro waits for keyboard input. The menu bar doesn't appear, but you can display it by pressing Alt or F10 or by double clicking the right mouse button.
- SET SYSMENU OFF – The menu bar isn't enabled during program execution.
- SET SYSMENU AUTOMATIC – The menu bar appears at all times during program execution and is enabled during program execution while FoxPro waits for keyboard input.
- SET SYSMENU TO DEFAULT – The default FoxPro menu system is restored.

For more information about SET SYSMENU, see the *FoxPro Language Reference*.

PUSH MENU and POP MENU

The PUSH MENU and POP MENU commands help you save and restore menus. Using these commands, you can push a menu onto a stack in memory and then restore it later by popping it off the stack.

Pushing a menu onto the stack saves the menu's current state but doesn't remove the menu from the screen. While the menu is in memory, you can change or replace the menu on the screen. After changing or replacing the menu, you can restore the original menu from the stack by issuing the command POP MENU.

Menus are pushed onto and popped off the stack in a "last in, first out" order. The number of menus you save in memory is limited only by the amount of memory available.

The ORGANIZER application illustrates how to replace and restore menus. When you first run ORGANIZER, it replaces the FoxPro **System** menu popup with its own. When you choose an option from the ORGANIZER popup, a screen program runs. This screen program pushes the current menu onto a stack in memory and then replaces the displayed menu with a new one. When the screen program exits, FoxPro restores the original ORGANIZER menu from memory.

For example, the following commands in the CONVERT.SCX screen program use the PUSH command to push the ORGANIZER menu onto a stack in memory, replace the ORGANIZER menu with its own menu and then restore the original ORGANIZER menu when the screen program exits.

```
PUSH MENU _MSYSMENU  ← Save the current menu to memory.
.
.
.
DO convmenu.mpr      ← Display and activate the CONVMENU menu.
.
.
.
POP MENU _MSYSMENU   ← Restore the previous menu from memory.
```

For more information about PUSH MENU and POP MENU, see the *FoxPro Language Reference*.

Your Working Environment

The following sections include some suggestions for making menu design faster and easier.

Using an Extended Video Mode

If your video hardware supports an extended display mode, use it. If you design menus in an extended video mode, you can display more than 25 screen lines simultaneously, including the Menu Design window and several code snippet windows, making cutting and pasting between code snippets windows easier.

Manipulating Code Snippet Windows

You can size, minimize and dock code snippet windows as you do other FoxPro windows.

When you save your menu, you also save the state of the code snippet windows. When you reopen the menu, the code snippet windows look as they did when you last saved the menu.

Using Quick Menu

If your menu design includes the FoxPro system menus, you can create the menu quickly and easily using the FoxPro Quick Menu feature. To create a quick menu, choose the **Quick Menu** option from the **Menu** menu popup. The **Quick Menu** option is available only when the Menu Design window is empty (that is, when it contains no menu pads, popups or options). For more information about quick menus, refer to the Menu Builder chapter in the FoxPro *User's Guide*.

Restoring the Menu System and Command Window

When you are testing a menu that replaces the FoxPro menu system, you might not be able to access the menu system and the Command window. To restore access to the Command window and the default FoxPro menu system, issue the following command before testing your menu:

```
ON KEY LABEL ALT+F9 SET SYSMENU TO DEFAULT
```

After issuing the command, simply press Alt+F9 to restore the default system menus and the Command window. You can specify any valid key or key combination with the ON KEY LABEL command, not just Alt+F9.

Design Considerations

Your application determines your menu design and options. The following sections include suggestions for designing your menus so they are user friendly.

Emulating the FoxPro Interface

If your application's interface emulates the FoxPro interface, your application will be instantly familiar to users who have experience with this type of interface. For example, because the FoxBASE+® for the Macintosh® interface is similar to the FoxPro interface, a FoxBASE+ for the Macintosh user could quickly learn your application if it looked like the FoxPro interface.

Executing Rarely Used Routines from Menus

If you anticipate that a particular routine won't be used often, design your menu so that the routine is executed through a menu option instead of a control. Placing an option for an infrequently executed routine on a menu reduces clutter on your screens.

Avoid assigning keyboard shortcuts to options associated with routines that make irreversible changes. Avoiding such shortcuts prevents users from choosing the options accidentally.

Using FoxPro Menu Options

You can place most of the menu options available on FoxPro system menus on your menus. When you include the name of a FoxPro option on your menu, the option is available on your menu.

While using the Menu Builder, you can place a FoxPro menu option on your popup by choosing the **Bar #** option from the **Result** popup and then typing the name of the FoxPro option in the text box.

When your menu includes FoxPro menu options, FoxPro automatically manages the enabling and disabling of the options. For example, if you include the **Paste** option on your popup, **Paste** is enabled only when the cursor is in a text editing region and the clipboard is not empty.

The names of the FoxPro system menu options you can use are:

- Described in the topic Menu – Systems Menu Names of the online help facility or the section Menu – Systems Menu Names in the *FoxPro Language Reference*.
- Returned by the SYS(2013) function. SYS(2013) returns a space-delimited character string containing the names of the system menu bar, the menu pads on the system menu bar, the system menu popups and the menu options.

The screenshot shows the FoxPro Menu Builder interface. On the left, a menu structure is displayed with items: Help (F1), Calculator, Calendar/Diary, Puzzle, Conversions, Environment, and OK. The 'Environment' item is selected, and its details are shown in the main window. The main window has three columns: 'pt', 'Result', and 'Options'. The 'Result' column lists the FoxPro system menu option names for each item. The 'Options' column shows the available options for each item, including checkboxes and buttons like 'Try it', 'Item', 'Insert', and 'Delete'.

| pt | Result | Options |
|------------------|-------------------------|---------|
| ‡ \<Help | Bar # _MST_HELP | [X] |
| ‡ \< | Bar # _MST_SP100 | [] |
| ‡ \<Calculator | Bar # _MST_CALCUL | [] |
| ‡ Calendar\<Diar | Bar # _MST_DIARY | [] |
| ‡ Pu\<zle | Bar # _MST_PUZZL | [] |
| ‡ Co\<nversions | Command DO mhit IN MAIN | [X] |
| ‡ \<Environment | Submenu < Edit > | [X] |
| ‡ \< | Command | [] |
| ‡ \<OK | Command DO mexit IN MAI | [X] |

Buttons on the right: < Try it >, Item, < Insert >, < Delete >

A Menu Using Some FoxPro System Menu Options

Choosing Options from the Result Popup

Using the **Result** popup in the Menu Design window, you can create a submenu for a menu option, include a FoxPro system option in your menu, or specify a command or procedure that you want to execute when an option is chosen.

Command

Choose **Command** from the **Result** popup to specify a command that you want to execute when an option is chosen. When you enter a command to execute, FoxPro adds the following command to the menu definition code:

```
ON SELECTION BAR <option number> OF <menu name> <command>
```

Each option on a popup has a unique <option number>.

Bar

Choose **Bar #** from the **Result** popup to put a FoxPro system option on your menu (see the example on the previous page).

Submenu

Choose **Submenu** from the **Result** popup to create a submenu that appears when an option is chosen. When you create a hierarchical popup, FoxPro adds the following command to the menu definition code:

```
ON SELECTION BAR <option number> OF <menu name> ;  
    ACTIVATE POPUP <popup name>
```

Procedure

Choose **Proc.** from the **Result** popup to create a procedure that executes when an option is chosen. When you create the procedure, FoxPro adds the following command to the menu definition code:

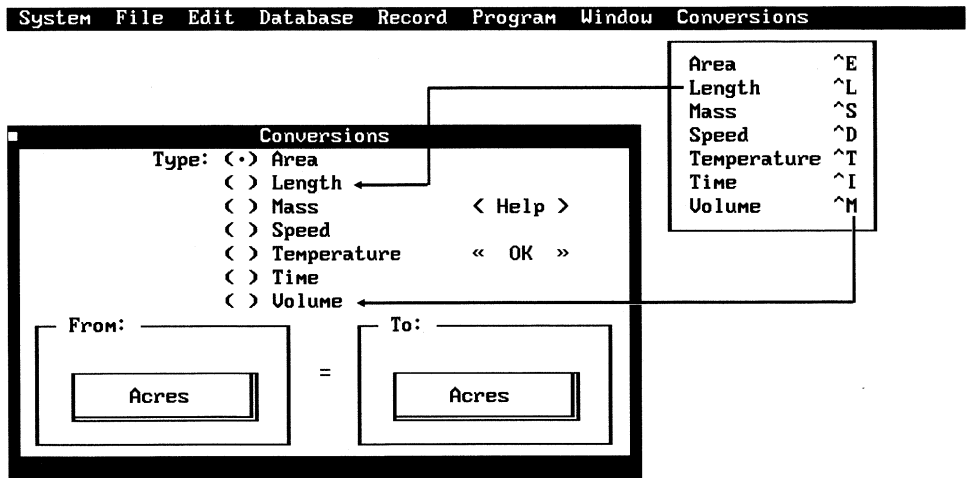
```
ON SELECTION BAR <option number> OF <menu name> ;  
    DO <procedure name> IN <menu name>
```


In the command syntax, <procedure name> is a unique name generated by the menu code generator, and <menu name> is the name of the generated menu program.

Using Keyboard Shortcuts for Screen Controls

To help keyboard users, include menu options for controls that are frequently used and create keyboard shortcuts for these options.

For example, the CONVERT.SCX screen has a set of radio buttons for selecting the type of quantity being measured, such as area or length. You can choose the appropriate radio button by pressing the corresponding key combination shown on the **Conversions** menu popup.



Keyboard Shortcuts for Screen Controls

About the Generated Program

GENMENU, the FoxPro menu program generator, creates a menu program from the information in the menu's .MNX database. The resulting program has an .MPR extension.

Menu program code consists of four parts which FoxPro generates and executes in the following order:

1. Setup Code – Initializes variables used by the menu, saves the current environment and creates an environment for the menu. You create the code snippets placed in the setup code.
2. Menu Definition Code – Creates the menu pads, popups and options. Menu definition code also includes commands, such as ON SELECTION PAD and ON SELECTION BAR, that specify the procedure or command that executes when a menu pad or option is chosen. The menu generator automatically creates the commands in this code section.
3. Cleanup Code – Typically turns mark characters on or off and enables or disables menu pads and options. You create the code snippets placed in the cleanup code.
4. Procedures – These are code snippets you create that execute when a menu pad or popup is chosen.

The following example contains sections from the generated code for the MAINMENU menu in the ORGANIZER application. This example illustrates how the generated code is organized.

```
* *****
*
* * 12/08/92          MAINMENU.MPR          18:30:39 *
* *
* *****
*
* * Author's Name
*
* * Copyright (c) 1992 Company Name
* * Address
* * City,      Zip
*
* * Description:
* * This program was automatically generated by GENMENU.
*
* *****
```

Program Header.
FoxPro always
generates this
information.

About the Generated Program

```

*          *****
*          *
*          *                      Menu Definition          *
*          *
*          *****
*

```

SET SYSMENU TO  **Remove the FoxPro menu system.**

SET SYSMENU AUTOMATIC  **Always display the system menu bar.**

```

DEFINE PAD _px913c77x OF _MSYSMENU PROMPT "\<System" ;
    COLOR SCHEME 3          KEY ALT+S, ""
DEFINE PAD _px913c78u OF _MSYSMENU PROMPT "\<Edit" ;
    COLOR SCHEME 3          KEY ALT+E, ""
DEFINE PAD _px913c79j OF _MSYSMENU PROMPT "\<Record" ;
    COLOR SCHEME 3          KEY ALT+R, ""
DEFINE PAD _px913c7a8 OF _MSYSMENU PROMPT "\<Window" ;
    COLOR SCHEME 3          KEY ALT+W, ""
DEFINE PAD _px913c7at OF _MSYSMENU PROMPT "Re\<ports" ;
    COLOR SCHEME 3          KEY ALT+P, ""

```

Create the menu pads on the system menu bar.

```

ON PAD _px913c77x OF _MSYSMENU ACTIVATE POPUP _msystem
ON PAD _px913c78u OF _MSYSMENU ACTIVATE POPUP _medit
ON PAD _px913c79j OF _MSYSMENU ACTIVATE POPUP _mrecord
ON PAD _px913c7a8 OF _MSYSMENU ACTIVATE POPUP _mwindow
ON PAD _px913c7at OF _MSYSMENU ACTIVATE POPUP reports

```

Activate the popups when the pads are chosen.

```

DEFINE POPUP _msystem MARGIN RELATIVE SHADOW COLOR SCHEME 4

```

Create the System popup that is activated when the System pad is chosen.

```

DEFINE BAR _MST_HELP OF _msystem PROMPT "\<Help" ;
    KEY F1, "F1"
DEFINE BAR _MST_SP100 OF _msystem PROMPT "\-"
DEFINE BAR _MST_CALCUL OF _msystem PROMPT "\<Calculator"
DEFINE BAR _MST_DIARY OF _msystem PROMPT "Calendar/\<Diary"
DEFINE BAR _MST_FUZZL OF _msystem PROMPT "Pu\<zze"
DEFINE BAR 6 OF _msystem PROMPT "Co\<nversions" ;
    KEY ALT+N, ""
DEFINE BAR 7 OF _msystem PROMPT "\<Environment" ;
    KEY ALT+E, ""
DEFINE BAR 8 OF _msystem PROMPT "\-"
DEFINE BAR 9 OF _msystem PROMPT "\<OK" KEY ALT+O, ""

```

Create the options on the System popup.

```

ON SELECTION BAR 6 OF _msystem ;
    DO mhit IN MAINMENU.MPR WITH 'convert.app'
ON BAR 7 OF _msystem ACTIVATE POPUP environmen

```

Execute procedures and popups when these options are chosen.

ON SELECTION BAR 9 OF _msystem DO mexit IN MAINMENU.MPR

```
* *****
* *
* * Cleanup Code & Procedures *
* *
* *****
*
```

```
FOR i = 1 TO cntbar('environmen')
DO CASE
CASE PRMBAR('environmen',i) = 'Clock'
SET MARK OF BAR i OF environmen TO SET('CLOCK') = 'ON'
CASE PRMBAR('environmen',i) = 'Extended Video'
SET MARK OF BAR i OF environmen TO SROW( ) > 25
CASE PRMBAR('environmen',i) = 'Sticky'
SET MARK OF BAR i OF environmen TO SET('STICKY') = 'ON'
CASE PRMBAR('environmen',i) = 'Status Bar'
SET MARK OF BAR i OF environmen TO SET('STATUS') = 'ON'
ENDCASE
ENDFOR
```

Cleanup code.
This code is created in the General Options dialog and is executed after the menu is created and activated.

```
* *****
* *
* * _PV215RLQE ON SELECTION BAR 1 OF POPUP environmen *
* *
* * Procedure Origin: *
* *
* * From Menu: MAINMENU.MPR, Record: 13 *
* * Called By: ON SELECTION BAR 1 OF POPUP environmen *
* * Prompt: Clock *
* * Snippet: 1 *
* *
```

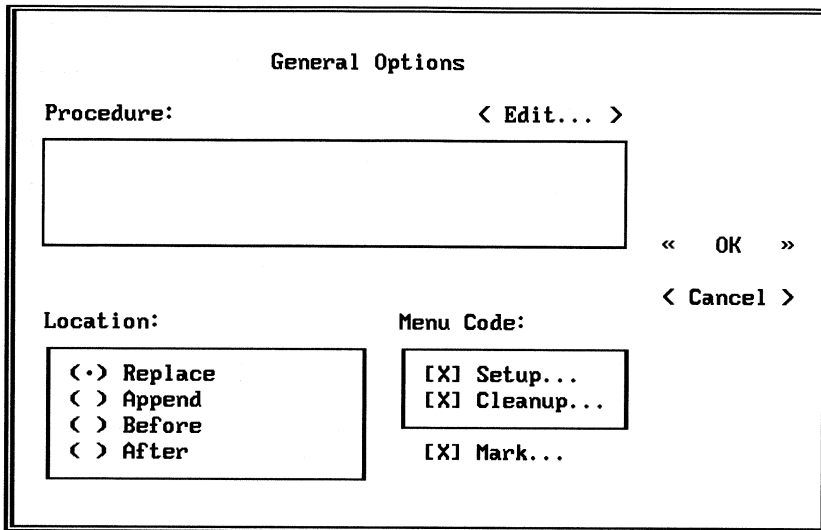
```
* *****
*
PROCEDURE _pv215rlqe

IF MRKBAR("environmen",BAR( ))
SET CLOCK OFF
SET MARK OF BAR BAR( ) OF environmen TO .F.
ELSE
SET CLOCK ON
SET MARK OF BAR BAR( ) OF environmen TO .T.
ENDIF
```

Procedure.
This code is executed when the Clock option of the Environment popup is chosen. The code turns the clock on or off and sets the mark character.

General Options...

When the Menu Design window is open and you choose **General Options...** from the **Menu** menu popup, the General Options dialog appears.



The dialog box is titled "General Options". It contains the following elements:

- Procedure:** A text field with the button "< Edit... >" to its right.
- Location:** A list box with four options:
(<.) Replace
(>) Append
(>) Before
(>) After
- Menu Code:** A list box with four options:
[X] Setup...
[X] Cleanup...
[X] Mark...
- Buttons:** "« OK »" and "< Cancel >" are located on the right side of the dialog.

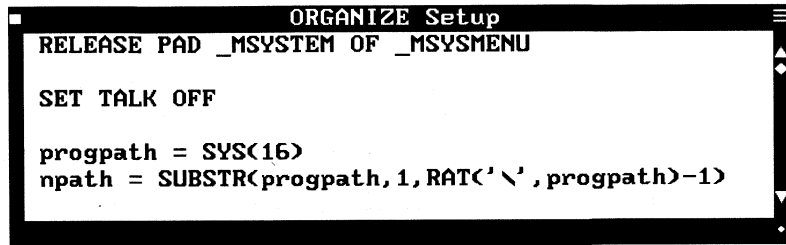
General Options Dialog

In the General Options dialog, you can:

- Create setup code, which is executed before other menu procedures.
- Create a general procedure that is executed for each menu pad.
- Create cleanup code, which is generated and executed after the setup and menu definition code but before other menu procedures.
- Specify the location of menu pads on the menu bar.
- Specify a mark character for menu pads.

Setup Code

To view or create setup code for a menu, choose **Setup...** in the General Options dialog, and then choose **OK**. For example, to view the ORGANIZE.MNX menu file used by the ORGANIZER application, open the file, choose **Setup...** in the General Options dialog and then choose **OK**.



```

ORGANIZE Setup
RELEASE PAD _MSYSTEM OF _MSYSMENU

SET TALK OFF

progpah = SYS(16)
npath = SUBSTR(progpah,1,RAT('\',progpah)-1)
  
```

ORGANIZER Setup Code

FoxPro generates and executes setup code before the menu definition code and other menu code. Because setup code is executed first, typically you use it to do some or all of the following:

- Create memory variables and arrays used by the menu.
- Save the current environment, which is restored later.
- Create a new environment.
- Specify an error-handling routine.
- Save the current menu.

If your menu replaces specific FoxPro system menu pads, your setup code should release the corresponding system menu pads.

An Example of Generated Setup Code

- The following example illustrates the setup procedure generated by FoxPro for the ORGANIZE menu.

```
* *****
* *
* * Setup Code *
* *
* *****
*
```

RELEASE PAD _MSYSTEM OF _MSYSMENU ← Remove the System menu pad.

SET TALK OFF ← Suppress the output of commands to the screen.

proppath = SYS(16) ← Program name with path

npath = SUBSTR(proppath,1,RAT('\',proppath)-1) ← Remove the program name and extension.

```
conv = '"' + npath + "\convert.app" + '"'
rest = '"' + npath + "\restaurs.app" + '"'
clie = '"' + npath + "\clients.app" + '"'
fami = '"' + npath + "\family.app" + '"'
cred = '"' + npath + "\credit.app" + '"'
cred = '"' + npath + "\credit.app" + '"'
accn = '"' + npath + "\accnts.app" + '"'
tran = '"' + npath + "\trans.app" + '"'
```

Add a new path to the applications.

opath = SET("PATH") ← Current path

IF AT(npath,opath) = 0 ← If the new path isn't in the old path...

Add the DBFS and REPORTS directories to the old path.

```
opath = npath+ ;
";"+npath+"\DBFS" + ;
";"+npath+"\REPORTS" + ;
IIF(EMPTY(opath),",",";") + opath
```

SET PATH TO &opath ← Enable the new path.

ENDIF

Procedure

You can create a general procedure that FoxPro executes when certain menu pads are chosen. (A general procedure isn't executed for menu pads already associated with commands, menu popups or other procedures.)

Suppose you are developing an application for which some menu pads do not yet have associated popups, procedures or other code. For these menu pads, you can create a code snippet stub that executes when the pads are chosen. A *code snippet stub* typically is a message that displays for an unfinished menu. For instance, you can create a general procedure that consists of the following code snippet:

```
WAIT 'Feature not available' WINDOW NOWAIT
```

Whenever a menu pad is chosen that has no associated code or popup, FoxPro displays the "Feature not available" message, and then the application continues.

To create a general procedure, type its code in the text editing region (Procedure box) of the General Options dialog. Or, choose the **Edit...** push button, choose **OK** then enter the code in the window that appears.

Generated Menu Code

In the generated menu code, FoxPro places an ON SELECTION MENU command after the menu definition code. If the code snippet in a general procedure is only one line long, FoxPro includes the snippet in the ON SELECTION MENU command; however, if the snippet is more than one line long, the menu code generator creates the following command:

```
ON SELECTION MENU DO <procedure name> IN <menu name>
```

The menu generator makes the code snippet a separate procedure and gives it the unique name <procedure name>.

Cleanup Code

You can create cleanup code for a menu by choosing **Cleanup...** in the General Options dialog.

Cleanup code typically contains code snippets that initially:

- Turn mark characters on or off.
- Enable or disable menu pads, popups and options.

When you choose a menu pad, popup or option, the cleanup code included in menu pad or option *procedures* can turn a mark character on or off, enable or disable a menu pad and so on.

In generated code, cleanup code follows the setup code and the menu definition code but precedes the procedures assigned to menu pads or menu options.

Cleanup Code Example

The following example shows the cleanup code for the MAINMENU menu. This cleanup code turns the mark characters on or off for individual options on the **Environment** popup.

For example, if the clock is set on when the cleanup code is executed, SET('CLOCK') = 'ON' returns true and the mark character for the **Clock** option appears.

| | | |
|-----------------------------------|--|------------|
| System Edit Record Window Reports | | 9:30:36 pm |
|-----------------------------------|--|------------|

Help

Calculator
Calendar/Diary
Puzzle
Environment ▶

OK

◆Clock
Extended Video
◆Sticky Menus
Status Bar

Client Manager

| | | |
|---|--|--|
| Company: Aspen Planning & Inc. | | Balance: 0.00 |
| Contact: Randy Flood Address: 41 Wept Drive 6086 New York, NY 10023 | | Notes: |
| Area-Phone: 718-023-3651 EXT: 742 | | CTRL+TAB to exit |
| Client Type: () Active (◊) Inactive () Prospect | | Cuisine Preference: American |

< Help > < New > < Save > < Cancel > < Balance >

Display the mark beside the Clock option if appropriate.

```
FOR i = 1 TO cntbar('environmen') ← Loop for the number of options.
DO CASE
CASE PRMBAR('environmen',i) = 'Clock' ← Clock option.
→ SET MARK OF BAR i OF environmen TO SET('CLOCK') = 'ON'
```

Display the mark beside the Extended Video option if appropriate.

```
CASE PRMBAR('environmen',i) = 'Extended Video' ← Video option.
→ SET MARK OF BAR i OF environmen TO SROW() > 25
```

Display the mark beside the Sticky option if appropriate.

```
CASE PRMBAR('environmen',i) = 'Sticky' ← Menu option.
→ SET MARK OF BAR i OF environmen TO SET('STICKY') = 'ON'
```

Display the mark beside the Status Bar option if appropriate.

```
CASE PRMBAR('environmen',i) = 'Status Bar' ← Status bar option.
→ SET MARK OF BAR i OF environmen TO SET('STATUS') = 'ON'
ENDCASE
ENDFOR
```

Location

The **Location** radio buttons determine how FoxPro integrates your menu with the menus on the FoxPro system menu bar:

Replace Choosing **Replace** replaces the entire FoxPro system menu bar with your menu bar. When you choose this option, FoxPro puts the following command at the beginning of the menu definition code:

```
SET SYSMENU TO
```

This replaces the menu pads on the system menu bar with your menu pads.

Append Choosing **Append** adds your menu to the right end of the system menu bar.

Before Choosing **Before** places all your menus on the system menu bar before a menu you choose from the **Location** popup. The following clause is appended to each menu pad's DEFINE PAD command:

```
BEFORE <system pad name>
```

The <system pad name> is the name of the system menu pad you choose from the **Location** popup.

After Choosing **After** places all your menus on the system menu bar after the menu pad you choose from the **Location** popup. The following clause is appended to each menu pad's DEFINE PAD command:

```
AFTER <system pad name>
```

The <system pad name> is the name of the system menu pad you choose from the **Location** popup.

Mark... You can specify a global mark character for each menu pad by choosing the **Mark...** check box. The default mark character for menu pads is a diamond (◆).

When you choose the **Mark...** check box, a scrollable list of mark characters appears. You can choose a mark character from this list, and if you choose a one, it replaces the default mark character (◆).

Although you can use any character as a mark, the characters in the following table are better than others.

| Character | ASCII Value |
|-----------|-------------|
| ◆ | 4 |
| ● | 7 |
| * | 42 |
| ✓ | 251 |



Mark... only specifies a mark character for menu pads. It does not turn the mark character on or off.

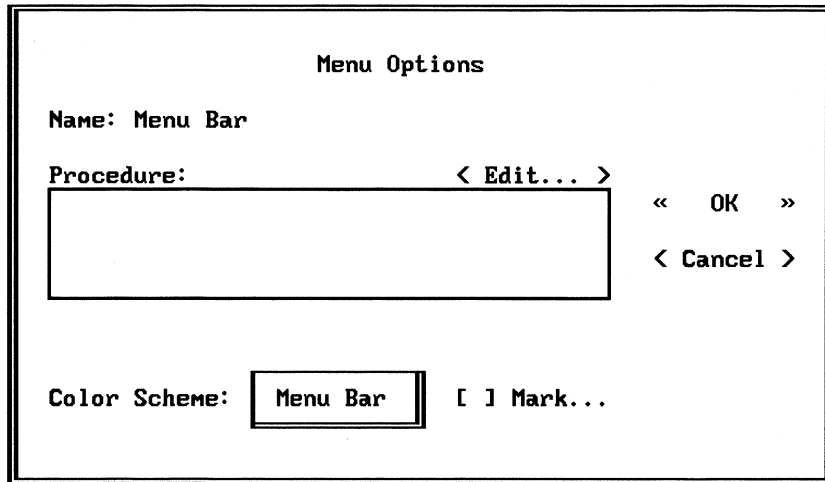
Generated Menu Code

When you specify a mark character, the menu generator adds the command SET MARK OF MENU to the menu definition code. For example, if you specify a bullet as the mark character, the menu generator adds the following line:

```
SET MARK OF MENU _MSYSMENU TO "●"
```

Menu Bar Options...

When you create menu pads with the Menu Builder, the option **Menu Bar Options...** appears on the **Menu** menu popup. Choosing **Menu Bar Options...** displays the Menu Options dialog.



The screenshot shows a dialog box titled "Menu Options". It contains the following elements:

- Name:** Menu Bar
- Procedure:** A text input field with the text "< Edit... >" to its right.
- Color Scheme:** A dropdown menu currently showing "Menu Bar" and a button labeled "[] Mark..." to its right.
- Buttons:** "OK" and "Cancel" buttons are located on the right side of the dialog.

Menu Options Dialog

In the Menu Options dialog you can:

- Create code snippets for a global procedure that executes when a popup option is chosen.
- Choose a color scheme that determines the colors of the menu bar and menu bar pads.
- Specify a mark character for menu pads.

Procedure

You can create a menu bar procedure that executes when *any* option is chosen from a popup. However, if a command, submenu, menu popup or procedure is assigned to a particular popup option, the menu bar procedure does not execute when that option is chosen.

To create a menu bar procedure, type its code in the text editing region (Procedure box) of the Menu Options dialog. Or, choose the **Edit...** push button, choose **OK** then enter the code in the window that appears.

Color Scheme

You can specify the colors of the menu bar and the menu pads by choosing a color scheme from the **Color Scheme** popup in the Menu Bar Options dialog. The Menu Bar color scheme 3 is the default.

Generated Menu Code

While generating a menu's code, FoxPro appends a **COLOR SCHEME** <color scheme number> clause to each **DEFINE PAD** command. The <color scheme number> is the option you choose from the **Color Scheme** popup.

Mark

You can specify a mark character for every menu pad on the menu bar by choosing the **Mark...** check box. A mark character appears in front of a menu pad when a specific condition occurs.

The default mark character for menu pads is a diamond (◆).

When you choose the **Mark...** check box, a scrollable list of mark characters appears. You can choose a character from this list and if you choose one, it replaces the default mark character (◆).



This option only specifies a mark character for menu pads. It does not turn mark characters on or off.

The mark character specified in the Menu Bar Options dialog overrides a mark character specified in the General Options dialog.

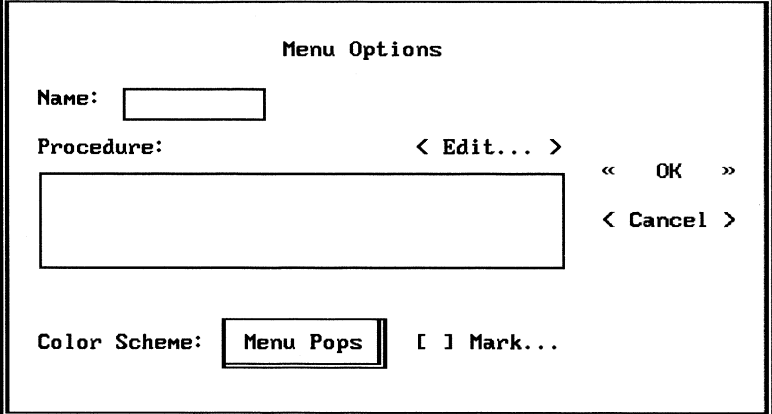
Generated Menu Code

When you specify a mark character in the Menu Bar Options dialog, the menu generator appends a MARK clause to each menu pad's DEFINE PAD command, as shown in the following example:

```
DEFINE PAD <pad name> OF <menu name> MARK "●"
```


Menu Popup Options...

When you create a menu popup with the Menu Builder, an option with the popup's name replaces the **Menu Bar Options...** option on the **Menu** menu popup. If you choose this option from the **Menu** menu popup, the Menu Options dialog appears.

The image shows a dialog box titled "Menu Options". It has a "Name:" label followed by a text input field. Below that is a "Procedure:" label followed by a large text area and a "< Edit... >" button. To the right of the text area are two buttons: "< OK >" and "< Cancel >". At the bottom, there is a "Color Scheme:" label followed by a dropdown menu showing "Menu Pops" and a "[] Mark..." button.

Menu Options

Name:

Procedure: < Edit... >

< OK >

< Cancel >

Color Scheme: [] Mark...

Menu Options Dialog

In this dialog you can:

- Create code snippets that execute when an option is chosen from the popup.
- Specify a different name for the popup.
- Choose a color scheme that determines the colors of the popup and its options.
- Specify a mark character for the popup's options.

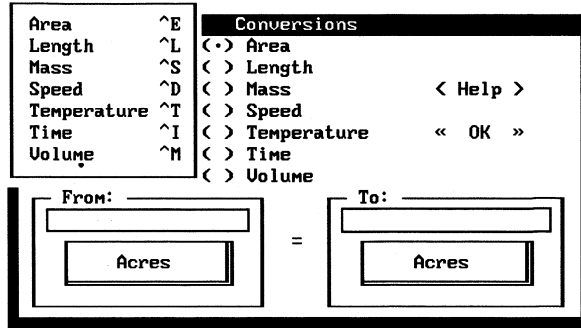
Procedure

You can create code snippets that are executed when *any* option is chosen from the popup. However, if a command, a submenu or another procedure is already assigned to an option, these code snippets do not execute when that option is chosen.

To create code snippets for the options, type the code snippets in the text editing region; or choose the **Edit...** push button, choose **OK** then enter the code in the window that appears.

Menu Popup Procedure Example

The following is the menu popup procedure for the CONVERSION menu. This procedure executes whenever an option is chosen from the **Conversions** menu popup. Using the SELECT command, the procedure queries the UNITS database, stores the results in an array and then uses the array to create the options in the **From** and **To** popups.



```
ON SELECTION POPUP conversion ;
DO _pvj16av0h
```

When any option is chosen from the CONVERSION popup ...

```
PROCEDURE _pvj16av0h
```

This procedure executes.

```
REGIONAL i, size
```

```
unittype = PROPER(PROMPT( ))
```

In the variable UNITTTYPE, store the prompt of the option.

```
SELECT DISTINCT units.unit ;
FROM units ;
WHERE units.type = unittype ;
ORDER BY units.type ;
INTO ARRAY fromarray
```

Get the screen popup options from the UNITS database, and then store the options in the FROMARRAY array.

```
size = ALLEN(fromarray)
DIMENSION toarray[size]
FOR i = 1 TO size
    fromarray[i] = ALLTRIM(fromarray[i])
    toarray[i] = fromarray[i]
ENDFOR
```

Create an array named TOARRAY from FROMARRAY. TOARRAY contains the options in the second screen popup.

```
frompop = fromarray[1]
topop = toarray[1]
fromval = SPACE(19)
toval = SPACE(19)
```

Initialize variables.

```
_CUROBJ = OBJNUM(fromval)
SHOW GETS WINDOW convert
```

Refresh the window containing the popups.

Name

A menu popup has a default name, which is the prompt text of the menu pad to which the popup is attached. If the popup is a submenu, the default name is the prompt text of the option to which the popup is attached. The default name appears in the Name text box. You can specify a different name for the popup in this text box.

Generated Menu Code

In the generated menu code, the following line represents the menu popup name:

```
DEFINE POPUP <popup name>
```

The <popup name> is the name displayed in the Name text box.

Color Scheme

You can specify the colors of the popup and its options by choosing a color scheme from the **Color Scheme** popup. The Menu Pops color scheme (color scheme 4) is the default.

Generated Menu Code

In the generated menu code, FoxPro appends a COLOR SCHEME <color scheme number> clause to the popup's DEFINE POPUP command in the menu definition code. The <color scheme number> is the number of the option you choose from the **Color Scheme** popup.

Mark

You can specify a mark character for every popup option by choosing the **Mark...** check box in the Menu Options dialog. The default mark character is a diamond (♦).

When you choose **Mark...**, a scrollable list of mark characters appears. You can choose a character from this list, and if you choose one, it replaces the default mark character (♦).



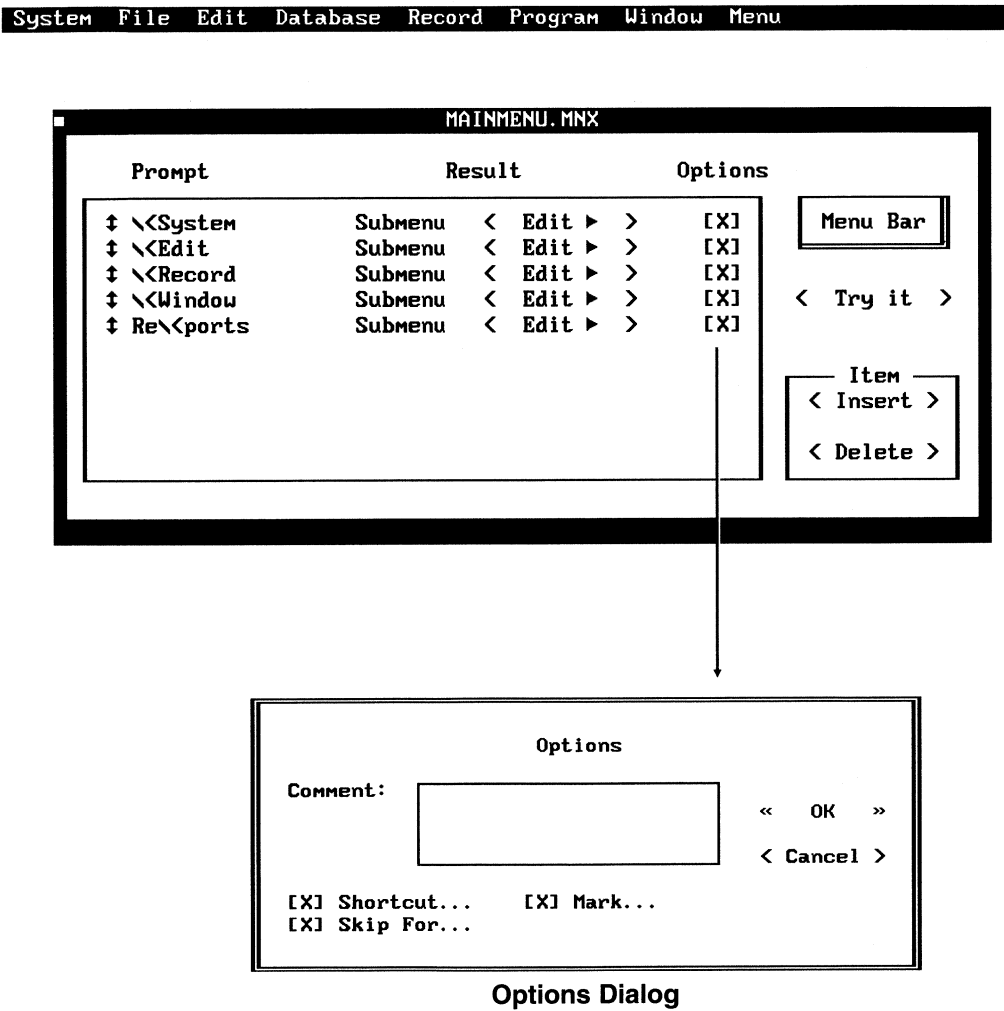
This option only specifies a mark character for the popup's options. It does not turn the mark character on or off.

Generated Menu Code

When you specify a mark character in this dialog, the menu generator appends a MARK clause to DEFINE POPUP.

Option Check Box

Every menu pad and option you create can have a comment, a keyboard shortcut, a unique mark character and a disabled status — if a specific condition exists. To create these options for a menu pad or an option, choose the **Options** check box that appears when you create the menu pad or option. When you choose this check box, the Options dialog appears.



Shortcut...

A keyboard shortcut is a key or key combination that you can press to choose a menu pad or option. To create a keyboard shortcut for a menu pad or option, choose the **Shortcut...** check box. When you choose this check box, the Shortcut dialog appears.

To specify a keyboard shortcut, enter a key label and key text in the Shortcut Dialog. The *key label* is the key or key combination that can be used to choose the menu pad or option. The *key text* appears to the right of a menu pad or popup option as a reminder. If you don't want to display key text in your menu, delete the key text from the Key Text text box.



If your menu uses the FoxPro system menu bar _MSYSMENU, key text reminders do not appear next to menu pads in the menu bar. However, they do appear next to popup options.

A keyboard macro takes precedence over a menu pad or popup option keyboard shortcut.

Generated Menu Code

When you create a keyboard shortcut for a menu pad or popup option, the menu generator adds a KEY clause to the menu pad's DEFINE PAD command or the popup option's DEFINE BAR command. The first expression in the KEY clause is the keyboard shortcut, and the second expression is the key text.

For example, if you use the key combination Ctrl+J for the shortcut and ^J as the key text, the menu generator adds the following line to DEFINE PAD or DEFINE BAR command:

```
KEY CTRL+J, "^J"
```

If you delete the key text from the Shortcut dialog, the menu generator adds the following line to the DEFINE PAD or DEFINE BAR command:

```
KEY CTRL+J, ""
```

Mark...

You can specify a mark character for a single menu pad or popup option by choosing the **Mark...** check box in the Options dialog. FoxPro puts a mark next to a menu pad or popup option when a specific condition exists. For example, FoxPro can put a mark next to an option when the option is enabled.

The default mark character for menu pads and popup options is a diamond (◆).

When you choose the **Mark...** check box, a scrollable list of mark characters appears. From this list, you can choose a mark character, and if you choose one it replaces the default mark character (◆).



If you specify a mark character for a menu pad or popup option in this dialog, this character overrides mark characters specified in other dialogs.

Mark... only specifies a mark character for a menu pad or popup option. It does not turn the mark character on or off.

Generated Menu Code

When you specify a mark character for a menu pad or popup option, the menu generator adds a MARK clause to the menu pad's DEFINE PAD command or the popup option's DEFINE BAR command. For example, if you specify a bullet (●) as the mark character, the menu generator appends the following to DEFINE PAD or DEFINE BAR:

```
MARK "●"
```

Skip For...

If you choose the **Skip For...** check box in the Options dialog, you can disable or enable a menu pad or popup option based upon a logical condition that you specify. If the logical condition evaluates to true (.T.), FoxPro disables the pad or option, preventing you from choosing it. If the logical condition evaluates to false (.F.), FoxPro enables the pad or option, allowing you to choose it.

To specify a logical condition, use the Expression Builder, which FoxPro displays when you choose the **Skip For...** option. For example, consider the following expression:

```
CDOw(DATE( )) = 'Monday'
```

If you enter this expression in the Expression Builder dialog, FoxPro enables the menu pad or popup option every day of the week except Monday (when CDOw(DATE()) = 'Monday' is true).

Generated Menu Code

When you create a logical expression to disable or enable a menu pad or popup option, the menu generator adds a SKIP FOR clause to the menu pad's DEFINE PAD command or the popup option's DEFINE BAR command. For example, if you create the logical expression shown in the previous example, the menu generator adds the following line to the DEFINE PAD or DEFINE BAR command:

```
SKIP FOR CDOw(DATE( )) = 'Monday'
```

Pad Name...

If you choose the **Pad Name...** check box in the Options dialog, you can specify a name for the menu pad. If you do not specify a name, FoxPro uses the SYS(2015) function to create a name for the menu pad.

Generated Menu Code

When you specify a menu pad name by choosing the the **Pad Name...** check box, FoxPro includes the name in the menu definition code. For example:

```
DEFINE PAD <MyMenuPad> OF _MSYSMENU
```

Comment

You can type a comment for the menu pad or popup option in the text editing region of the Options dialog. The comment can serve as a reminder or provide information about the menu pad or popup option. FoxPro stores the comment in the COMMENT memo field in your menu's database.

Generated Menu Code

FoxPro does not include the comment in the generated menu code.

Debugging Your Menus

After you create a menu, it might not behave as you intended. To diagnose the menu's problems, you can examine its code using the Trace and Debug windows. You can also open, examine and modify a menu program in the FoxPro text editor using the MODIFY COMMAND command. For details about this command, refer to the *FoxPro Language Reference*.



Never change the menu program (the .MPR file). If you change it, you will lose the changes when you modify menus using the Menu Builder and then regenerate the menu program.

Trace Window

You can display menu program code in the the Trace window as the program executes. FoxPro highlights each line as it is executed.

In this window, you can set breakpoints on lines of menu program code to pause program execution just before each line, and you can single step through a menu program, executing one line at a time.

Debug Window

You can display the values of memory variables, array elements, functions and expressions in the Debug window as a menu program executes. Additionally, you can set breakpoints in this window to halt program execution when the values of these items change.

FoxDoc

Included with FoxPro, FoxDoc is a tool that creates documentation for menu programs or applications containing menus.

For more information about using FoxDoc with menu programs, see Documenting Applications with FoxDoc later in this manual.

Comment Boxes

The menu generator automatically inserts a comment box before each menu program section. Comment boxes describe the origin of code that follows the box and are useful for debugging menu programs. Additionally, the information in comment boxes is used by FoxDoc to document the program.

A comment box describes the type of code (setup, menu definition, cleanup or procedure) that follows the comment box. When the code is a procedure, the comment box includes:

- The procedure name.
- The command that calls the procedure.
- The number of the .MNX database record that contains the procedure.
- The prompt text of the menu pad or popup option that calls the procedure.
- The procedure's snippet number. (Each procedure in the generated code has a number.)

For more information about debugging programs with FoxPro, see Debugging Your Applications later in this manual.

The following example is a procedure comment box from the CONVMENU menu program.

```

*          *****
*          *
*          * _PX90P3U3N  ON SELECTION POPUP conversion
*          *
*          * Procedure Origin:
*          *
*          * From Menu:  CONVMENU.MPR,           Record:    4
*          * Called By:  ON SELECTION POPUP conversion
*          * Snippet:    1
*          *
*          *****

```

Additional Tip

Hiding the Command Window

If your application runs in the development version of FoxPro (instead of a runtime version), you might want to hide the Command window so that only your menu is visible. To hide the Command window, create a small window and then use the `ACTIVATE WINDOW` command to activate the Command window in the small window. Do not activate the small window. For example, the following program lines hide the Command window:

```
DEFINE WINDOW hidecomm FROM 1,1 TO 3,3  
ACTIVATE WINDOW command IN hidecomm
```

4 Coordinating Screens and Menus

Many screens have associated menu systems that are called from the screen program. These menu systems usually contain:

- Options with keyboard shortcuts for accessing screen controls.
- Options that are not available in the screen because they are used infrequently.
- Options that trigger irreversible actions (such as PACK).

This chapter covers managing a menu system and accessing screen controls via a menu.

Managing a Menu System

Menus created in the Menu Builder automatically interact with the FoxPro menu system. If you create a menu and add it to the FoxPro menu system, you don't have to explicitly activate the menu with `ACTIVATE MENU`.

The following sections give you more information about managing menus that you create.

Accessing Menus During a READ

The `READ` command activates controls in FoxPro screens. Whether or not your menus are accessible when a `READ` is active depends on the type of `READ` you issue.

A modal `READ` is a `READ` that includes the `MODAL` key word or a `WITH <window title list>` clause. When you issue a modal `READ`, your menu is disabled. However, during the `READ` you can reactivate the menu and make it accessible by including an appropriate `WHEN` clause in the `READ` command.

For example, to reactivate a menu that interacts with the FoxPro menu system, include the following statement in a `READ WHEN` clause:

```
SET SKIP OF MENU _MSYSMENU .F.
```

You can also use `SET SKIP OF PAD` and `SET SKIP OF BAR` in a `READ WHEN` clause to selectively enable menu pads and options in your menu.

After you issue a `READ`, access to your menu depends on the setting of `SYSMENU`, as discussed in the following section.

Controlling Menus with SET SYSMENU

With the SET SYSMENU command, you can disable your menu, selectively add and remove items from the menu, restore the default FoxPro menus, and control access to your menu during program execution. Some of the ways you can use SET SYSMENU are:

- SET SYSMENU ON – Your menu bar is accessible during program execution when FoxPro waits for keyboard input, such as during BROWSE, a non-modal READ, or MODIFY MEMO. The menu bar is not displayed, but you can display it and make it accessible by pressing the Alt or F10 keys or by double clicking the right mouse button.
- SET SYSMENU OFF – Your menu bar is not accessible during program execution.
- SET SYSMENU AUTOMATIC – Your menu bar is displayed at all times during program execution and is accessible during program execution when FoxPro waits for keyboard input.
- SET SYSMENU TO DEFAULT – The default FoxPro menu system is restored to its default configuration.

For more information about the SET SYSMENU command, see the FoxPro *Language Reference*.

Saving and Restoring Menus

PUSH MENU and POP MENU help you save and restore menus. Using these commands, you can push a menu onto a *stack* in memory and restore it later by popping it off the stack.

Pushing a menu onto a stack saves its current state but does not remove it from the screen. While the menu is saved in memory, you can change the menu on the screen, or you can replace it with another menu. After changing or replacing the original menu, you can restore the original menu by using POP MENU.

Menus are pushed onto and popped off the stack in a “last in, first out” order. The number of menus saved in memory is limited only by the amount of memory available. For more information about `PUSH MENU` and `POP MENU`, refer to the *FoxPro Language Reference*.

The ORGANIZER application demonstrates how menus are replaced and restored. When you choose a menu option in the ORGANIZER, a screen program runs. This program pushes the current menu into memory then runs a menu program that creates a new menu to replace the original one. When the screen program ends, the original ORGANIZER menu is restored (popped) from memory.

The following example shows the `CONVERT.SPR` screen program commands that `PUSH` the ORGANIZER menu onto a stack in memory, replace the ORGANIZER menu with its own menu and then restore the original ORGANIZER menu when the screen program ends.

| | | |
|----------------------------------|---|---|
| <code>PUSH MENU _MSYSMENU</code> | ← | Defined in the setup code for the screen. |
| <code>.</code> | | |
| <code>.</code> | | |
| <code>.</code> | | |
| <code>DO convmenu.mpr</code> | ← | Defined in the READ WHEN code snippet for the screen. |
| <code>.</code> | | |
| <code>.</code> | | |
| <code>.</code> | | |
| <code>POP MENU _MSYSMENU</code> | ← | Defined in the cleanup code for the screen. |

Calling Screen and Menu Programs

To call a menu program, use the following syntax:

```
DO <menu name>.MPR
```

Similarly, to call a screen program, use the following syntax:

```
DO <filename>.SPR
```

You must include the `.MPR` or `.SPR` extension, because different types of executable files such as menus, screens and queries, can have the same names.

Accessing Screen Controls via a Menu

If screen controls such as radio buttons are frequently used, you can help users by allowing access to these controls via menu options and keyboard shortcuts. For example, the `CONVERT.SCX` screen has a set of radio buttons, menu options, and keyboard shortcuts that you can use to select a measurement unit, such as area, length or mass.

To define a menu option, you can often use the code that defines the behavior of the corresponding screen control. Simply copy the code used for the screen control into the code snippet for the menu option.

5 Project — The Main Organizing Tool

The Project Manager organizes a FoxPro application by gathering into a project the necessary components, including screens, menus, programs, reports and so on. The project ensures that the components are up-to-date when you want to build the application.

Creating a project is the first step in developing a FoxPro application. To create a project, you add all application components to a new project file, even if the components are not complete. Then, you develop the components by editing them in the Project window.

This chapter covers:

- Understanding the advantages of a project
- Knowing what a project can contain
- Creating one project or several projects
- Using a home directory for portable applications
- Selecting a main file
- Including modifiable files in applications
- Coping with unknown references in a project
- Including procedural code in a project

Advantages of a Project

Locates and Assembles Referenced Files

When you build a project, FoxPro automatically locates and gathers all components of the application. You can build a project from an existing application by adding the startup program for the application and then rebuilding the project. When built, the project includes all referenced programs, screens, menus, reports, queries, labels and libraries.

Remembers the Location of Every File it Contains

A project tracks the location of all components that comprise an application. This tracking ability gives you flexibility in how you organize programs, screens and other components on your hard disk. For example, because you can put the components anywhere on the disk, you can create many directories so that you can organize the components by function, subsystem, or other appropriate category.

Accesses Prewritten Programs and Interface Components Easily

Because a project permits an application's components to be in many directories, it permits access to libraries of prewritten program elements like control panels and browsers that are stored in a common directory and used by many applications.

Stores Object Code

A project stores object code in its memo fields. This reduces the disk clutter caused by saving object files for each compiled version of a program.

Tracks Current Versions of Files

When you build a project, FoxPro ensures that the object code stored in the project is current. If the code isn't current, FoxPro recompiles programs, regenerates and recompiles screens and menus, and so on. This tracking feature is similar to the function of "make" utilities, with which you might be familiar.

Streamlines Distribution

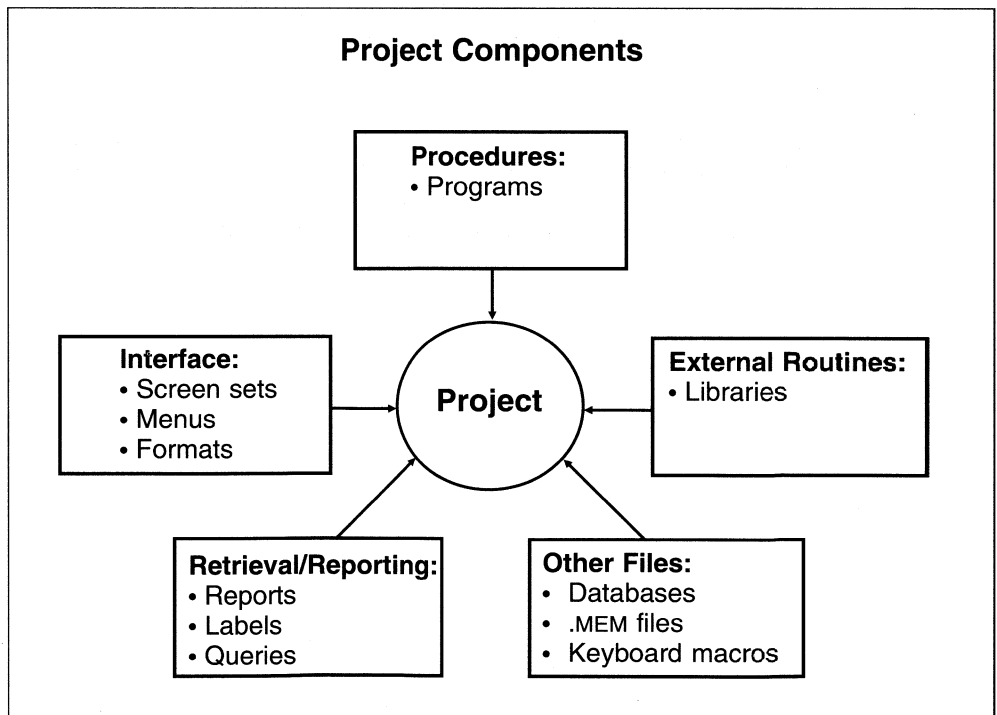
When you generate an application or .EXE file from a project, all components of the application are gathered into a single .APP or .EXE file. This makes distribution of the application particularly convenient.

What Can Projects Contain?

A project coordinates all components of an application. FoxPro projects can contain the following components: programs, menus, formats, queries, reports, labels, libraries, screen sets (containing one or more screens), and any other type of file.

These components fall into the following categories:

- Procedural: programs
- Interface: screen sets, menus, formats
- Data Retrieval/Reporting: reports, labels, queries
- External API Routines: libraries
- Other Components: databases, .MEM files, keyboard macros, and so on



One Project Versus Multiple Projects

The ORGANIZER contains several projects — one for every module of the application. The ORGANIZER uses a menu to call the appropriate project. The following projects are part of the ORGANIZER:

- ACCNTS.PJX
- CLIENTS.PJX
- CONVERT.PJX
- CREDIT.PJX
- FAMILY.PJX
- ORGANIZE.PJX
- RESTAURS.PJX
- TRANS.PJX

Though the ORGANIZER includes several projects, it could include only one project. The advantage of using one project for an entire application is that changes made to shared utility screens, programs, and so on are propagated throughout the project.

Multiple projects are useful in applications for which you change or sell individual modules of the application. With multiple projects, remember that when you make changes to a component used by several projects, you must rebuild all the projects containing the component.

Home Directory for Portable Applications

Each project has a home directory that you can use to make your applications portable. While developing an application, save the project and the application built from the project in the home directory.

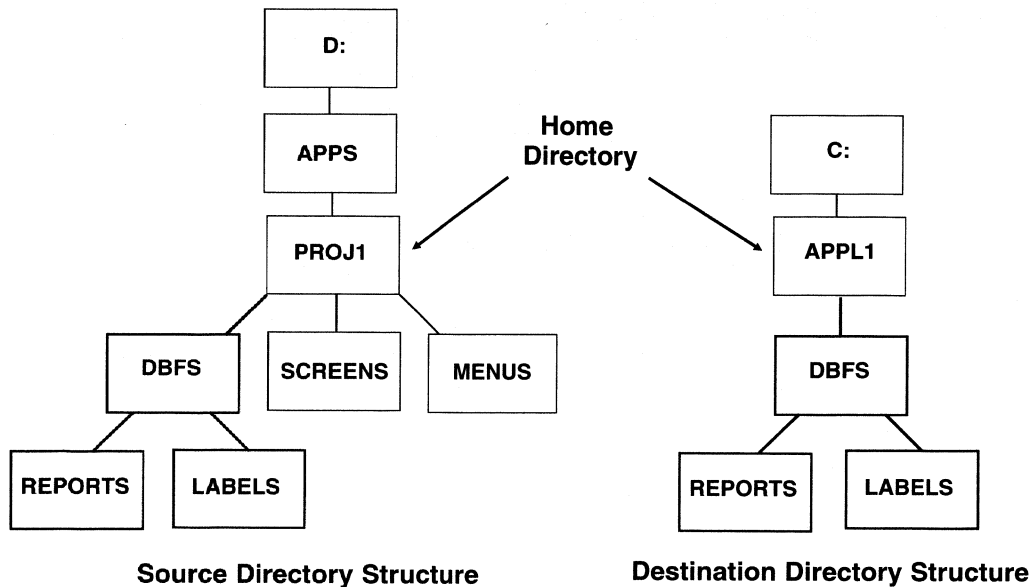
You can specify a home directory in the Home Directory area of the Project Options dialog. For details, refer to the chapter titled Project Manager in the *FoxPro User's Guide*.

Store files needed by the project in subdirectories of the home directory. (Create a subdirectory for each type of file needed by the project so that you can organize files by file type.)

To distribute an application to another computer, you can use any directory on the destination computer as the home directory. The home directory can have any name and can be anywhere in the directory structure.

When distributing an application, you must distribute any project files marked as excluded (refer to Including Modifiable Files in Applications later in the chapter). On the destination computer, first duplicate the names and structures of directories holding the excluded files on the source computer, and then copy the files from the source directories to the destination directories.

To summarize the distribution procedure, copy the application to its new home directory, create the needed subdirectories in the home directory and then copy the excluded files to these subdirectories.

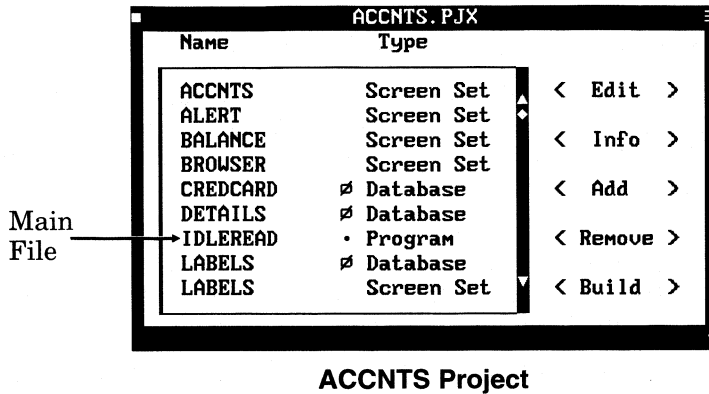


In the previous example, the source home directory is `D:\APPS\PROJ1`, and this directory contains subdirectories for databases, reports, labels, screens and menus.

The destination home directory is `C:\APPL1`, and this contains subdirectories for tables, reports and labels. These directories exist because the files in them were marked as excluded in the source directories. Notice that each directory containing these files has the same name and structure as that of the corresponding directory on the source computer.

Selecting a Main File

When you run an application, the main file in the project executes first. In the Project window, the main file has a bullet next to its type, as shown in the following figure.



For example, in the ACCNTS project, the IDLEREAD file starts the ACCNTS module when you choose **Money Manager** from the **Organize...** submenu and then choose **Accounts** from the next submenu.

Typically, the main file is the first executable file (screen, menu, or program) added to the project, but you can specify a different main file, by using **Set Main** on the **Project** menu.

Including Modifiable Files in Applications

All files referenced by an application should be part of a project. When you build an application from a project, FoxPro combines all executable project files (programs, screens, menus), making them part of the application's code. FoxPro does not automatically include non-executable files, such as reports, labels and databases; however, these can be included in read-only form.

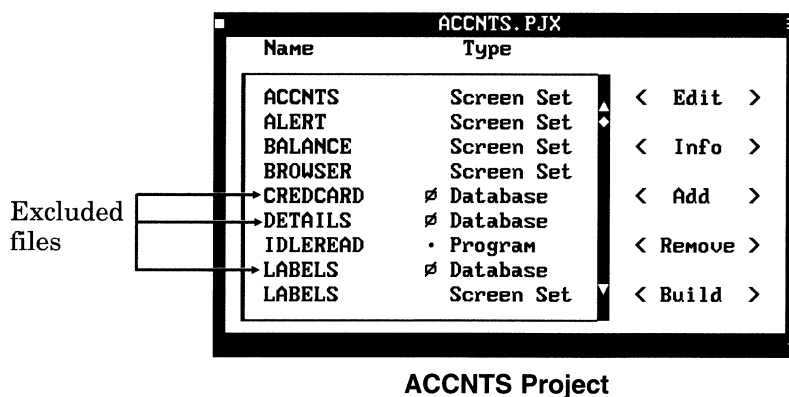
To give users the ability to change a non-executable project file, select the file in the Project window, then choose **Exclude** from the **Project** menu popup. A \emptyset appears in the Project window, next to the selected file.



Exclude files such as databases, indexes, reports and labels that you want the user to be able to change. You must distribute excluded files with an application if the application needs them.

Include help databases and databases that contain look-up information to prevent users from modifying them.

For example, in ACCNTS.PJX, three databases are excluded so that users can modify them when running the ORGANIZER.



If you change your mind and want to include a file you excluded, select the file in the Project window, and then choose **Include** from the **Project** menu popup.

Unknown References in Projects

When you build a project, the Project Manager alerts you when it cannot find a file or an array that is referenced in the project. In this case, do one of the following:

- Ensure that the name is spelled correctly and that the file or array exists.
- For a file, manually add the file to the project, and then build the project.
- Temporarily ignore the message and build the project without resolving the reference. Doing this usually does not cause problems with an application but might cause problems with an executable file.
- Add an `EXTERNAL` command so that FoxPro automatically includes the file or finds the array, and then build the project.

Use `EXTERNAL` to include files or resolve undefined references in a project created by the Project Manager. You must include a key word (`LABEL`, `LIBRARY`, `MENU`, `PROCEDURE`, `REPORT` or `SCREEN`) before the file name to tell the Project Manager the type of file to include in the project. `EXTERNAL` is used only by the Project Manager and is ignored during program execution.

The following examples illustrate ways of using the `EXTERNAL` command.

Use an `EXTERNAL PROCEDURE` command to identify an external procedure or a user-defined function (UDF):

```
EXTERNAL PROCEDURE delblank          PROCEDURE delblank must exist
STORE 'delblank' TO trimblanks
DO (trimblanks) WITH 'A B C D E'
```

If a report definition file is referenced with a name expression or macro substitution, use the `EXTERNAL REPORT` command:

```
EXTERNAL REPORT dataentr            && REPORT dataentr must exist
STORE 'dataentr' TO reportfile
MODIFY REPORT (reportfile)
```

If you create an array in a program and then use the array in a lower-level program, as in the following example, you might need to include the **EXTERNAL ARRAY** command to tell the Project Manager where to look for the array outside the program:

```
DIMENSION invoice(4)
STORE 'Paid' TO invoice
DO dispinvo

*** Program dispinvo ***
PROCEDURE
EXTERNAL ARRAY invoice
? invoice(1)
? invoice(2)
? invoice(3)
? invoice(4)
RETURN
*** End of dispinvo program ***
```

When you pass an array to a UDF or procedure, you might need to identify the array in the UDF or procedure for the Project Manager. To do this, use the **EXTERNAL ARRAY** command:

```
DIMENSION firstarray(2)      && Create an array
EXTERNAL ARRAY arraytwo      && Name of array used in the UDF
SET TALK OFF
STORE 10 TO firstarray(1)
STORE 2 TO firstarray(2)
= ADDTWO(@firstarray)        && Pass array by reference to a UDF

FUNCTION ADDTWO
PARAMETER arraytwo
CLEAR
arraytwo(1) = arraytwo(1) + 2
arraytwo(2) = arraytwo(2) + 2
? arraytwo(1)
? arraytwo(2)
```

For more information about the **EXTERNAL** command, refer to the *FoxPro Language Reference*.

Procedural Code in Projects

The Project Manager combines files into a single application. In most applications, you will include procedural code to do one or more of the following:

- Provide an error handling routine.
- Establish a global working environment (save the current environment and create a new environment).
- Preserve and restore the system menu bar.
- Test for available resources.
- Contain utility procedures that don't pertain to a specific screen or menu, but might be used by several screens or menus.

In the ORGANIZER, each project except for CONVERT contains a main program that establishes a global working environment for the application. The following sections use examples from the ORGANIZER to illustrate the use of procedural code in projects.

Error Handling

One of the first procedure calls in IDLEREAD.PRG is to an error handling routine. An error-handling routine traps for errors in your application so that you can gracefully recover from them.

When you include an error-handling routine at the beginning of your startup program, you can catch any error that occurs in the application (including the startup program). For instance, when an error occurs in the ORGANIZER, the following procedure executes.

```

*
* ERRORHANDLER - Error Processing Center.
*
PROCEDURE errorhandler
  PARAMETER messg, lineno
  PRIVATE fromrow, fromcol, torow, tocol
    fromrow = INT((SROW( )-6)/2)
    fromcol = INT((SCOL( )-50)/2)
    torow   = fromrow + 6
    tocol   = fromcol + 50

    DEFINE WINDOW alert;
      FROM fromrow, fromcol TO torow, tocol;
      FLOAT NOGROW NOCLOSE NOZOOM SHADOW DOUBLE;
      COLOR SCHEME 7

  ACTIVATE WINDOW alert

  @ 0,0 CLEAR
  @ 1,0 SAY PADC(ALLTRIM(messg), WCOLS( ))
  IF NOT EMPTY(lineno)
    @ 2,0 SAY PADC("Line Number: "+STR(lineno,4), WCOLS( ))
  ENDIF
  @ 3,0 SAY PADC("Press any key to cleanup and exit", WCOLS( ))
  WAIT ""

  ON ERROR
  POP MENU _MSYSMENU
  CLEAR READ ALL
  RELEASE WINDOW alert
  RELEASE workarea, exact, safety
  CANCEL

RETURN

```

Make variables private.

Define window coordinates.

Define an error window.

Display the error window.

Display an error message in the window.

Clean up the environment.

Saving the Current Environment

If your application returns control to the Command window or another application when finished, save the current environment in your startup routine so that you can restore it later. However, if your application returns to MS-DOS[®] when finished, you do not need to restore the environment.

The environment includes:

- Open files such as databases and indexes in all 25 work areas
- Relations between databases
- Filters in effect
- DEFAULT and PATH settings
- Current procedure, help and resource files
- SET command status (ON, OFF and so on)
- Color settings
- System menu bar and menu popups
- Keyboard macros

If you save the environment in your startup routine, you can restore it just before your application terminates in a cleanup routine. For example, if TALK is SET ON before your application runs and your application sets TALK OFF, the application should SET TALK ON before terminating.

TALK is ON by default but usually needs to be OFF when you run an application. If you want TALK to be OFF, make the first line in your application SET TALK OFF. For example, in the ORGANIZER, the following code from IDLEREAD.PRG checks to see if TALK is on and, if necessary, sets TALK OFF.

```
IF SET("TALK") = "ON"
    SET TALK OFF
    m.talkstat = "ON"
ELSE
    m.talkstat = "OFF"
ENDIF
```

How ORGANIZER Saves Environment Settings

The ORGANIZER application changes the settings of ESCAPE, NOTIFY, EXACT, SAFETY and DECIMALS, so UTILITY.PRG saves the current settings of these commands in memory variables. These memory variables are released from memory when the application is finished.

For example, the selected work area is saved with the following command:

```
m.area = SELECT()
```

The following lines from IDLEREAD.PRG save the setting of EXACT, SAFETY and DECIMALS:

```
m.escap = SET("ESCAPE")  
m.noti = SET("NOTIFY")  
m.exact = SET("EXACT")  
m.safety = SET("SAFETY")  
m.deci = SET("DECIMALS")
```

Additional Commands to Save Environment Settings

You can use the following commands to save the current FoxPro environment, keyboard macros, memory variables and arrays, screen or window image, and window definitions for later restoration:

- **CREATE VIEW** – Saves the current FoxPro environment. You can restore this environment by using **RESTORE VIEW**.
- **SAVE MACROS** – Saves the current keyboard macros to a file or memo field. You can restore these macros by using **RESTORE MACROS**.
- **SAVE TO** – Saves the current memory variables and arrays to a file or memo field. You can restore these variables and arrays by using **RESTORE FROM**.
- **SAVE SCREEN** – Saves the current screen or window image to memory. You can restore the screen or window from memory by using **RESTORE SCREEN**.
- **SAVE WINDOW** – Saves the current window definitions to a file or memo field. You can restore the windows by using **RESTORE WINDOW**.

Creating the New Environment

After you save the current environment, you can define a new environment for the application. Some aspects of the environment that you can define are:

- Global memory variables and arrays
- SET commands
- Colors
- Procedure, help and resource files

For example, in the ORGANIZER application, UTILITY.PRG defines the environment by using the following commands:

```
SET HELP TO "ORGHELP.DBF"
SET HELP ON
SET TEXTMERGE DELIMITERS
SET MEMOWIDTH TO 256
SET UDFPARMS TO VALUE
SET DATE TO AMERICAN
SET EXACT ON
SET SAFETY OFF
SET DECIMALS TO 18
```

Preserving and Restoring the System Menu Bar

To save the FoxPro system menu bar before changing it, push the bar into memory. By pushing the bar, you can restore it later. The following command preserves the FoxPro system menu bar:

```
PUSH MENU _MSYSMENU
```

If you push the system menu bar before changing it, you can restore the bar pushed by using the following command:

```
POP MENU _MSYSMENU
```

Testing For Resources

Your application might need certain resources to run, such as specific files, a specific amount of memory or disk space and so on. Several FoxPro functions can test for resources that your application might need:

- `FILE()` – Tests for the existence of a specified file on disk. Use `FILE()` if your application requires certain files.
- `SYS(2010)` – Returns the `FILES` setting in your `CONFIG.SYS` system configuration file. If your application opens many files, use `SYS(2010)` to ensure that you can open all of them.
- `MEMORY()`, `SYS(12)`, `SYS(1001)` and `SYS(1016)` – Test the amount of available memory. If your application requires a minimum amount of memory, these functions can determine if your application will run successfully.
- `DISKSPACE()` – Returns the amount of remaining disk space. Certain FoxPro operations (`SORT`, for example) require substantial disk space for the temporary work files they create.

If the required resources are not available, you can display a warning before executing the rest of the application.

Utility Procedures

Procedural code often contains utility procedures that do not pertain to a specific screen or menu but might be used by several screens or menus. Some of the utility procedures in `UTILITY.PRG` are:

- `STRIPEXT` – Removes the extension from a file name.
- `STRIPPATH` – Removes the path from a file name.
- `LOCATEDB` – Attempts to locate and open a database, asking for your help if the table cannot be found.
- `CHECKFPT` – Checks to see if a memo file exists for a database.

Other Development Tools

6 Debugging Your Application

FoxPro provides a comprehensive set of program debugging tools, including a Trace window, Debug window, text editor and online help. These tools help you locate and fix program errors.

This chapter covers:

- Compilation errors
- Runtime errors
- Debugging suggestions

Program Errors

FoxPro programs can include two types of errors:

- Compilation errors that occur as a program compiles
- Runtime errors that occur as a program executes

For example, suppose you try to open the CUSTOMER database by issuing the USE command but you type it incorrectly:

```
MUSE customer
```

When compiled, this command produces the “Unrecognized command verb” error message. This represents a compilation error.

Alternatively, a runtime error occurs when a command compiles successfully but does not execute as expected. For instance, suppose the CUSTOMER database was erased, but you try to use it by issuing the command:

```
USE customer
```

This command is syntactically correct and compiles without error, but when executed it produces the error message “File ‘customer’ does not exist.” This represents a runtime error.

Compilation Errors

Before you can run a FoxPro program you must compile it. You can compile manually or let FoxPro do so automatically.

Compiling Manually

You can compile programs manually from the Compile dialog that appears when you choose **Compile...** from the **Program** menu:

- Choose **To .ERRs**, to create a compilation error log file having the program's name and the .ERR file extension.
- Choose **To File**, to direct compilation errors to a log file of your choice.

The compilation error log file contains each program line that caused an error during compilation, followed by the line's number and error message. You can use the FoxPro editor to open and examine the error log file.

Using the COMPILE Command

You can compile programs using COMPILE from either the Command window or a program. The SET LOGERRORS command determines whether you get a compilation error log file when you use COMPILE:

- If LOGERRORS is SET ON before you issue COMPILE, you get a compilation error log file having the program's name and the .ERR file extension. If a log file with the same name already exists, FoxPro writes over it without warning you.
- If a program compiles without errors, or if LOGERRORS is SET OFF before you compile, a log file is not created. Additionally, if a program compiles without errors and a log file with the compiled program's name exists, the file is deleted.

Compiling Automatically After Saving

When you create or edit a program using the FoxPro editor, a **Compile when saved** check box is available in the Preferences dialog. The Preferences dialog is displayed when you choose **Preferences...** from the **Edit** menu. If you check **Compiled when saved**, programs compile automatically each time you save them:

- If LOGERRORS is SET ON and **Compile when saved** is checked when you save a program, a compilation error log file having the program's name and the .ERR file extension is created. If a log file having the same name already exists, FoxPro writes over it without warning you.
- If a program compiles without errors, or if LOGERRORS is SET OFF, the log file is not created. If a program compiles without errors and a log file having the program's name already exists, the file is deleted.

Understanding the Causes of Compile Errors

Common causes of compilation errors are:

- Syntax errors that occur when a FoxPro command or function is misspelled or contains illegal characters.
- Mismatched or missing key words in any FoxPro structured commands. The structured commands are DO CASE, DO WHILE, IF ... ENDIF, FOR ... ENDFOR, and SCAN ... ENDSCAN. For example, using IF without including the required ENDIF generates an "If/else/endif mismatch" error.
- Program lines that are too long. A command or function cannot exceed the maximum line length of 2,048 characters.

If a program generates compilation errors, fix the program lines that generate the errors and then recompile the program. Continue this process until the program compiles without error.

Runtime Errors

Runtime errors are errors that occur while a program executes. Although a program can compile without errors, it can still generate runtime errors. Often, runtime errors are more difficult to find than compilation errors, but the FoxPro Debug and Trace windows can help you find them.

The Trace window displays a program's source code as the program executes, highlighting each line as it executes. Using this window, you can set breakpoints on program lines to pause program execution just before each line, and you can single step through a program, executing a single program line at a time.

As a program executes, the Debug window helps you monitor the values of memory variables, array elements, functions, .DBF fields, and expressions. Using this window, you can set breakpoints to halt program execution when the values of any of these items change.

The Trace and Debug windows can be open simultaneously, and you can set program breakpoints in both windows. The number of breakpoints you can set in both windows is limited only by the available memory.

For more information about the Trace and Debug windows, refer to the Program Menu chapter in the *FoxPro User's Guide*.

Debugging Suggestions

When debugging a program, consider the following suggestions.

Adjust Your Video Display

If your video hardware supports extended display modes, consider switching to an extended video mode before you start debugging. When your video display is in extended mode, you can open the Trace and Debug windows below any output windows the program puts on the screen.

If your video hardware does not support an extended video mode and a program window covers the Trace or Debug windows, pause program execution and then manually move the Trace or Debug window.

Document Your Code with FoxDoc

Use FoxDoc to document one program or an entire application. For example, with FoxDoc you can create tree structures for an application, application summaries, and variable cross-reference reports. For more information, refer to the Documenting Applications with FoxDoc chapter later in this manual.

SET ESCAPE ON

The command `SET ESCAPE OFF` prevents a program from pausing when you press Escape. If you're debugging a program that contains `SET ESCAPE OFF`, temporarily change it to a comment by putting an asterisk (*) at the beginning of the line containing `SET ESCAPE OFF`.

Pause Execution with Breakpoints

Set breakpoints in the Trace window to pause program execution before the line with the breakpoint.

Set breakpoints in the Debug window to pause program execution whenever the value of an item changes.

After pausing a program, you can issue commands (in the Command window) that help you examine and change the current FoxPro environment.

Tracing a Program that is Passed Parameters

You can pass parameters to a program and then trace its execution by taking the following steps:

1. Open the Trace window.
2. From the **Program** menu in the Trace window, choose the **Open...** option and then choose the program to trace.
3. Set a breakpoint on the first executable line of the program.
4. In the Command window, DO the program WITH the parameters.

DISPLAY MEMORY and DISPLAY STATUS

After pausing a program's execution, you can get valuable information about the current FoxPro environment by using the DISPLAY MEMORY and DISPLAY STATUS commands:

- DISPLAY MEMORY displays the name, type, contents, and status of all currently-defined memory variables and memory variable arrays. In addition, system memory variables and their values are displayed, as are all defined menu bars and pads, menus and windows.
- DISPLAY STATUS displays the current status of the FoxPro environment. Items displayed include active databases and indexes, relations, low-level file status, SET command settings, and information about record and file locking if you use FoxPro in a network environment.

Screen and Menu Code

You can include screens and menus in a project created by the Project Manager. To also include the generated screen and menu source code (for debugging purposes), choose the **Save Generated Code** check box in the Project Options dialog. This dialog appears when you choose **Options...** from the **Project** menu.

Generated menu and screen programs are well documented. All code snippets are labeled with their unique names (provided by the generator), the screens, the READ or object level clauses, and the object types with which the code snippets are associated.

If you find errors while running a generated program, suspend or cancel the program and note the location in the generated program where the errors occurred. Then, return to the screens or menus that generated the errors and change the appropriate code snippets.

SET DOHISTORY

SET DOHISTORY is useful for isolating particularly stubborn bugs. Setting DOHISTORY ON places commands from programs into the Command window as they execute, allowing you to edit and re-execute the commands (if appropriate).



Use SET DOHISTORY only temporarily because it creates a disk-based document as the program executes, and this can fill up a large disk *very quickly*. When you finish debugging your programs, be sure to remove any SET DOHISTORY ON commands from the programs before you execute or distribute them.

7 Using SQL SELECT

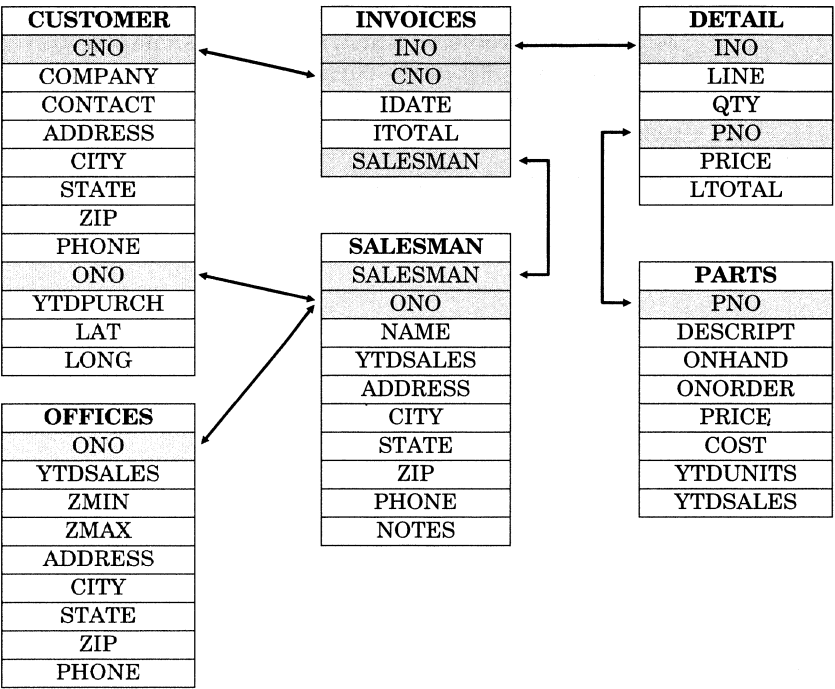
This chapter illustrates some features of SQL SELECT by posing problems and suggesting solutions to them. However, it illustrates only some of the things SQL SELECT can do. For details about SQL SELECT, refer to the FoxPro *Language Reference*.

Most of the problems illustrated in this chapter have several solutions because SQL is so versatile. Therefore, don't be concerned if your solutions do not match the ones shown on the following pages.

For problems having several solutions, pick the one that seems most natural to you. (However, remember that performance varies between solutions.)

Query Databases

The queries in this chapter use the CUSTOMER, INVOICES, DETAIL, SALESMAN, OFFICES and PARTS tutorial databases. These tables are in the TUTORIAL directory, and the following figure shows the relationships between them.



Database Relationships

Problems

- Q1** List the company names in the CUSTOMER database that contains the word “Computer”.
- Q2** Determine how many states have at least one customer residing in them.
- Q3** List the offices (that is, OFFICE.CITY) and invoiced total for each office, from the largest total to the smallest.
Hint: This requires joining three databases.
- Q4** List the part numbers, descriptions, total units and dollars sold for fast-moving parts, defined as those parts with more than 50 invoiced units.
Hint: Try the HAVING clause.
- Q5** List the companies that purchased more than one “Woodyard lizard,” the total number purchased, and the total amount paid.
Hint: This requires joining four databases.
- Q6** List companies that have an “x” in the third position of their company name.
- Q7** List company, city, and state for customers located in the same city as one of the offices.
- Q8** List descriptions of parts invoiced to customers in the state of NY.
Hint: This requires joining four databases.

- Q9** For each salesman, list all sales together with the average sales of salesmen who sold more.

Hint: The solution involves joining the SALESMAN database with itself. Joining a table with itself is called a “self-join”.

- Q10** List the pairs of part numbers and descriptions for which both parts were invoiced to the same customer.

Hint: This query generates over 6,000 rows in the result. If you’re not careful, you’ll generate over 12,000 rows.

- Q11** List the states in which at least one customer is located above 45 degrees latitude.

- Q12** List the states in which all customers are located between 40 and 45 degrees latitude.

- Q13** List companies with no invoices.

Hint: A subquery might be useful.

- Q14** Display the largest invoice amount together with the salesman’s name, the company to which the product was sold, the invoice number, and the invoice date.

Hint: A subquery might be useful.

- Q15** List states with no invoices.

- Q16** List the states in which every customer has an invoice.

Hint: Try using two queries.

- Q17** Given the following commission scale, calculate commissions on the sales in INVOICES by salesman showing the salesman's name, total sales, and commission. Display the information in ascending commission order.

| | | |
|-----|-------|----------|
| 10% | 1st | \$5,000 |
| 9% | 2nd | \$5,000 |
| 8% | 3rd | \$5,000 |
| 6% | above | \$15,000 |

- Q18** List the salesmen whose YTD sales are more than 10% above average YTD sales.

- Q19** Display the maximum distance between two customers within the same state for each of the following states: IL, WI, IA, MO, OH, and MI.

Hint 1: If you're an expert at celestial navigation or spherical trigonometry feel free to skip this hint. Otherwise, here's a function that calculates the distance in miles between two locations given the latitude and longitude of each.

```

FUNCTION geodist
PARAMETERS lat1, lng1, lat2, lng2
*
*           Degrees to Radian
*
lat1 = DTOR(lat1)
lng1 = DTOR(lng1)
lat2 = DTOR(lat2)
lng2 = DTOR(lng2)

x = SIN(lat1)*SIN(lat2) + ;
    COS(lat1)*COS(lat2)*COS(lng2-lng1)
RETURN 3959*ACOS(x)

```

Hint 2. Joining CUSTOMER.DBF with itself might be helpful.

Hint 3. See the solution to Query 10.

- Q20** List all customers with more than one invoice.

Problems

- Q21** Show the invoice number, part number and description for all parts that appear on only one invoice.
- Q22** Show all data on invoices dated between 17-May-90 and 23-May-90.
- Q23** Show any offices, together with their city and state, that have year-to-date sales exceeded by those of some individual salesman.
- Q24** Show any offices, with their city and state, that have year-to-date sales greater than those of all individual salesmen.

Solutions

- Q1** List the company names in CUSTOMER.DBF that contain the word "Computer".

Solution A

```
SELECT company ;  
      FROM customer ;  
      WHERE company LIKE "%Computer%"
```

Solution B

```
SELECT company ;  
      FROM customer ;  
      WHERE "Computer"$company
```

Solution C

```
SELECT company ;  
      FROM customer ;  
      WHERE AT("Computer",company) > 0
```

Note: Of these three techniques, only the first is available in standard SQL. The ability to use arbitrary expressions (including UDFs) throughout a query is unique to FoxPro version 2.5 and greatly enhances its power and ease-of-use.

Queries like these that involve searching for substrings are generally difficult to optimize and don't execute quickly. This is particularly true if memo fields are involved.

- Q2** Determine how many states have at least one customer residing in them.

Solution

```
SELECT COUNT(DISTINCT state) FROM customer
```

- Q3** List the offices (that is, OFFICE.CITY) and invoiced total for each office, from the largest total to the smallest.

Hint: This requires joining three databases.

Solution

```
SELECT offices.city, SUM(invoices.itotal) ;
      FROM offices, invoices, salesman ;
      WHERE invoices.salesman = salesman.salesman ;
            AND salesman.ono = offices.ono ;
      GROUP BY offices.ono ;
      ORDER BY 2 DESCENDING
```

- Q4** List the part numbers, descriptions, total units and dollars sold for fast-moving parts, defined as those parts with more than 50 invoiced units.

Hint: Try the HAVING clause.

Solution

```
SELECT detail.pno, parts.descript, ;
      SUM(qty), SUM(qty*detail.price) ;
      FROM detail, parts ;
      WHERE detail.pno = parts.pno ;
      GROUP BY detail.pno ;
      HAVING SUM(qty) > 50
```

- Q5** List the companies that purchased more than one “Woodyard lizard,” the total number purchased, and the total amount paid.

Hint: This requires joining four databases.

Solution

```
SELECT customer.company, SUM(detail.qty), ;
      SUM(detail.qty*detail.price) ;
      FROM customer, parts, invoices, detail ;
      WHERE customer.cno = invoices.cno ;
            AND invoices.ino = detail.ino ;
            AND detail.pno = parts.pno ;
            AND parts.descript = "Woodyard lizard" ;
      GROUP BY customer.cno ;
      HAVING SUM(detail.qty) > 1
```

- Q6** List companies that have an “x” in the third position of their company name.

Solution A

```
SELECT company ;
FROM customer ;
WHERE company LIKE "__x%"
```

Solution B

```
SELECT company ;
FROM customer ;
WHERE SUBSTR(company, 3, 1) = "x"
```

Note: Only Solution A is available in other SQL implementations.

- Q7** List company, city, and state for customers located in the same city as one of the offices.

Solution

```
SELECT customer.company, customer.city, customer.state ;
FROM customer, offices ;
WHERE customer.city = offices.city ;
AND customer.state = offices.state
```

- Q8** List descriptions of parts invoiced to customers in the state of NY.

Hint: This requires joining four databases.

Solution

```
SELECT DISTINCT parts.descript ;
FROM parts, customer, invoices, detail ;
WHERE customer.cno = invoices.cno ;
AND invoices.ino = detail.ino ;
AND detail.pno = parts.pno ;
AND customer.state = "NY"
```

- Q9** For each salesman, list all sales, together with the average sales of salesmen who sold more.

Hint: The solution involves joining SALESMAN.DBF with itself. Joining a database with itself is sometimes called a “self-join”.

Solution

```
SELECT a.salesman, a.name, a.ytdsales, AVG(b.ytdsales) ;
      FROM salesman a, salesman b ;
      WHERE a.ytdsales < b.ytdsales ;
      GROUP BY a.salesman
```

Note: Don’t try this on a large database. You’ll generate *lots* of output.

Q10

List the pairs of part numbers and descriptions for which both parts were invoiced to the same customer.

Hint: This query generates over 6,000 rows in the result. If you’re not careful, you’ll generate over 12,000 rows.

Solution

```
SELECT a1.pno, a1.descript, a2.pno, a2.descript ;
      FROM parts a1, parts a2, invoices b1, invoices b2, ;
           detail c1, detail c2 ;
      WHERE b1.ino = c1.ino AND c1.pno = a1.pno ;
           AND b2.ino = c2.ino AND c2.pno = a2.pno ;
           AND b1.cno = b2.cno ;
           AND a1.pno < a2.pno
```

Note: The trick that keeps the output down to 6,000 rows is the last line of the query that prevents selecting each pair of parts twice. Without it, each pair of parts (x,y) would be selected again as (y,x).

Q11

List the states in which at least one customer is located above 45 degrees latitude.

Solution

```
SELECT DISTINCT state ;
      FROM customer ;
      WHERE lat > 45
```

Q12 List the states in which all customers are located between 40 and 45 degrees latitude.

Solution A

```
SELECT DISTINCT state FROM customer ;
      WHERE state NOT IN ;
            (SELECT state FROM customer ;
              WHERE lat < 40 OR lat > 45)
```

Solution B

```
SELECT state FROM customer ;
      GROUP BY state ;
      HAVING 40 <= MIN(lat) AND MAX(lat) <= 45
```

Note: Two different solutions are shown above. However, Solution B executes over twice as fast as Solution A because it doesn't involve a subquery.

Q13 List companies with no invoices.

Hint: A subquery might be useful.

Solution A

```
SELECT company ;
      FROM customer ;
      WHERE cno NOT IN ;
            (SELECT cno FROM invoices)
```

Solution B

```
SELECT company ;
      FROM customer ;
      WHERE NOT EXISTS ;
            (SELECT * ;
              FROM invoices WHERE invoices.cno = customer.cno)
```

Note: You can reformulate queries that use EXISTS as equivalent queries that use IN or other clauses.

Whether you use EXISTS or IN is a matter of taste and style; use whichever seems natural. However, sometimes performance differences occur, especially if you can reformulate a query by eliminating subqueries.

Q14 Display the largest invoice amount together with the salesman's name, the company to which the product was sold, the invoice number, and the invoice date.

Hint: A subquery might be useful.

Solution

```
SELECT salesman.name, customer.company, invoices.ino, ;
       invoices.idate, invoices.itotal ;
FROM salesman, invoices, customer ;
WHERE salesman.salesman = invoices.salesman ;
       AND invoices.cno = customer.cno ;
       AND invoices.itotal = ;
               (SELECT MAX(itotal) FROM invoices)
```

Q15 List states with no invoices.

Solution A

```
SELECT DISTINCT state FROM customer;
WHERE state NOT IN ;
       (SELECT customer.state ;
        FROM customer, invoices ;
        WHERE invoices.cno = customer.cno)
```

Solution B

```
SELECT DISTINCT cc.state FROM customer cc;
WHERE NOT EXISTS ;
       (SELECT * ;
        FROM customer, invoices ;
        WHERE invoices.cno = customer.cno ;
        AND customer.state = cc.state)
```

Note: This example again illustrates that you can formulate a query using either IN or EXISTS, whichever seems most natural.

About Solution A — The DISTINCT clause is not required. If you're testing whether a value is IN a set, it doesn't matter how many repetitions of the value are in the set. Therefore, DISTINCT is implicit in subqueries associated with IN.

About Solution B — Why is "*" in the subquery select list for this query? The answer is that you're only interested in whether any rows exist; it doesn't matter which fields are in the rows.

Solution A and B are equally efficient.

Q16 List the states in which every customer has an invoice.

Hint: Try using two queries.

Solution

```
SELECT DISTINCT state ;
      FROM customer ;
      WHERE cno NOT IN ;
            (SELECT cno FROM invoices) ;
      INTO CURSOR cc
```

```
SELECT DISTINCT state ;
      FROM customer ;
      WHERE state NOT IN ;
            (SELECT state FROM cc)
```

Note: The first of the two queries selects all states that include at least one customer without an invoice.

The second query selects states appearing in CUSTOMER.DBF that are not among the states selected in the first query.

Q17 Given the following commission scale, calculate commissions on the sales in INVOICES by salesman, showing the salesman's name, total sales, and commission. Display the information in ascending commission order.

| | | |
|-----|-------|----------|
| 10% | 1st | \$5,000 |
| 9% | 2nd | \$5,000 |
| 8% | 3rd | \$5,000 |
| 6% | above | \$15,000 |

Solution

```
SELECT name, SUM(itotal), commiss(SUM(itotal)) ;
      FROM invoices, salesman ;
      WHERE invoices.salesman = salesman.salesman ;
      GROUP BY name ;
      ORDER BY 3
```

where “commiss” is the following function:

```
FUNCTION commiss
PARAMETER sales
PRIVATE c
c = MIN(sales, 5000) * 0.10
sales = MAX(0, sales-5000)
c = c + MIN(sales, 5000) * 0.09
sales = MAX(0, sales-5000)
c = c + MIN(sales, 5000) * 0.08
c = c + MAX(0, sales-5000) * 0.06
RETURN ROUND(c, 2)
```

Note: This query illustrates the usefulness of permitting arbitrary expressions and user-defined functions (UDFs) in queries.

- Q18** List the salesmen whose YTD sales are more than 10% above average YTD sales.

Solution

```
SELECT salesman, name FROM salesman ;
      WHERE ytdsales > ;
          (SELECT AVG(ytdsales)*1.10 FROM salesman)
```

Q19 Display the maximum distance between two customers within the same state for each of the following states: IL, WI, IA, MO, OH, and MI.

Hint 1. If you're an expert at celestial navigation or spherical trigonometry feel free to skip this hint. Otherwise, here's a function that calculates the distance in miles between two locations given the latitude and longitude of each.

```
FUNCTION geodist
PARAMETERS lat1, lng1, lat2, lng2
*
*      Degrees to Radian
*
lat1 = DTOR(lat1)
lng1 = DTOR(lng1)
lat2 = DTOR(lat2)
lng2 = DTOR(lng2)

x = SIN(lat1)*SIN(lat2) + ;
    COS(lat1)*COS(lat2)*COS(lng2-lng1)
RETURN 3959*ACOS(x)
```

Hint 2. Joining CUSTOMER.DBF with itself might be helpful.

Hint 3. See the solution to Query 10.

Solution

```
SELECT a.state, MAX(geodist(a.lat, a.long, b.lat, b.long)) ;
FROM customer a, customer b ;
WHERE b.zip < a.zip ;
      AND b.state = a.state ;
      AND a.state IN ("IL", "WI", "IA", "MO", "OH", "MI");
GROUP BY a.state
```

Q20 List all customers with more than one invoice.

Solution A

```
SELECT DISTINCT cno;
FROM invoices ;
WHERE EXISTS ;
      (SELECT * FROM invoices i2 ;
      WHERE invoices.cno = i2.cno ;
      AND invoices.ino <> i2.ino)
```

Solution B

```
SELECT cno ;
      FROM invoices ;
      GROUP BY cno ;
      HAVING COUNT(ino) > 1
```

Note: Solution A uses EXISTS and a subquery, whereas Solution B uses just one level of query. Solution B is simpler.

Q21 Show the invoice number, part number and description for all parts that appear on only one invoice.

Solution A

```
SELECT invoices.ino, detail.pno, parts.descript ;
      FROM invoices, detail, parts ;
      WHERE invoices.ino = detail.ino ;
            AND detail.pno = parts.pno ;
            AND NOT EXISTS ;
                  (SELECT * ;
                    FROM detail d2 ;
                    WHERE detail.pno = d2.pno ;
                      AND detail.ino <> d2.ino)
```

Solution B

```
SELECT detail.ino, detail.pno, parts.descript ;
      FROM detail, parts ;
      WHERE detail.pno = parts.pno ;
      GROUP BY detail.pno ;
      HAVING COUNT(DISTINCT detail.ino) = 1
```

Note: You can perform this query in at least two ways: the first uses a subquery, and the second does not. Consequently, the second is simpler and executes faster.

In Solution B, the DISTINCT modifier in COUNT is necessary to deal with a part that appears on an invoice several times.

Q22 Show all data on invoices dated between 17-May-90 and 23-May-90.

Solution

```
SELECT * FROM invoices ;
      WHERE idate BETWEEN {05/17/90} AND {05/23/90}
```

Note: This exercise illustrates the use of the BETWEEN operator.

Q23 Show any offices, together with their city and state, that have year-to-date sales exceeded by those of some individual salesman.

Solution

```
SELECT ono, city, state ;
      FROM offices ;
      WHERE ytdsales < ANY ;
            (SELECT ytdsales FROM salesman)
```

Note: This exercise illustrates use of the ANY quantifier.

Q24 Show any offices, with their city and state, that have year-to-date sales greater than those of all individual salesmen.

Solution

```
SELECT ono, city, state ;
      FROM offices ;
      WHERE ytdsales > ALL ;
            (SELECT ytdsales FROM salesman)
```

Note: This exercise illustrates use of the ALL quantifier.

8 Report Variable Hints

With Report Writer variables, most user-defined functions (UDFs) are no longer necessary.

Report variables allow you to do calculations that can be used in subsequent calculations. This means that, without actually writing any code, you can include information in your report that previously would have required writing a UDF. And once you define variables, it is easy to use them in subsequent calculations.

For example, if you wanted to make a time sheet report, you could define a variable ARRIVE with the following expression:

```
hour_in + (min_in / 60)
```

Another variable, LEAVE would contain a similar expression:

```
hour_out + (min_out / 60)
```

A third variable, DAYTOTAL could hold the total amount of hours worked during the day. Assign it the following expression:

```
leave - arrive
```

This third variable could be used in a variety of other calculations to determine the number of hours worked in a week, a month, a year, the average number of hours worked each day, and so on.

Report Variable Do's and Don't's

When setting up report variables, remember the following:

- The order in which the variables are initialized is important. If var1 is used to define the value of var2, var1 must be precede var2 in the list in the Report Variables dialog so that var1 will be evaluated first. In the previous time sheet example, ARRIVE and LEAVE would have to precede DAYTOTAL in the list of variables.
- The initial value of the variable is important. A default value of 0 is assigned if no other value is specified. In situations in which you are multiplying the variable in calculations, be sure to avoid a division by zero error.
- When the variable is reset is important. By default, variables are reset at the end of the report. If your calculations are dependent upon data grouping, make sure that you select the proper group from the **Reset** popup.
- If you reorder the groups in your report, your report variables may no longer be resetting on the correct field. For example, if an outer group is set to STATE and an inner group is by DATE, inverting the order of these groups would affect the variables that are reset according to the original positions of the groups.
- Report variables can be used to calculate selected information from a database. For example, if you wanted to count all the companies in a certain state in your report, you could create a variable with an expression similar to the following and then choose **Sum** in the Variable Definition dialog:

```
IIF(state="CA",1,0)
```


9 Arrays

FoxPro supports one- and two-dimensional memory variable arrays. An *array* is a collection of variables with a common name. Each item in the array is an *element* that you can reference by its row and column subscripts. *Subscripts* are numbers or numeric expressions that specify the location of an element in the array.

Each array element can contain any type of data (character, numeric, date or logical) and is initialized to a logical false (.F.) when the array is created.

This chapter covers:

- Creating arrays
- FoxPro array functions
- Manipulating arrays
- Public and private arrays
- Passing entire arrays to user-defined functions
- Transferring data between arrays and databases
- Arrays and SQL SELECT
- Arrays and FoxPro controls

Creating Arrays

To create an array, use the `DIMENSION` or `DECLARE` command. These commands are identical, so you can use them interchangeably. For details about the commands, refer to the *FoxPro Language Reference*.

Several FoxPro commands and functions store results to an array. If the array you specify doesn't exist, the following commands and functions automatically create the array:

| | |
|--------------------------------|----------------------------|
| <code>ACOPY()</code> | <code>CALCULATE</code> |
| <code>ADIR()</code> | <code>COPY TO ARRAY</code> |
| <code>AFIELDS()</code> | <code>SCATTER</code> |
| <code>APPEND FROM ARRAY</code> | <code>SQL SELECT</code> |
| <code>AVERAGE</code> | <code>SUM</code> |

Array names can include up to 10 characters, consisting of alphabetic characters, underscores, and numbers. An array name cannot begin with a number or contain embedded spaces.



Because FoxPro system memory variables begin with an underscore, avoid using an underscore as the first character of an array name.

To create a one-dimensional array, include one subscript that specifies the number of rows in the array.

To create a two-dimensional array, include a pair of subscripts: the first subscript designates the number of rows in the array, and the second subscript specifies the number of columns. Array subscripts always start at one.

The following examples create a one-dimensional array named **DEPTNUMBER** consisting of ten rows and a two-dimensional array named **TAXRATES** consisting of ten rows and five columns:

```
DIMENSION deptnumber(10)  
DIMENSION taxrates(10,5)
```

You can create several arrays using a single **DIMENSION** or **DECLARE** command. For instance, the following command creates the arrays described in the previous example:

```
DIMENSION deptnumber(10), taxrates(10,5)
```

FoxPro Array Functions

FoxPro supports a variety of functions for manipulating arrays. The following table lists these functions and their uses.

| Function | Description |
|---------------|---|
| ACOPY() | Copies a series of elements from one array to another. |
| ADEL() | Deletes an element, row, or column from an array. |
| ADIR() | Places matching file information into an array. |
| AELEMENT() | Returns an array element's number from its row and column subscripts. |
| AFIELDS() | Places database structure information into an array. |
| AINS() | Inserts an element, row, or column into an array. |
| ALEN() | Returns the number of elements, rows, or columns in an array. |
| ASCAN() | Searches a memory variable array for an expression. |
| ASORT() | Sorts a memory variable array in ascending or descending order. |
| ASUBSCRIPT() | Returns an element's row or column subscript from the element's number. |

For further information about these functions, refer to the *FoxPro Language Reference*.

Manipulating Arrays

This section describes how to initialize an entire array, how to initialize individual elements, and how to change the size or dimensions of an array.

Initializing Entire Arrays

You can initialize every array element to the same value by issuing the `STORE` command or by using the `=` operator. When `COMPATIBLE` is set to `OFF` or `FOXPLUS` (the default setting), and you include the name of an array (without subscripts) in `STORE` or `=`, every element in the array is initialized to the same value.

In the following example, every element in the array named `EPSILON` is initialized with the value “foo”.

```
SET COMPATIBLE OFF
DIMENSION epsilon(2,3)
STORE 'foo' TO epsilon
DISPLAY MEMORY LIKE epsilon
```

If `COMPATIBLE` is set to `ON` or `DB4`, as in the next example, the array is released from memory, a single memory variable with the array’s name is created, and this variable is assigned the value.

```
SET COMPATIBLE ON
DIMENSION epsilon(2,3)
STORE 'foo' TO epsilon
DISPLAY MEMORY LIKE epsilon
```

Referencing Array Elements

You can reference an array element by its element number or by its row and column subscripts. The first subscript specifies the row location of an element, and the second subscript specifies the column location.

For example, the subscripts `1,1` specify the element in the first row and first column of an array. The subscripts `2,5` specify the element in the second row and fifth column of an array.

In one-dimensional arrays, an element's number is equal to its row subscript. In two-dimensional arrays, an element's number is determined by counting along rows. For example, suppose you create the following 3-by-3 array:

```
a b c
d e f
g h i
```

The element numbers for a, b, and c are 1, 2, and 3. The element numbers for d, e and f are 4, 5, and 6, and so on.

Two frequently used array functions are AELEMENT() and ASUBSCRIPT():

- AELEMENT() returns an element's number when given its row and column subscripts.
- ASUBSCRIPT() returns the row or column subscript when given an element's number.

Assigning Values to Array Elements

Using the STORE command or = operator, you can assign values to array elements. You can assign different values to array elements, or you can assign the same value to each array element.

To assign a value to one array element, specify the element's subscript (one-dimensional array) or subscripts (two-dimensional array). In the following examples, STORE and = assign the values A through F to the array elements in the arrays named ALPHA and BETA.

```
DIMENSION alpha(2,3), beta(2,3)
STORE 'A' TO alpha(1,1)
STORE 'B' TO alpha(1,2)
STORE 'C' TO alpha(1,3)
STORE 'D' TO alpha(2,1)
STORE 'E' TO alpha(2,2)
STORE 'F' TO alpha(2,3)
```

```
beta(1,1) = 'A'
beta(1,2) = 'B'
beta(1,3) = 'C'
beta(2,1) = 'D'
beta(2,2) = 'E'
beta(2,3) = 'F'
```

```
DISPLAY MEMORY LIKE alpha
DISPLAY MEMORY LIKE beta
```

Additionally, you can assign values to elements by using their element numbers, as shown in the following example:

```
DIMENSION gamma(2,3)
STORE 'A' TO gamma(1)
STORE 'B' TO gamma(2)
STORE 'C' TO gamma(3)
STORE 'D' TO gamma(4)
STORE 'E' TO gamma(5)
STORE 'F' TO gamma(6)

DISPLAY MEMORY LIKE gamma
```

Redimensioning Arrays

You can change the size and dimensions of an array by using `DIMENSION` or `DECLARE`. With these commands, you can increase or decrease an array's size, convert a one-dimensional array to two dimensions, and reduce a two-dimensional array to one dimension.

When you increase the number of elements in an array, the contents of all elements in the original array are copied in element order to the newly-redimensioned array, and the additional array elements are initialized to a logical false (.F.).

You can remove specific rows or columns from an array by using `ADEL()`. When you decrease the number of elements, the array is truncated in element-number order.

Public and Private Arrays

Arrays, like memory variables, can be public or private.

Public Arrays

A public array is available to any program during the FoxPro session and is accessible from the Command window. To declare an array as public, use the PUBLIC command.

Arrays created in the Command window are public automatically.



If you try to declare an array PUBLIC *after* the array is created, an error message appears.

Private Arrays

PRIVATE hides an array in a higher level program from the currently executing program and called programs, allowing you to use an array having the same name in your currently-executing program.

Once the currently executing program containing the private array declaration finishes executing, you can access the array having the same name in the higher level program.



Unlike PUBLIC, PRIVATE cannot create an array; it only hides arrays declared in higher level programs from the current program or routine.

The following example creates and initializes a public array named MYARRAY.

| | |
|-------------------------------|--|
| SET TALK OFF | |
| CLEAR MEMORY | |
| CLEAR | |
| PUBLIC myarray(1,2) | ← Create a public array. |
| STORE 'main' TO myarray | ← Store a value to the array. |
| | |
| ? 'Main program' | |
| DISPLAY MEMORY LIKE myarray | ← Show the contents of the array. |
| ? | |
| | |
| DO this | ← Execute a lower level procedure. |
| ? 'Main program again' | |
| DISPLAY MEMORY LIKE myarray | ← Contents of the array – no change! |
| | |
| PROCEDURE this | ← Lower level procedure |
| PRIVATE myarray | ← Same array name as main. Declare it private. |
| DIMENSION myarray(1,1) | ← Different size array |
| STORE 'this' TO myarray | ← Store a value to the array. |
| | |
| ? 'Lower level procedure' | ← Show the contents of both arrays. |
| DISPLAY MEMORY LIKE myarray | |
| STORE 'this again' TO myarray | |
| ? 'Lower level again' | ← Change the array contents. |
| DISPLAY MEMORY LIKE myarray | ← Show the contents of both arrays. |
| ? | |
| RETURN | ← Return to the main program. |

Array Limitations

You can create up to 3,600 arrays. In FoxPro (X), the Extended version of FoxPro, each array can have a maximum of 65,000 elements. In the Standard version of FoxPro, each array can have a maximum of 3,600 elements.

The number of arrays and elements you can create might be limited by the available memory in your computer.

Passing Entire Arrays to User-Defined Functions

You can pass an entire array to a procedure or user-defined function (UDF). To pass an entire array, you pass it by reference, meaning that the UDF can change the array in the calling program.

To pass an entire array by reference to a UDF, you must set UDFPARMS to REFERENCE or preface the array name with an AT symbol (@).

If UDFPARMS is set to VALUE or the array name is in parentheses, only the first array element is passed to the UDF, and it is passed by value. (VALUE is the default.)

The following sample program creates a three-element array named MULTIPLY. The entire array is passed to a UDF called PRODUCT that multiplies the first array element by the second element and then puts the result in the third array element.

After the PRODUCT routine is executed the MULTIPLY array is displayed. Because the array was passed by reference, changes made by PRODUCT to the array are made to the MULTIPLY array in the calling program.

| | | |
|---|---|--|
| DIMENSION multiply(3) | ← | Create an array. |
| STORE 2 TO multiply(1) | ← | Multiplicand |
| STORE 4 TO multiply(2) | ← | Multiplier |
| STORE 0 TO multiply(3) | ← | Product |
| | | |
| = product(@multiply) | ← | Do the product routine with entire array. |
| DISPLAY MEMORY LIKE multiply | ← | Display the contents of the array. |
| | | |
| PROCEDURE product | ← | Routine called by the program. |
| PARAMETER localarray | ← | LOCALARRAY references MULTIPLY array. |
| localarray(3) = localarray(1) * localarray(2) | ← | Product |

When you DO a program and pass an array to the program using the WITH clause, the array is passed by reference unless you enclose it in parentheses. The setting of UDFPARMS does not affect the DO WITH list.

Transferring Data Between Arrays and Databases

The following FoxPro commands help you transfer data from a database to an array:

- SCATTER transfers data from a single database record to an array.
- COPY TO ARRAY transfers data from a series of records to an array.
- SQL SELECT can transfer the results of a query to an array. (For details about SQL SELECT, refer to the chapter titled Using SQL SELECT in this manual.)

SCATTER and COPY TO ARRAY differ in the following respects:

- SCATTER transfers data from the current record in the current database. COPY TO ARRAY can transfer data from multiple records in the current database.
- SCATTER's BLANK option automatically creates an array having elements the same size and type as the fields in the database, but the array elements are empty.
- SCATTER's MEMVAR option automatically creates a set of memory variables having the same size, type and name as the fields in the database.

The following FoxPro commands help you transfer data from an array to a database:

- GATHER transfers data from an array to a single database record.
- APPEND FROM ARRAY adds new records to a database and fills the records with data from an array.
- SQL INSERT appends a single new record to a database and fills the record with data from an array.

GATHER, APPEND FROM ARRAY, and SQL INSERT differ in the following respects:

- GATHER transfers data from an array to the current record in the current database. Additionally, GATHER's MEMVAR option transfers data from a set of memory variables to the current database record.
- APPEND FROM ARRAY appends new records to the end of the current database then transfers data from the array to the newly-appended records.
- SQL INSERT appends a new record then transfers data from the array to the newly-appended record. Unlike GATHER and APPEND FROM ARRAY, SQL INSERT can append a record in an unselected database (a database open in a work area other than the current work area).



APPEND FROM ARRAY or SQL INSERT performs faster than APPEND BLANK followed by REPLACE, especially on a network.

For more information about transferring data between arrays and databases, refer to the descriptions of these commands in the FoxPro *Language Reference*.

Arrays and SQL SELECT

SQL SELECT is a versatile command for querying databases and can direct query results to an array.

To send SQL SELECT query results to an array, specify the INTO ARRAY clause with an array name. If the array doesn't exist, it is automatically created. If the array does exist, it is redimensioned automatically to accommodate the query results.

In the following example, SQL SELECT directs its query results to an array named RESULTS:

```
SELECT DISTINCT a.cust_id, a.company, b.amount ;
      FROM customer a, payments b ;
      WHERE a.cust_id = b.cust_id INTO ARRAY results
```

DISPLAY MEMORY LIKE results

| RESULTS | Priv | A | TEST |
|---------|------|-----------------------|----------------|
| (1, 1) | C | "000004" | |
| (1, 2) | C | "Stylistic Inc." | |
| (1, 3) | N | 13.91 (| 13.91000000) |
| (2, 1) | C | "000008" | |
| (2, 2) | C | "Ashe Aircraft" | |
| (2, 3) | N | 4021.98 (| 4021.98000000) |
| (3, 1) | C | "000010" | |
| (3, 2) | C | "Miakonda Industries" | |
| (3, 3) | N | 9.84 (| 9.84000000) |

For more information about SQL SELECT, refer to the chapter Using SQL SELECT earlier in the manual and to the description of SQL SELECT in the FoxPro *Language Reference*.

Arrays and FoxPro Controls

From arrays, you can create menu options or list items. To create menu options, use the @ ... GET – Popups command, and to create list items use the @ ... GET – Lists command. For details about these commands, refer to the FoxPro *Language Reference*.

Because menus and lists use arrays, you can change lists and menus dynamically. For example, by modifying an array, you can insert or remove menu options or list items. You can display the new menu options or list items by using the SHOW GET or SHOW GETS commands.

The following code snippet shows how to create a list from an array. In the example, first you create an array of five elements and sort it in ascending order using the ASORT() function. Then you create a list and use the elements of the array to create the options in the list.

```

CLEAR
SET TALK OFF
STORE 1 TO mchoice ← The first option is selected.

DIMENSION scrollarray(5) ← The array that creates the options.
STORE 'Apples'          TO scrollarray(1) ← First option.
STORE 'Bananas'         TO scrollarray(2) ← Second option.
STORE 'Limes'           TO scrollarray(3)
STORE 'Strawberries'    TO scrollarray(4)
STORE 'Lemons'         TO scrollarray(5)

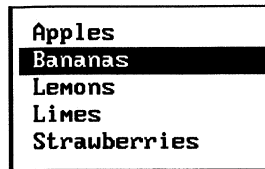
=ASORT(scrollarray) ← Sort the array in ascending order.

@2,2 GET mchoice FROM scrollarray ; ← Create the list from the array.
  SIZE 7,20 VALID scrollproc( ) ← When an option is selected from the
                                list, execute SCROLLPROC.

READ ← Activate the list.

PROCEDURE scrollproc ← This executes when an option
@12,18 CLEAR          is selected.
@12,2 SAY 'Your selection: '
@12,18 SAY scrollarray(mchoice) ← Display the selected option.
RETURN .T.
```

Here is how the list looks after an option is chosen from it:



Your selection: Bananas

10 Low-Level File Input/Output

FoxPro provides powerful low-level file functions that you can use to manipulate any type of file. Using these functions, you can create, open, read from and write to a file in any format. (The file does not have to be in FoxPro format.)

These low-level functions help you in other ways. They provide access to your computer's communication ports, and they are fast because they use the highly optimized input/output (I/O) routines that FoxPro provides.



Be careful when using low-level file functions, especially when you are manipulating files containing valuable data. Thoroughly test programs containing low-level functions on sample or backup data before using the programs with valuable data.

The following table lists the low-level functions and their uses. For more information about these functions, refer to the *FoxPro Language Reference*.

| Function | Use | Returns |
|------------|--|--|
| FCHSIZE() | Changes the size of a file. | The final file size if successful, otherwise -1. |
| FCLOSE() | Closes a file opened with FCREATE() or FOPEN(). | .T. if successful, otherwise .F. |
| FCREATE() | Creates and opens a file. | The file handle if successful, otherwise -1. |
| FEOF() | Determines if the file pointer is positioned at the end of a file. | .T. or .F. |

| Function | Use | Returns |
|-----------|---|---|
| FERROR() | Determines the success of the last low-level file function. | The error number or 0 if no error occurred. |
| FFLUSH() | Flushes a buffered file to disk. | .T. if successful, otherwise .F. |
| FGETS() | Returns a series of bytes from a file or a communications port. | Data from the file. |
| FOPEN() | Opens a file or a communication port for low-level use. | The file handle if successful, otherwise -1. |
| FPUTS() | Writes a character string, carriage return and line feed to a file or a communication port. | The number of bytes written if successful, otherwise 0. |
| FREAD() | Returns a specified number of bytes from a file or a communication port. | Data from the file. |
| FSEEK() | Moves the file pointer in a file. | The file pointer position relative to the beginning of the file. |
| FWRITE() | Writes a character string to a file or a communication port. | The number of bytes written to the file if successful, otherwise 0. |

Creating Files

`FCREATE()` creates a new file and opens the file for use.



If you try to create a file using the name of a file that already exists, the existing file is overwritten without warning. To prevent the overwriting of an existing file, use `FILE()` to see if the file exists. If the file exists, you can open it with `FOPEN()`.

If `FCREATE()` creates the file successfully, it returns a unique numeric handle to identify the file. You should store the handle to a memory variable so that you can identify the file in other low-level functions. `FCREATE()` returns -1 if it cannot create the file.

`FCREATE()` supports optional numeric arguments that specify the MS-DOS attributes of the file you create. The following table lists these arguments and their corresponding MS-DOS attributes.

| File Attributes | |
|------------------|-------------------------|
| Numeric Argument | File Attributes |
| 0 | Read/Write (default) |
| 1 | Read-Only |
| 2 | Hidden |
| 3 | Read-Only/Hidden |
| 4 | System |
| 5 | Read-Only/System |
| 6 | System/Hidden |
| 7 | Read-Only/Hidden/System |

If you open a file having the read-only attribute, you can retrieve data from the file, but you cannot modify it. When you open a file having the read/write attribute, you can retrieve data from the file, and you can write to or modify it. For additional information about file attributes, consult your MS-DOS manual.

For instance, to create the file SAMPLE.TXT with read-only/hidden attributes and save the file handle to a memory variable named SAMPLEHAND, use the following:

```
samplehand = FCREATE('sample.txt', 3)
```

You cannot open communication ports using FCREATE because FCREATE() returns -1 when the name of a port is included.

Opening Files and Ports

`FOPEN()` opens an existing file or a communication port. If the specified file or port is opened successfully, its handle is returned; if the file or port cannot be opened, -1 is returned.

You can specify the read/write privileges and the buffering scheme for the file or port by including one of the following optional numeric arguments in `FOPEN()`.

| Numeric Argument | Privileges | Buffering Scheme |
|-------------------------|---------------------|-------------------------|
| 0 | Read-Only (default) | Buffered |
| 1 | Write-Only | Buffered |
| 2 | Read/Write | Buffered |
| 10 | Read-Only | Unbuffered |
| 11 | Write-Only | Unbuffered |
| 12 | Read/Write | Unbuffered |

When you open a file with buffering, all or part of the file is stored in memory so that it can be accessed many times faster.

Because a buffered file resides in memory, the version on disk is not always current. To ensure that the current version of a file resides on disk, open the file without buffering. Whenever an unbuffered file is modified, it is written to disk. Always open communication ports without buffering.

The following command opens the file `SAMPLE.TXT` with read/write privileges and buffering, and then stores the file handle to the memory variable `SAMPLEHAND`:

```
samplehand = FOPEN('sample.txt', 2)
```

Sharing Files on a Network

On a network, you can share files opened with read-only privileges by using `FCREATE()` and `FOPEN()`. However, you cannot share files opened for exclusive use that have write or read/write privileges.

File Pointer

When a file is open, a *file pointer* designates the current byte in the file. This file pointer is similar to a database's record pointer — the record pointer designates the current record in the database.

The file pointer is always on the first byte when you open a file using `FCREATE()` or `FOPEN()`. You can move the pointer using `FGETS()`, `FPUTS()`, `FREAD()` and `FWRITE()`; you can move it to a specific position in the file by using `FSEEK()`.

A communication port does not have a file pointer. Data is sequentially read from or written to a port.

Reading from Files and Ports

When a file or port is open, you can read data from the file or port using `FREAD()` and `FGETS()`. Include the handle of the file or port in `FREAD()` or `FGETS()`.

`FREAD()` Returns a specified number of bytes from a file or port. Data returned from a file starts at the current file pointer position. The specified number of bytes are returned from a port.

`FGETS()` Returns a specified number of bytes from a file or port until a carriage return is encountered. When `FGETS()` encounters a carriage return, it stops returning data from the file and positions the file pointer on the byte immediately following the carriage return. By default, `FGETS()` returns 254 bytes from the file if no carriage return is encountered. You can specify a number of bytes other than 254. `FGETS()` ignores line feeds.

`FREAD()` sequentially returns data from a file or port, whereas `FGETS()` returns a series of lines from a file. Because many files use a carriage return to specify the end of a line, `FGETS()` is preferable to `FREAD()` for retrieving data from files in this format.

`FPUTS()` places a carriage return at the end of each line it writes to a file. If you use `FPUTS()` with `FGETS()`, you can write to and read from a file line by line.

The following example demonstrates how you can use `FGETS()` to return individual lines from a file.

```

CLOSE ALL
SELECT 0
USE customer ← Open the CUSTOMER database.
STORE RECSIZE( ) TO recordlen ← The record size can be > 254.
COPY TO custtemp.txt DELIMITED WITH BLANK ← Delimited file.

STORE FOPEN('custtemp.txt') TO custhandle ← Open the delimited file.

IF custhandle < 0 ← Can't open the file.
    WAIT 'Cannot open the file. Press a key to exit.' WINDOW
    CANCEL ← Exit this program.
ENDIF

CLEAR
DO WHILE NOT FEOF(custhandle) ← Loop through the entire file.
    @6,2 SAY FGETS(custhandle, recordlen) ← Retrieve a line.
    WAIT 'Press a key to see the next record.' WINDOW
    CLEAR
ENDDO

= FCLOSE(custhandle) ← Close the delimited file.
```

Writing to Files and Ports

The `FWRITE()` and `FPUTS()` functions write to a file or port opened with write privileges. Include the handle of the file or port in `FWRITE()` or `FPUTS()`.

`FWRITE()` Writes a specified number of bytes to a file or port. When `FWRITE()` writes to a file, writing begins at the current file pointer position. The specified number of bytes are sent to a communication port.

`FPUTS()` Writes a specified number of bytes to a file or port as `FWRITE()` does, except that each line written automatically terminates with a carriage return and line feed. If you use `FPUTS()` in conjunction with `FGETS()`, you can read and write a series of lines.

Closing Files and Ports

To preserve the integrity of data written to and read from a file or port, you must close the file or port properly. To close a file opened with `FCREATE()` or `FOPEN()`, include the file's handle in `FCLOSE()`. If the file is closed properly, `FCLOSE()` returns `.T.`, and the file handle is released.

You can also use `CLOSE ALL` to close *all* files opened with `FCREATE()` or `FOPEN()`. However, `CLOSE ALL` closes all file types in all work areas, all program and text files, and certain FoxPro system windows.

Exiting FoxPro with `QUIT` also closes files opened with `FCREATE()` or `FOPEN()`.

Additional Commands and Functions for Low-Level I/O

Other Useful Functions

Several other low-level functions are useful for manipulating files:

FCHSIZE() Changes the size of a file opened with write privileges. For example, you can use **FCHSIZE()** to truncate a file to length 0.

FEOF() Returns a true value (.T.) if the file pointer is at the end of a file. **FEOF()** always returns true if you specify a port.

FERROR() Determines the success of the last low-level file function executed. **FERROR()** returns 0 if the previous low-level function executed successfully, and it returns a non-zero number if an error occurred. This function is useful in error handling routines.

FFLUSH() Flushes the buffered portions of a file to disk. Files opened with buffering are stored in memory to increase performance. Flushing the buffers to disk ensures that the current version of the file resides on disk.

FSEEK() Moves the file pointer within a file. **FSEEK()** has no effect on ports.

HEADER() Returns the size of a database file's header. You cannot use **HEADER()** with a database opened with **FCREATE()** or **FOPEN()**, but you can use the value returned by **HEADER()** with **FSEEK()** to position the file pointer on the first byte of the first field and record in a database.

Useful Commands

DISPLAY STATUS and **LIST STATUS** return the following information about open files and ports:

- The drive, directory and file name for each open file
- The handle number of each open file and port
- The file pointer position in each open file
- The Read/Write attributes of each file and port

Low-Level Access to Communication Ports

You can use FoxPro low-level file functions to access communications ports such as COM1 and COM2. By gaining access to ports, you can read from or write to modems and external devices.

Before you access a port, you must initialize it using the MS-DOS MODE command. MODE specifies the parameters (baud rate, parity, data bits, stop bits and retries) for the port. You can execute MODE before starting FoxPro, or you can issue RUN MODE in the Command window or within a program.

The following code snippet illustrates how to dial a phone number by modem from within FoxPro. This example uses standard Hayes[®] modem commands to initialize the modem, dial the number and hang up the modem. Because of differences among modems and phone systems, this program might require changes to work properly for you.

```

SET ESCAPE ON
STORE '5551212' TO phonenumber ← The number to dial.
RUN MODE COM2:1200,N,8,1 ← Initialize the COM2 port.

modemhandl = FOPEN('COM2',12) ← Open the COM2 port.

IF ERROR( ) # 0 ← Can't open COM2.
    WAIT 'Cannot open COM port. Press any key to exit.' WINDOW
    RETURN
ENDIF

= FPUTS(modemhandl, 'ATDT' + phonenumber) ← Send the phone number.

WAIT 'Press any key to disconnect the modem.' WINDOW
= FPUTS(modemhandl, 'ATZ') ← Send the disconnect string.
= FCLOSE(modemhandl) ← Close COM2.

```

In this example, first the COM2 port is initialized with the MS-DOS MODE command, and the port is opened with FOPEN(). The handle returned by FOPEN() is stored to the memory variable MODEMHANDL. If the COM2 port cannot be opened, ERROR() returns a non-zero value, displays an error message in a WAIT window, and then exits the program.

If the port is successfully opened, FPUTS() sends a dialing command to the port followed by the phone number. The phone number is stored in a memory variable but could also be stored in a character database field.

11 Text Merge

FoxPro version 2.5 provides commands and functions for merging text. Using these commands and functions, you can combine text with the following text merge components:

- Contents of database fields
- Contents of memory variables
- Contents of array elements
- Results of functions
- Expressions and results of calculations

For example, you can use the DATE() function to put the current date at the top of a letter, and you can use fields from a database to put a customer's name, company and address below the date.

The following table lists text merge commands and functions and their use.

| Command or Function | Use |
|-------------------------------|---|
| \ \\ TEXT ... ENDTEXT | Outputs lines of text. |
| SET TEXTMERGE | Enables or disables the evaluation of database fields, variables, elements, functions and calculations. |
| SET TEXTMERGE DELIMITERS | Specifies delimiters that surround database fields, variables, elements, functions and calculations. |
| _TEXT | Directs output from \ \\ ENDTEXT to a file opened with a low-level function. |
| _PRETEXT | Specifies a character expression that prefaces text merge lines. |

Merging Text with Text Merge Components

To merge text and text merge components such as fields, variables, and results of functions, FoxPro must first evaluate the components. For example, if `DATE()` is at the top of a letter, FoxPro first determines the value of `DATE()`, and then produces it as output.

For FoxPro to evaluate a text merge component, three conditions must exist:

- You must have `SET TEXTMERGE ON`.
- The current text merge delimiters, which you can specify with the `SET TEXTMERGE DELIMITERS` command, must surround the component.
- The component must be within a `TEXT ... ENDTEXT` block or on a line beginning with `\` or `\\`.

Enabling Text Merge with `SET TEXTMERGE`

When you start FoxPro, the default for `TEXTMERGE` is `OFF`. To enable the evaluation of text merge components, use the command `SET TEXTMERGE ON`.

Suppose you use field names from a database in a form letter. When you have `SET TEXTMERGE ON` and the field names are surrounded by text merge delimiters, the *contents* of the fields merge with the text of the letter. If delimiters do not surround the field names, the names — not their contents — merge with the text of the letter.

If you have `SET TEXTMERGE OFF`, FoxPro does not evaluate text merge components but instead outputs them literally (along with the surrounding delimiters). In the previous example, the *field names* and their delimiters merge with the text of the letter.

Specifying Text Merge Delimiters

For FoxPro to evaluate text merge components, they must be surrounded by *text merge delimiters*. When you start FoxPro, the default delimiters are sets of double angle brackets (`<<` and `>>`).

With the `SET TEXTMERGE DELIMITERS` command, you can specify the characters you want to use for text merge delimiters, or you can restore the default delimiters. For more information about `SET TEXTMERGE DELIMITERS`, refer to the *FoxPro Language Reference*.

Specifying Text Merge Blocks

In addition to satisfying the previous requirements, text merge components must be within a TEXT ... ENDTEXT block or must be on a line beginning with \ or \\.

Using TEXT ... ENDTEXT

The following example illustrates how SET TEXTMERGE, text merge delimiters and TEXT ... ENDTEXT operate together in a text merge.

In this example, first TEXTMERGE is SET ON, enabling the evaluation of functions and fields. Then, SET TEXTMERGE DELIMITERS TO restores the text merge delimiters to the default set of double angle brackets (<< and >>). Finally, the CUSTOMER database is opened, and TEXT begins the evaluation of the text merge components.

All output goes to the screen. The value of DATE() is output, followed by the contents of the CONTACT, COMPANY and address fields from the CUSTOMER database. Then, the body of the letter follows as text.

| | | |
|-----------------------------|---|--|
| CLEAR | ← | Clear the screen. |
| SET TEXTMERGE ON | ← | Enable the evaluation of fields, functions, etc. |
| SET TEXTMERGE DELIMITERS TO | ← | Restore the default delimiters (<< >>). |
| SELECT 0 | | |
| USE customer | | |
| SCAN | ← | Traverse the database. |
| TEXT | ← | Start merging text and text merge components. |

<<DATE()>>

<<ALLTRIM(PROPER(contact))>>

<<ALLTRIM(PROPER(company))>>

<<ALLTRIM(PROPER(address1))>>

<<ALLTRIM(PROPER(address2))>>

<<ALLTRIM(PROPER(city))>>, <<ALLTRIM((state))>> <<ALLTRIM(zip)>>

Dear <<ALLTRIM(PROPER(contact))>>,

Thank youfor your interest in our product. The literature you requested is on its way!

Sincerely,

Microsoft Corporation

ENDTEXT

Merging Text with Text Merge Components

| | | |
|-------------|---|-------------------------------|
| WAIT WINDOW | ← | Pause before the next record. |
| CLEAR | ← | Clear the screen. |
| ENDSCAN | ← | Move to the next record. |
| USE | ← | Close the database. |

The example uses a SCAN ... ENDSCAN loop to move through the database and WAIT WINDOW to pause program execution before moving to the next record.

Here is the output from the first record:

12/10/92

N. Baker
Datatech Inc.
480 Village St.
Suite 102
San Rolfos, CA 10514

Dear N. Baker,

Thank you for your interest in our product.
The literature you requested is on its way!

Sincerely,

Microsoft Corporation

If this example is modified so that TEXTMERGE is SET OFF or the TEXT ... ENDTEXT command is removed, the output looks like this:

```
<<DATE( )>>

<<ALL/TRIM(PROPER(contact))>>
<<ALL/TRIM(PROPER(company))>>
<<ALL/TRIM(PROPER(address1))>>
<<ALL/TRIM(PROPER(address2))>>
<<ALL/TRIM(PROPER(city))>>, <<ALL/TRIM(state)>> <<ALL/TRIM(zip)>>

Dear <<ALL/TRIM(PROPER(contact))>>,

Thank you for your interest in our product.
The literature you requested is on its way!

Sincerely,

Microsoft Corporation
```

Using \ | \

In the previous example, TEXT ... ENDTEXT specifies where the text merge starts and ends. Instead of using TEXT ... ENDTEXT, you can use single (\) or double backslashes (\\) to specify where the merge starts and ends.

If the first character of a program or Command window line is \, FoxPro evaluates the content of the line as if it were within a TEXT ... ENDTEXT block. Before evaluating the line, FoxPro sends a line feed and carriage return as output. If you put \ at the beginning of the line, FoxPro evaluates the line as if it were within a TEXT ... ENDTEXT block but does not send a line feed and carriage return.

Merging Text with Text Merge Components

Here is the previous example with \ and \\ replacing the TEXT ... ENDTXT command. The output is identical. Note that \\ is used to place the contents of the STATE and ZIP fields on the same line as the contents of the ADDRESS2 field.

```
CLEAR ← Clear the screen.
SET TEXTMERGE ON ← Enable the evaluation of fields, functions, etc.
SET TEXTMERGE DELIMITERS TO ← Restore the default delimiters (<< >>).
SELECT 0
USE customer
SCAN ← Traverse the database.

\<<DATE( )>>
\
\<<ALLTRIM(PROPER(contact))>>
\<<ALLTRIM(PROPER(company))>>
\<<ALLTRIM(PROPER(address1))>>
\<<ALLTRIM(PROPER(address2))>>
\<ALLTRIM(PROPER(city))>>,
\\ <<ALLTRIM((state))>>
\\ <<ALLTRIM(zip)>>
\
\Dear <<ALLTRIM(PROPER(contact))>>,
\
\Thank you for your interest in our product.
\The literature you requested is on its way!
\
\Sincerely,
\
\Microsoft Corporation
\
WAIT WINDOW ← Pause before the next record.
CLEAR ← Clear the screen.
ENDSCAN ← Move to the next record.
USE ← Close the database.
```


Indenting and Prefacing Text Merge Lines with _PRETEXT

`_PRETEXT` is a system memory variable that you can use to indent and preface lines beginning with `\` and `\\`. For example, you can store tabs to `_PRETEXT` to create indentation in program templates. When you store a character expression to `_PRETEXT`, that expression is output before the text or text merge components.

For more information about `_PRETEXT`, refer to the System Memory Variables chapter in the *FoxPro Language Reference*.

Text Merge and Memo Fields

In memo fields, you can include text for merging. For example, a memo field can contain field names, memory variables, functions or expressions surrounded by the current text merge delimiters. For the merge to work, you must put text merge delimiters around the memo field name.

In the following example from the ORGANIZER application, the `DATE()` and `TIME()` functions are surrounded by text merge delimiters and replace the NOTES memo field in the CLIENTS database. After this replacement, FoxPro evaluates the memo field.

```
SELECT 0
USE clients
CLEAR
REPLACE notes WITH '<<DATE( )>> <<TIME( )>>'
\<<notes>>
USE
```

To see how the text merge delimiters affect the output, try this example after removing the delimiters from the functions in the memo field, then try it again after removing the delimiters from the memo field name and compare the output.

Directing Output to Windows and Files

By default, output from a text merge goes to the desktop. By including options with SET TEXTMERGE, you can suppress output to the desktop and direct output to other windows and files. For instance, using the _TEXT system memory variable you can direct output to files opened with FCREATE() or FOPEN().

Output to the Desktop

To suppress text merge output to the desktop, include the NOSHOW option with SET TEXTMERGE:

```
SET TEXTMERGE NOSHOW
```

To display output on the desktop again, include the SHOW option with SET TEXTMERGE:

```
SET TEXTMERGE SHOW
```

Output to a Window

You can direct output from a text merge to a window by including the WINDOW clause in SET TEXTMERGE. You can direct text merge output to a window that is not active or visible. After directing output to a window, you can use the NOSHOW or SHOW option with SET TEXTMERGE to suppress or enable output to the window.

The following code snippet creates a window named Datetime and uses text merge to display the current time and date in the window. The date and time are sent to the window before it is visible and active. Then ACTIVATE WINDOW Datetime activates and displays the window.

```
DEFINE WINDOW datetime FROM 2,2 TO 10,41 FLOAT CLOSE
SET TEXTMERGE ON WINDOW datetime
\
\Today's date:
\\ <<DATE( )>>
\
\The time:
\\ <<TIME( )>>

ACTIVATE WINDOW datetime
```

Output to a File

Instead of directing text merge output to a window, you can direct it to a file by doing one of the following:

- Including the name of a file in the TO <file name> clause of SET TEXTMERGE.
- Storing the handle of a file opened with FCREATE() or FOPEN() to the _TEXT system memory variable. When you include a file name in SET TEXTMERGE, the file is opened as a low-level file and its file handle is stored to _TEXT. As a result, you can manipulate the file using low-level file functions. See the chapter titled Low-Level File Input/Output for more information about FoxPro low-level file functions.

To close a file opened with SET TEXTMERGE, issue the command SET TEXTMERGE TO without a file name, or include the file handle in FCLOSE().

If you open a file with FCREATE() or FOPEN() and store its handle to _TEXT, when TEXTMERGE is SET ON the text merge output goes to the file.

Program Templates and Programs

Because FoxPro programs are text files, you can create new programs with text merge and use commands and functions as text merge components.

The following example demonstrates how to create a program template using FoxPro text merge capabilities, commands and functions. The example uses two procedures from the GENSCRN program, which creates screen program code from information stored in a screen database. The first procedure creates the program code for push buttons, and the second creates program code for radio buttons.

These procedures use fields from the screen database to supply information for the commands GENSCRN creates. The names of fields in the screen database are enclosed by the default text merge delimiters (<< and >>). For example, <<VPOS>>, <<HPOS>>, and <<NAME>> are fields from the screen database, and the fields' contents are used to create the commands.

*
 * GENPUSH - Generate Push buttons.
 *
 * Description:
 * Generate code to display push buttons exactly as they appear
 * in the painted screen(s).
 *

PROCEDURE genpush

```
\@ <<Vpos>>,<<Hpos>> GET <<Name>> ;  
  \      PICTURE <<Picture>> ;  
  \      SIZE <<Height>>,<<Width>>,<<Spacing>> ;  
  \      DEFAULT <<Initialnum>>
```

*
 * GENRADBUT - Generate Radio Buttons.
 *
 * Description:
 * Generate code to display radio buttons exactly as they appear
 * in the painted screen(s).
 *

PROCEDURE genradbut

```
\@ <<Vpos>>,<<Hpos>> GET <<Name>> ;  
  \      PICTURE <<Picture>> ;  
  \      SIZE <<Height>>,<<Width>>,<<Spacing>> ;  
  \      DEFAULT <<Initialnum>>
```

Here's an example of the screen program code these procedures create. This code was generated from the CONVMENU.SCX screen.

```
@ 3,38 GET gethelp ;  
  PICTURE "@*VN \<Help" ;  
  SIZE 1,8,1 ;  
  DEFAULT 1 ;
```

```
@ 5,38 GET exit ;  
  PICTURE "@*HT \!OK" ;  
  SIZE 1,8,5 ;  
  DEFAULT 1 ;
```

```
@ 1,18 GET unitttype ;  
  PICTURE "@*RVN \<Area>;\<Length>;\<Mass>;\<Speed>;\<Temperature>;\<Time>;\<Volume" ;  
  SIZE 1,15,0 ;  
  DEFAULT 1
```


12 Customizing Help

While using FoxPro, you can get help instantly. For example, while using a particular menu or dialog you can get help with it by pressing F1. This kind of help is called *context-sensitive* help, and this chapter explains how to create context-sensitive help for your own applications.

The topics covered in this chapter are:

- Getting context-sensitive help
- Understanding FOXHELP
- Specifying a help database
- Using help filters
- Specifying the help window's location
- Using help filter codes

Getting Context-Sensitive Help

You can get context-sensitive help for a menu, window, dialog, or topic. To get help for a menu, window, or dialog, press F1 when the menu is displayed or when the window or dialog is frontmost. You can also ALT+click on the appropriate menu, window or dialog. To get help about a particular topic, enter an appropriate help command in the Command window:

```
HELP <topicname>
```

For custom applications, you can create context-sensitive help based upon the following conditions:

- A specific topic name
- The screen field READ at the moment
- The program executing
- Conditions specified in SET HELPFILTER
- Any combination of the previous conditions

Understanding FOXHELP

This section explains the design of FoxPro's default help file, FOXHELP. You can pattern your help file after FOXHELP, or you can create it differently.

Because a help file is a database, you can create custom help by creating a new database or by copying and changing an existing one such as FOXHELP.

To view or edit the FOXHELP database, first SET HELP OFF. Then, open and browse FOXHELP as you do other FoxPro databases.

The structure of FOXHELP appears in the table below.

| FOXHELP Structure | | |
|-------------------|------------|--|
| Field | Field Type | Field Description |
| TOPIC | Character | The list of topics displayed in the Help window. |
| DETAILS | Memo | The information about a topic. |
| CLASS | Character | Codes used to filter help topics. |

A table of the codes that appear in the CLASS field is at the end of this chapter.

Help Database Requirements

Help databases hold a maximum of 32,767 records and must contain at least two fields:

- The first field must be a character field. This is the topic name that appears in the Topics panel of the Help window.
- The second field must be a memo field. This field holds the information displayed in the Details panel of the Help window.

Beyond these requirements, you can add as many additional fields as you wish. Adding fields to your help database provides greater latitude for determining which help text to display for any given context.

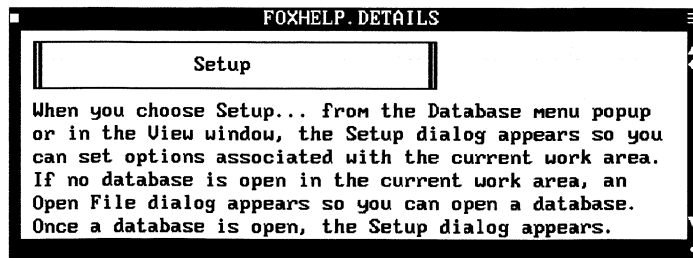
FOXHELP Topics

In the Topics Panel of the Help window, FOXHELP includes general topics, interface topics, and command/function/system memory variable topics. General topics are preceded by a triangular marker (►), formed by pressing ALT+16. Interface topics are preceded by a square bullet (■), formed by pressing ALT+254. Other topic types do not have symbols preceding the topic names.

FOXHELP Details

Help information for a topic goes in the memo field named DETAILS. The following illustration shows the memo window from a FOXHELP memo field, after you SET HELP OFF.

After you enter information for a topic and close the memo window, FoxPro adds the information to the table.



Memo Window for the FOXHELP Database

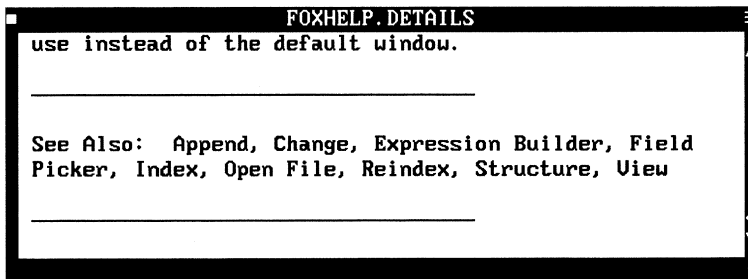
FOXHELP Cross References

“See Also” references appear at the end of the help information for each help topic. These references appear on the **See Also** popup and act as direct links to related topics. The topics that appear on the **See Also** popup conform to the following rules.

To create a cross reference, include the following at the end of the memo field:

- The phrase See Also, followed by a colon and optional spaces.
- A comma-delimited list of the topics that you want to appear on the **See Also** popup.
- A carriage return and blank line to signal the end of the list.

Case does not matter in the See Also list, and FoxPro trims leading and trailing spaces from each referenced topic. For example, cross references for the Setup topic appear in the following figure.



See Also References in the Setup Topic

Interface topics in the See Also list do not begin with a square bullet. When a See Also list appears at the end of an interface topic, FoxPro automatically prefaces each topic with a square bullet (■) before searching the help topics for a match.

To locate a See Also topic, FoxPro tries to match the topic with the first help topic that consists of, or begins with, the text string in the See Also topic.

In FoxPro, interface help topics point to other interface topics only. Thus, you cannot find cross references to a command or function name in the See Also help list for a window, dialog or menu.

Tailoring the Help Display

When you choose **Help...** from the **System** menu or type `HELP` in the Command window, the Help window appears, showing the Topics panel. You can scroll through the list to find the desired topic, or you can type a letter to move to the first topic starting with that letter. Choosing a topic displays information about the topic.

Specifying a Help Database

By default, FoxPro uses the `FOXHELP` database, but you can use another table for help by executing the following command in a program or by typing it in the Command window:

```
SET HELP TO <filename>
```

This closes the current help database and opens `<filename>` as the new help database.

To restore `FOXHELP` as your help database, issue one of the following commands:

```
SET HELP TO FOXHELP
```

```
SET HELP TO
```

Narrowing Displayed Help Topics

After specifying the help database with `SET HELP TO`, you can narrow the topic selection by using the `SET TOPIC` or `SET HELPFILTER` commands, as shown in the following examples:

- For topic selection by topic name
`SET TOPIC TO <expC>`
- For topic selection based on a logical expression
`SET TOPIC TO <expL>`
- For a subset of help topics
`SET HELPFILTER TO <expL>`

Topic Selection by Topic Name

SET TOPIC TO <expC> is the simplest way to specify which help text to display. When you request help, FoxPro evaluates the topic name <expC> and then searches the help table to find the record whose TOPIC field, matches <expC>. The search is case insensitive.

When FoxPro finds a match, it displays the contents of the memo field as help text.

For example, to display help information about the Customers topic when you choose the **Help...** push button on the CUSTOMERS screen, issue the following commands:

```
SET TOPIC TO 'CUSTOMERS'  
HELP
```

Alternatively, you can select a help topic by issuing a command in the form HELP <topic name>:

```
HELP Customers
```

You must include the square bullet (and a space), if ■ appears at the beginning of the topic name in the help database, as shown in the following example:

```
HELP ■ Customers
```

Topic Selection Based on a Logical Expression

If you issue the HELP command after you SET TOPIC TO <expL>, FoxPro evaluates <expL> and then displays help information about the first record in the help database for which the expression is true.

Suppose you want to provide help for data-entry fields in the CUSTOMER database. For instance, when users enter data into the COMPANY field, the help text would help them enter the proper data in the field.

To create this help text, first you add the names of each COMPANY field to a new field named GETFIELD in the HELPINFO database. Then, you add the help text associated with each field to the memo field HELPTTEXT.

After creating the help text and updating the memo field, activate your help facility by issuing the following commands:

```
SET HELP TO helpinfo
SET TOPIC TO UPPER(GETFIELD) = UPPER(VARREAD())
```

where `UPPER(GETFIELD) = UPPER(VARREAD())` is the logical expression.



`UPPER()` ensures a case-insensitive search.

At the beginning of the input routine, typically you use an `ON KEY LABEL` statement to assign a context-sensitive help routine to the F1 key (or another key):

```
ON KEY LABEL F1 HELP
```

`ON KEY LABEL` immediately traps the specified key and then executes the command.

`ON KEY LABEL` has other uses. For example, you can follow `ON KEY LABEL` with `SET TOPIC TO <expL>` to locate a specific field value, and you can use `ON KEY LABEL` to trap for a mouse click that activates help, as shown in the following example:

```
ON KEY LABEL LEFTMOUSE HELP
```

This traps for a click of the left mouse button. For more information about `ON KEY LABEL`, including a table of the Key Label Assignments you can use with `ON KEY LABEL`, see the *FoxPro Language Reference*.

You can also use the `PROGRAM()` function to return the name of the executing program to your help routine. See the *FoxPro Language Reference* for information about `PROGRAM()`.

Note that specifying `SET HELP TO <filename>` ensures that when a user presses F1, your help table `<filename>` will be used. And pressing F1 in a given window will then display help for the current window name, if `SET TOPIC` has not been issued.

Displaying A Subset of Help Topics

The `SET HELPFILTER` command displays a subset of help topics. The purpose of this command is to narrow the list of help topics displayed, rather than to focus on a specific topic name. Its syntax is:

```
SET HELPFILTER [AUTOMATIC] TO <expL>
```

The logical expression `<expL>` filters the help topics. Only the help records that meet the condition specified by the expression `<expL>` appear in the Help window.

Issuing `SET HELPFILTER TO` with no argument removes the help filter condition. If you include `AUTOMATIC` before `TO` in the command, FoxPro removes the help filter condition immediately after closing the Help window.

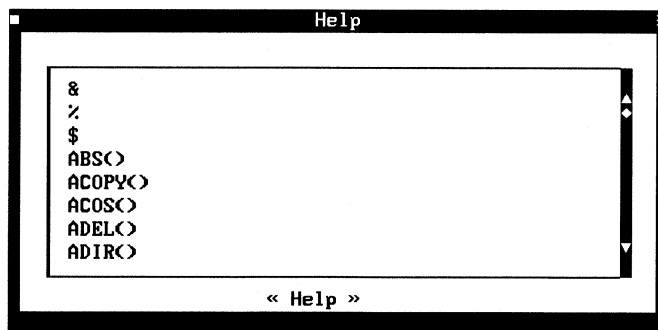
The filter expression `<expL>` typically includes a field from the help table, and you can include any field from the help database in the expression.

HELPFILTER Examples

Using the `FOXHELP` database, request help about FoxPro functions only. (In the table, a function has a topic whose `CLASS` field includes the term “Function.”)

```
SET HELPFILTER TO 'Function' $ class
HELP
```

The following figure illustrates the Help topics displayed. (Notice that only functions appear.)



Restricted Help Topics List

You can include many conditions in a `HELPFILTER` expression, and you represent these conditions as `CLASS` codes such as “Function” or “nu.” For example, the following command restricts the display to a list of numeric Functions:

```
SET HELPFILTER TO 'Function' $ class AND 'nu' $ class
```

See the end of this chapter for a complete list of CLASS codes used in the FOXHELP database. You can also try the program HELPTREE.MPR located in the GOODIES\HELPTREE subdirectory. When you run this program, it displays the CLASS codes as filter options on a menu, and you can select the appropriate options from the menu.

To remove all filter conditions manually, issue the following command:

```
SET HELPFILTER TO
```

Alternatively, you can remove the filters automatically, after the Help window closes, by including AUTOMATIC before TO, as shown in the following example:

```
SET HELPFILTER AUTOMATIC TO 'Function' $ class
```

Context-Sensitive Help in Windows

When a user presses F1 in a window of your application, FoxPro displays context-sensitive help for the window title, as specified with the TITLE clause. For example, if the window title is "CUSTOMER", FoxPro displays help information for the topic "CUSTOMER". (This assumes that the F1 key has not been redefined.)

If a window has no title, FoxPro displays context-sensitive help for the window name (with the leading and trailing blanks trimmed).

Controlling the Location of the Help Window

The location and size of the Help window displayed depend on the location and size of the Help window when it was last open. You cannot explicitly specify the location of a FoxPro system window such as the Help window.

To specify a location for custom help, you must create your own window using the DEFINE WINDOW command. With this command, you can specify the size and location of the window. Then, you can display the window by activating or showing it, as explained in *FoxPro Language Reference*.

For example, the following commands define a window named RED and activate the HELPER help database in that window:

```
DEFINE WINDOW red FROM 1,1 TO 35,60 COLOR SCHEME 7
SET HELP TO helper.dbf
ACTI WINDOW red
HELP IN WINDOW red
```

Grander Schemes

Because you can add any number of fields to a help database, and because you can use any logical expression to select help topics, only your imagination limits the kind of help system you can create.

For instance, you can:

- Define one or many program variables that control the behavior of your help system and then assign values to these variables based upon the operating mode of your program.
- Provide more detail in help files for novice users than you provide in files for experienced users.
- Permit users to access help only if they enter an appropriate password.

You can develop much more elaborate help systems using the FoxPro Menu Builder and Screen Builder. For information about these power tools, see the appropriate chapters in the FoxPro *User's Guide*.

Help File Codes

The following table lists the FOXHELP help filter categories and their corresponding two-letter codes, which are used in the CLASS field of the FOXHELP database. You can use these codes to filter help topics.

Note that if a HELPFILTER is SET, topics on the **See Also** popup must meet the filter condition for the associated help text to display. If one does not meet the filter condition, choosing it from the popup displays a “No help found for...” message.

| Help Filter Category | Code |
|---|----------|
| General | General |
| What's New | wn |
| Compatibility | cm |
| Configuration | cf |
| Error Messages | em |
| Commands | Command |
| Table (Fields, Indexes, Records, Relations) | db |
| Enhanced in FoxPro version 2.5 | ex |
| Environment (SET Commands, Screen, Keys) | en |
| Errors and Debugging | er |
| Event Handlers (ON ERROR, ON KEY) | eh |
| File Management | fm |
| Indexing | ix |
| Keyboard and Mouse | km |
| Memory Variables and Arrays | mv |
| Menus and Popups (System Names) | mp |
| New in FoxPro 2.5 | nx |
| Printing | pr |
| Program Execution | pe |
| Structured Programming | sp |
| SQL | sq |
| Text Merge | tm |
| Windows | wi |
| Interface | in |
| Multi-User | mu |
| Functions | Function |
| Character | ch |
| Numeric | nu |

| Help Filter Category | Code |
|---|-----------|
| Date and Time | dt |
| File I/O | fi |
| Logical | lo |
| Table (Fields, Indexes, Records, Relations) | db |
| Data Conversion | dc |
| Environment | en |
| File Management | fm |
| Keyboard and Mouse | km |
| Low-Level | ll |
| Memory Variables and Arrays | mv |
| Menus and Popups (System Names) | mp |
| Windows | wi |
| Printing | pr |
| Multi-User | mu |
| System Memory Variables | Sysmemvar |
| Desk Accessories | da |
| Printing | pr |
| Interface Operations | in |
| Text Merge | tm |
| System Menu Names | sn |
| Interface | Interface |
| Dialogs | di |
| General | ge |
| Menus | me |
| Windows | wi |

13 Documenting Applications with FoxDoc

FoxDoc is an automatic application documenter for FoxPro programs. With FoxDoc, documenting an application becomes a simple matter of entering some basic information and pressing a few keys.

FoxDoc maps the flow of an entire FoxPro project, application or a single program, producing a complete documentation package based on your specifications.

Overview

FoxDoc makes documenting FoxPro programs a snap by producing technical documentation for an entire application, including:

- System summary showing lines of code, file statistics, etc.
- Variable cross-reference report
- Tree structure of the system. The tree optionally can show databases, index files, etc., used by each program
- List of all files in the system
- Database summary
- Index, format file, label form, report form, screen file, menu file and procedure file summary
- Formatted source code listings
- Action diagrams
- Batch files to back up programs, databases, etc.
- Batch file to move output files back to the source subdirectory

In addition, FoxDoc can write a heading on each program file that includes the following information:

- Project, application and/or program name
- Author and copyright notice
- Which files call this program and which files this program calls
- Databases and indexes used
- Format files, report forms and memory files used
- Date and time last modified

If you wish, the source code headings can also be echoed to a separate file.

FoxDoc can indent your source code and capitalize FoxPro keywords to make your code easier to read, understand and maintain.

FoxDoc documentation is application-wide. In other words, not only can it tell you where variable X was used in a particular program, it can cross-reference all occurrences of variable X anywhere in the application you are documenting. You merely enter the name of the main program or the project file name and FoxDoc does the rest. Of course, you can also document a single program.

Getting Started

This section is divided into two parts. The first part shows you how to move around inside of FoxDoc and the second part takes you through a quick documentation session.

FoxDoc Files

The FoxDoc system is composed of the following files:

| | |
|--------------|--|
| FOXDOC.EXE | main program file |
| FOXDOC.HLP | help file |
| PROWORDS.FXD | file of FoxPro keywords |
| FXPWORDS.FXD | file of FoxBASE+ [®] keywords |
| CONFIG.FXD | configuration file (optional) |

With the exception of the optional configuration file, all of these files must reside together in the FoxPro home directory, which may be different from the subdirectory where your FoxPro source code files are stored.

Moving Around In FoxDoc

FoxDoc is very easy to use. You can use your mouse or your keyboard to guide you through its various operations and eight option screens.

FoxDoc can be used with a mouse. You can select any input area, Function Key options at the bottom of the screen, or options on the Main Menu by pointing to them and pressing the left mouse button. Pressing the right mouse button displays the Main Menu.



Ctrl+End deletes everything on the current line and Ctrl+D duplicates what's on the previous line.

Function Key Options

The following Function Key options appear the bottom of each screen. Each function key brings up an associated dialog or menu bar.

F1-Help Press the F1 key or choose **F1-Help** at the bottom of the screen at any time to receive context-sensitive help — information that's appropriate to the screen or field that the cursor is currently positioned on. To get information about any input field, simply position the cursor on the field and press F1.

F5-Save Defaults

To save the values of all option screen fields for use as defaults, press F5 or choose **F5-Save Defaults**. FoxDoc will prompt you for a file name before the defaults are saved. See the Changing, Saving and Restoring Default section for more information.

F6-Read Defaults

Press F6 or choose **F6-Read Defaults** to manually retrieve a set of saved specifications. FoxDoc prompts you for the file name that holds the defaults. Pressing F6 at the System screen and entering a file name is equivalent to using the /F command-line parameter when starting the program. See the Changing, Saving and Restoring Defaults and Program Limitations and Miscellaneous Notes for more information.

F10-Main Menu

After you specify a main program file on the System options screen, you can display FoxDoc's main menu by pressing F10, by choosing the **F10-Main Menu** option at the bottom of the screen or by pressing Escape. Escape toggles the main menu on and off.

The main menu allows you to access all eight FoxDoc option screens, as well as to begin documenting your system and to quit from FoxDoc. Choose options from the main menu by clicking on them with the mouse or pressing the first letter of the menu choice you want. For example, you can press R to see the Report options screen, B to Begin documenting, Q to Quit, and so forth.

| | | | | | | | | | |
|--------|---------|--------|------|----------|------|-------|-------|-------|------|
| System | Reports | Format | Xref | Headings | Tree | Print | Other | Begin | Quit |
|--------|---------|--------|------|----------|------|-------|-------|-------|------|

Main Menu

If you don't want to use the main menu, you can save a few keystrokes and quickly move through the various FoxDoc option screens by pressing Ctrl+PgDn or Ctrl+PgUp. Ctrl+PgDn takes you to the next screen, while Ctrl+PgUp takes you to the previous screen.

A Quick Run Through

To show you how easy documenting your program files can be, let's take a quick run through FoxDoc.

Make a backup copy of your program files then start FoxDoc. The FoxDoc System screen appears.

This is the only screen in which you must enter information. If you accept the program's default options, you only have to enter the following information.

| | |
|--|---|
| Application Name/Author/ Copyright Holder/ Copyright Date | These four lines of information are used only in the program headings and can be omitted if you choose not to write program headings. (See FoxDoc Format and Action Diagrams Options Menu for more information.) If you enter either an author or a copyright holder, but not both, FoxDoc assumes they are the same. |
|--|---|

| | |
|--|--|
| Main Program/ Project File Name | Enter the name of the main program file in your application or the name of your project. If the file you specify cannot be found, or if the FoxDoc files are not in the directories that you specified, FoxDoc returns an error message and allows you to correct the information or exit from FoxDoc. |
|--|--|

| | |
|--------------|--|
| Paths | The four fields at the bottom of the System Menu screen contain path information to the source code, data, output and FoxDoc files. All four fields should contain valid MS-DOS paths, with or without drive designations. You can omit the final backslash as in C:\FOXDOC, C:\FOXDOC\ and \FOXDOC. |
|--------------|--|

Once you enter the system information, you can bring the main menu forward. From here, you can select another option screen, begin the documentation process, or quit from FoxDoc.

If the output directory does not exist, FoxDoc creates it.

Documenting

| Filename | Lines | Pass |
|----------------------|-------|------|
| —Loaded OK | | |
| DEMO. PRG | | 1 |
| _PQB00THDC—procedure | | 1 |
| DEMO. FMT | | 1 |
| — Starting Pass 2 — | | |
| DEMO. PRG | 91 | 2 |
| DEMO. FMT | 62 | 2 |

| Status Window | | 0:20 |
|----------------------|--------|-----------------------|
| Available memory: | 251784 | Total index files: 2 |
| Total program lines: | 153 | Total format files: 0 |
| Total program files: | 2 | Total report forms: 0 |
| Total procedures: | 1 | Total memory files: 0 |
| Total databases: | 2 | Total variables: 18 |

Press space bar to continue

Status Screen

Status Screen

As FoxDoc documents your programs, it displays a status screen. The box in the upper-left corner of the status screen shows the file that's currently being documented, the number of lines in the file and the current pass number.

FoxDoc makes two passes through each file in your system as it prepares its documentation. During the first pass, FoxDoc determines which files use or call which other files. During the second pass FoxDoc cross-references the variables, formats the source code and prepares the action diagrams.

If you choose any option that modifies the source code or if you choose to display the cross-reference report, FoxDoc displays information in the lower portion of the screen indicating the following:

- How many programs, databases, indexes, format files, report forms and variables it has found
- The total number of program lines documented
- The amount of free memory available
- The elapsed time since documentation began

If FoxDoc finds any errors during its run, it prints an error message in the Error window of the screen. Error messages are also echoed to the ERROR.DOC file.

FoxDoc System Screen

Choose System from the Main Menu to bring you to the System Screen. This screen lets you specify system-level options.

| | | | | | |
|--|--|----------------------|--|----------|--|
| 04/18/91 | | FoxDoc System Screen | | 17:07:14 | |
| <div>Application name: [REDACTED]</div> <div>Author: [REDACTED]</div> <div>Copyright holder: [REDACTED]</div> <div>Copyright date: 1991 [REDACTED]</div> <div>Main program/project filename: [REDACTED]</div> <div>Path for program source code: C:\FOXPRO25 [REDACTED]</div> <div>Path for data files: C:\FOXPRO25 [REDACTED]</div> <div>Path for output files: C:\FOXPRO25 [REDACTED]</div> <div>Path for FoxDoc files: C:\FOXPRO25 [REDACTED]</div> | | | | | |

FoxDoc System Screen

For information about setting the defaults for this screen refer to Changing, Restoring and Saving Defaults later in this chapter.

Application Name

Enter the name of your project, application, or program.

Author

Enter your name here. If you enter an author or copyright holder but not both, FoxDoc assumes they are the same.

Copyright Holder

Enter the copyright holder information here. If you enter an author or copyright holder but not both, FoxDoc assumes they are the same.

Copyright Date

Enter the copyright year.

Main Program/Project File Name

This field holds the name of the first file in an application or the name of a project. If you enter a file name without an extension, FoxDoc searches first for a project file with that name, then for a program, screen or menu. This file is the main program file which starts your application.



This field must contain the name of a file that exists in the program source code directory before you can access the main menu or any of the other option screens.

If you specify a project, FoxDoc searches the project file for the “main” program and documents it first.

FoxDoc assumes that you want to document not only the file listed here, but all programs it calls, all programs called by programs that this file calls, and so on. Unless otherwise specified in the Other Options screen, FoxDoc searches the program tree for all programs, databases, index files, report forms, format files, label forms and memory files as it prepares system documentation.

Path for Program Source Code

Enter the complete path name where your source code is located. The program path you enter should be a valid MS-DOS path, with or without a drive designation. The final backslash can be omitted. For example, C:\FOXDOC, C:\FOXDOC\ and \FOXDOC are all valid paths.

FoxDoc can find programs in multiple subdirectories. You do have to tell FoxDoc which subdirectory to search, however. Use the `*#FOXDOC PRGPATH` path name direction to specify the paths to search for program files (programs, procedure files, format files, and so forth). See Other FoxDoc Directives later in this chapter for more information. If you are using a project file, FoxDoc will search these subdirectories automatically.

Path for Data Files

Enter the complete path name where your data files are located. The path you enter should be a valid MS-DOS path, with or without a drive designations. The final backslash may be omitted. For example, C:\FOXDOC, C:\FOXDOC\ and \FOXDOC are all valid paths.

FoxDoc can find data files in multiple subdirectories. You do have to tell FoxDoc which subdirectory to search, however. Use the `*#FOXDOC DATAPATH` path name direction to specify paths to search for data files (databases, index files, memory files and so

forth). See Other FoxDoc Directives later in this chapter for more information. If you are using a project file, FoxDoc will search these subdirectories automatically.



If you use explicit drive and path designations when you reference another file, FoxDoc will find it as long as you have not told FoxDoc to ignore drive designations; otherwise, FoxDoc cannot tell which subdirectory is current. FoxDoc pays no attention to SET PATH TO commands. FoxDoc also cannot handle FoxPro macro substitutions for drives or paths.

Path For Output Files

Enter the complete path name where you'd like the output files to be placed. The path should be a valid MS-DOS path, with or without drive designations. The final backslash can be omitted, for example, C:\FOXDOC, C:\FOXDOC\ and \FOXDOC are all valid path names.

If the path for the program source code files and the paths specified for the output files are different, FoxDoc does not modify your original source code files in any way. Only the output files will contain capitalized keywords, indents, headings and so on.

If, however, the source code and output paths are the same, FoxDoc adds .BAK extensions to your original source code files and creates formatted output files with the original names. For example, if one of your input programs is named EDIT.PRG, after you run FoxDoc the original source file will be called EDIT.BAK and a new, formatted EDIT.PRG will take its place. (FoxDoc never modifies database, index, format, form or memory variable save files.)

When FoxDoc decides you are about to do something potentially dangerous, it may require you to direct output files to a different directory than the input files are in. Some dangerous options include expanding or compressing keywords and eliminating comments from source code.



If the source program directory is the same as the output directory, the first eight characters of the input file names should be unique (that's not counting the extension). If only the extensions are unique, all but the last input file will be deleted. However, the modified output files will still be there.

Path For FoxDoc Files

Enter the complete path name where the FoxDoc files are located. The path should be a valid MS-DOS path, with or without drive designations. The final backslash can be omitted, for example, C:\FOXDOC, C:\FOXDOC\ and \FOXDOC are all valid path names.

FoxDoc Report Screen

Choose Report from the Main Menu to bring you to the Report Screen. This screen contains the many report options you can specify when creating program documentation. With FoxDoc you can produce several different reports. Each report is optional and can be selected or suppressed as you wish, although some reports, for example, Data dictionary and Index summary, require you to search the tree rather than document a single program.

Displaying Reports on the Screen

By default, FoxDoc does not display reports on the screen as they are produced. All reports except for the File Headings may be echoed to the screen as they are written to disk. Enable screen echoing by invoking FoxDoc with the /S command line parameter. Screen echoing will decrease FoxDoc's speed. The /S switch only affects the screen display of reports and has no effect on the status screens that are displayed as FoxDoc scans source code files. See Program Limitations and Miscellaneous Notes later in this chapter for more information.

04/18/91

FoxDoc Report Screen

17:08:17

| | | | |
|-------------------------------------|-------------------------------------|----------|--------------|
| Write program tree structure? | <input checked="" type="checkbox"/> | Filename | TREE.DOC |
| Write list of files used in system? | <input checked="" type="checkbox"/> | Filename | FILELIST.DOC |
| Write index file summary? | <input checked="" type="checkbox"/> | Filename | NDXSUMRV.DOC |
| Write database summary? | <input checked="" type="checkbox"/> | Filename | DATADICT.DOC |
| Write format file summary? | <input checked="" type="checkbox"/> | Filename | FMTSUMRV.DOC |
| Write report form summary? | <input checked="" type="checkbox"/> | Filename | FRMSUMRV.DOC |
| Write procedures summary? | <input checked="" type="checkbox"/> | Filename | PRCSUMRV.DOC |
| Write label form summary? | <input checked="" type="checkbox"/> | Filename | LBLSUMRV.DOC |
| Write screen file summary? | <input checked="" type="checkbox"/> | Filename | SCRNSMRY.DOC |
| Write menu file summary? | <input checked="" type="checkbox"/> | Filename | MENUSMRY.DOC |
| Write binary file summary? | <input checked="" type="checkbox"/> | Filename | BINSUMRV.DOC |
| Write memory file summary? | <input checked="" type="checkbox"/> | Filename | MEMSUMRV.DOC |
| Write system statistics? | <input checked="" type="checkbox"/> | Filename | STATS.DOC |
| Write variable cross-reference? | <input checked="" type="checkbox"/> | Filename | XREF.DOC |
| Write other file summary? | <input checked="" type="checkbox"/> | Filename | OTHER.DOC |
| Prepare batch files for backups? | <input checked="" type="checkbox"/> | | |

FoxDoc Report Screen

Default settings and file names are shown in the picture.

Program Tree Structure

The Tree Structure diagram shows which programs call which other programs and which programs use which databases. The tree diagram, like other FoxDoc summaries, is written to a file and can appear on the screen.

Any procedures or databases in the tree will be designated like this:

Procedure name (procedure in procedure.file)

Database name (database ALIAS baz)

By default, FoxDoc includes programs, procedures, functions, format files and databases in the tree, and it also includes options that allow you to put indexes, report forms, label forms and memory variable save files in the tree, or to suppress any of the default file types except program files. Databases and other non-program files, are shown underneath the programs that actually call them. If a database is opened in one program and remains open in another program, FoxDoc will show the database as called only by the first program file.

FoxDoc can detect recursion routines and prevent the tree from marching off the right side of the page and eventually off the edge of the known universe.

A sample tree report is included in at the end of this chapter.

List of Files in the Application

The File List Summary is the list of files used in the application including programs, databases, index files and so on. This report can be useful for identifying which files in a directory are associated with an application. The file list can also be used by the source code printing routines at a later point, so that program files can be printed without going through a complete documentation run again.

Index File Summary

The Index File Summary lists each index file referenced in the system, the index key and the files that use the index. If FoxDoc cannot find a referenced index file, it reports that fact.

See the section titled FoxDoc File Type Identification for information about how FoxDoc identifies indexes.

A sample index file summary is shown in the back of this chapter. This report also shows structural indexes and their tags.

Database Summary

The Database Summary contains the database structure for each database in the system, a list of programs that use each database, a listing of each data field in the system and the databases that contain each field.

For information about how FoxDoc tries to determine which indexes, report forms and labels forms are associated with each database, refer to FoxDoc File Type Identification later in this chapter.

A sample database summary is included at the end of this chapter.

Format File Summary

The Format File Summary shows each format file used in the system and the programs that use it.

For information about how FoxDoc identifies format files, refer to FoxDoc File Type Identification later in this chapter.

A sample format file summary report is at the end of this chapter.

Report Form Summary

The Report Form Summary shows each report form used in the system, the report parameters and the programs that call the report form. FoxDoc shows a mock-up of each report and the expressions that go into each column. It also indicates which database is associated with the report form.

For information about how FoxDoc identifies report forms, refer to FoxDoc File Types Identification later in this chapter.

Procedures Summary

The Procedures Summary shows all of the procedures and functions that are contained in the application. This summary shows the procedures in each program/procedure file and the programs and procedures that each one calls and is called by.

Label Form Summary

The Label Form Summary shows the parameters of each label form in the system. It also indicates which database is associated with each label form.

Screen File Summary

FoxDoc watches for references to screen files in your programs or project and prepares a report showing a graphic image of the screen file.

Menu File Summary

FoxDoc watches for menu file references in your programs or project and prepares a report showing a graphic image of the menu file.

Binary File Summary

FoxDoc watches for references to binary files in your programs and prepares a report showing the binary file names and the program files that use them.

Memory File Summary

FoxDoc watches for references to memory variable save files in your programs and prepares a report showing the memory variable save file names and the program files that use them.

System Statistics

The System Summary Report shows the following information:

- System name
- Author
- Current date and time
- Lines of code
- Number of programs, procedures, procedure files, indexes, etc.

- Names of databases, indexes, report forms, label forms and memory files

The system summary is the first page in the documentation. It, together with the tree diagram, provides a basic overview of the entire system. This option usually takes one or two pages. See System Summary example at the end of this chapter.

Variable Cross-Reference

The Variable Cross-Reference catalogues all of the variables found in the system, showing line numbers for each program that references a particular variable. Variables, in this context, include field names, file names and anything else that isn't a keyword, numeric constant, punctuation mark or quoted string.

The cross-reference report adds certain codes to the end of a line number reference when the reference has particular significance. See the section titled Cross-Reference Codes later in this chapter for more information.

At the bottom of the report, FoxDoc produces a list of public variables, a list of macros and a list of arrays. The macro list is subdivided into macros that you defined to FoxDoc with DOCCODE and DOCMACRO FoxDoc coding and those you didn't. If you defined a macro, its definition is also shown. Arrays are denoted with parentheses and their declared size shown, for example, APPLE(100). For more information on DOCCODE and DOCMACRO code, see the section titled FoxDoc Commands later in this chapter.

The cross-reference report interacts closely with the keyword file. Specifically, the keyword file tells FoxDoc what is to be considered a keyword and what is not, and by inserting the correct characters in the keyword file, you can cause certain keywords or variables to be handled differently. See the section titled Keyword File List Information later in this chapter for full details.

A sample cross-reference report is included at the end of this chapter.

Other File Summary

This option allows you to display and echo to file a summary of files other than those covered by separate documentation reports used in the application. FoxDoc watches for references to these files in your programs and prepares a report showing the file name and the program files that use it.

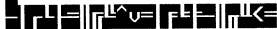
Preparing Batch Files

While FoxDoc is documenting your system, it can produce several MS-DOS batch files that can do useful things. For a listing of each .BAT file, refer to Batch Programs later in this chapter.

FoxDoc Format and Action Diagram Options Screen

Choose Format from the Main Menu to bring you to the Format and Action Diagrams Options Screen. This screen contains options that affect source code formatting and action diagrams. Action Diagrams document the structure of a program by using graphical symbols to group related statements together.

04/18/91 FoxDoc Format and Action Diagrams Options Screen 17:09:24

| | | |
|---------------------------------------|---|--|
| Create formatted source code files? | <input checked="" type="checkbox"/> | |
| Preserve original date and time? | <input checked="" type="checkbox"/> | |
| Write headings for source code files? | <input checked="" type="checkbox"/> | |
| Echo all headings to a separate file? | <input checked="" type="checkbox"/> | Filename: HEADINGS.DOC |
| Token capitalization (U/L/F/N)? | <input checked="" type="checkbox"/> L | (Up/Low/First/None) |
| Keyword capitalization (U/L/F/N)? | <input checked="" type="checkbox"/> U | Key word filename: PROWORDS.FXD |
| Tabs, spaces or no indenting (T/S/N)? | <input checked="" type="checkbox"/> S | No. of spaces: 3 |
| Align comments? | <input checked="" type="checkbox"/> N | Column: 50 |
| Indent procedures and functions? | <input checked="" type="checkbox"/> N | |
| Eliminate blank lines from output? | <input checked="" type="checkbox"/> N | |
| Eliminate comments from output? | <input checked="" type="checkbox"/> N | |
| Keyword expansion (E/C/N)? | <input checked="" type="checkbox"/> N | (Expand/Compress/None) |
| Comment control structures (Y/N/R/D)? | <input checked="" type="checkbox"/> N | (Yes/No/Replace/Delete) |
| | | |
| Create action diagrams? | <input checked="" type="checkbox"/> | |
| File extension for action diagrams? | <input checked="" type="checkbox"/> ACT | |
| Add line numbers to action diagrams? | <input checked="" type="checkbox"/> | |
| Graphics, ASCII or other characters? | <input checked="" type="checkbox"/> G | (G, A or O) |
| Symbols for action diagrams? | |  |

FoxDoc Format and Action Diagrams Options Screen

Defaults for each setting are shown in the picture.

Formatted Source Code Files

By default, FoxDoc produces formatted source code output files. These output files are a copy of each of your source code files formatted as you specify through the various options.

FoxDoc generally recognizes valid FoxPro abbreviations. For example, it will properly detect that the following statement initiates a DO loop.

```
<tab>DO          <tab> WHILE  <tab>
```

As always, it's not good practice to use keywords in a way other than that intended by the language. If you use keywords as variable names FoxDoc can become confused. For example, if you use PROC as a variable name within a procedure file and put the statement PROC = APPLE within the file, FoxDoc will see PROC and think that it is initiating a new procedure.

Original Date and Time

Unless you specify otherwise, when FoxDoc formats source code files, producing new output files, the new files have the same date and time as the original input files. If you do not preserve the original date and time, the new files will have the current MS-DOS date and time.

Headings for Source Code Files

FoxDoc will automatically write a heading to each new program file in the system. This heading includes such information as:

- Application, program and author name
- Copyright notice
- Procedures defined within this file
- Which programs this program calls
- Which programs call this program
- Databases, index files, report forms, format files, label forms and memory files used by this program
- Date and time documented

You may want to add more information to the header, such as a brief narrative description of the program's purpose.



If you choose to write headings on your source code files, or if you choose any other option that modifies the source code file, it's a good idea to send the output files to a different directory than the input files. This ensures that your original source code remains unchanged.



If input source program files do not have unique filenames (excluding the extension), and you send output files to the input directory, some of your original source code files could be destroyed.

When you send output files to the input directory, your original source code files will be renamed with a .BAK extension. If you use unique extensions to distinguish between program files, some of your original source code files could be destroyed. For example, if your system uses the following program file names:

```
SYSTEM.INP  
SYSTEM.EDT  
SYSTEM.RPT  
SYSTEM.DEL
```

and so on, the output files containing the headings will retain these names. Each of the input files, however, will be renamed to SYSTEM.BAK. Only the last one processed will still exist when FoxDoc completes. Therefore, you should always send output files to a separate subdirectory if you use this naming convention.

Echoing Headings

A copy of the headings that are written to each source code file can also be sent to a separate headings file, HEADINGS.DOC, if you wish.

Token Capitalization

Token words refer to all occurrences of words not found in the PROWORDS.SNP, FXPWORDS.SNP or FX2WORDS.SNP files. These words can be written in upper-case (U), lower-case (L), initial caps (F) or left as they appear in the original source code (N).

Keyword Capitalization

FoxDoc capitalizes all FoxPro keywords found in your source code. Keywords are stored in the following files: PROWORDS.FXD and FXPWORDS.FXD. You can also specify your own keyword file. FoxDoc does not attempt to parse code statements to determine how a word is used, so if you use keywords as variable or field names, they will also be capitalized. Should you wish not to capitalize a particular keyword, you can either delete it from the appropriate keyword file, or comment it out by putting an asterisk before it, for example, RELEASE becomes *RELEASE.

The default settings for capitalization assume that you prefer the “standard” format: keywords are capitalized, functions have an initial capital letter and everything else is in lower case. You can change these case specifications as you wish.

Keywords can be written in upper-case (U), lower-case (L), initial caps (F) or left as they appear in the original source code (N).

Keyword File Name

FoxDoc allows you to document a FoxPro or a FoxBASE+ application or program. FoxPro, however, has many keywords that FoxBASE+ does not contain. If you're documenting a FoxPro application system, use the PROWORDS.FXD file. If you're documenting a FoxBASE+ program, use the FXPWORDS.FXD file. See the section titled Keyword File List Information for information on creating your own keyword file and editing existing files.

Tabs, Spaces or No Indenting

Choose T for tabs or S for spaces to indent code underneath control structures such as DO WHILE, IF, CASE, FOR and so forth. If you choose to indent your code, one tab character, or the number of spaces you specify, will be inserted for each control statement nesting level. Choose N for no indentation.

This option also scans your source code for unmatched control structures (for example, IFs not matched with ENDIFs) and reports any discrepancies.

Align Comments

You can have FoxDoc align comments to a specified column. FoxDoc will try to make all comments "line up" on that column. If it cannot align a column, possibly because the source code extends past the specified column, FoxDoc will start the comments immediately after the source code. This option has no effect on * comments.

Indenting of Procedures and Functions

Choose this option to have FoxDoc add an indent beneath procedure and function declaration, like this:

```
PROCEDURE barn
    PARAMETERS horse,cow
    <more statements>
RETURN
```

FoxDoc does not add the extra indent underneath procedures and functions unless you choose this option.

Eliminating Blank Lines from Output

You can have FoxDoc eliminate blank lines that appear in the input files when creating formatted output files. FoxDoc's default option is to leave blank lines in. You cannot select this option if the source and output subdirectories are the same.

Eliminating Comments from Output

You can have FoxDoc eliminate comments that appear in the input files when creating formatted output files. FoxDoc's default option is to leave comments in. You cannot select this option if the source and output subdirectories are the same.

Keyword Expansion

FoxDoc can expand abbreviated keywords to their full length (E) or abbreviate the keywords to four characters (C). The default is no expansion or compression (N), meaning that FoxDoc leaves your source code the way it found it.

Be cautious with this option, especially if you're not very selective when naming variables. The only thing FoxDoc knows about keywords is that they are in the keyword file. FoxDoc cannot tell if a particular word is being used as a command keyword or as a variable. If the name of one of your variables matches a keyword or a valid abbreviation for a keyword, FoxDoc will expand or contract it along with everything else.

Problems can also arise if two variables are valid abbreviations of the same keyword. For example, "other" and "otherw", are both valid abbreviations of the keyword OTHERWISE, and both variables will compress to OTHE. Once they are compressed, there is no way to separate them again. The same holds true for user-defined function names and array names.

Because of the irrevocable changes that this option can make, FoxDoc will not allow you to exercise it when the source and output directories are the same. FoxDoc will not overwrite your original source code files if you select this option.



Keywords identified in the keyword file as functions, that is they have () after their name, are not affected by keyword compression or expansion.

Comment Control Structures

FoxDoc allows you to add comments to control structure terminators such as `ENDIF`, `ENDDO` and so on as it creates the new, formatted program files. For instance, FoxDoc can append the condition specified in an `IF` to the matching `ENDIF`:

```
IF this_expr = .T.  
    DO some_act  
ENDIF this_expr = .T.
```

This same control structure without comment would appear as:

```
IF this_expr = .T.  
    DO some_act  
ENDIF
```

With this option, you can add comments (Y), leave any existing comments alone and not add new ones (N), replace existing comments with new ones (R), or delete existing comments and not add new ones (D).

Action Diagrams

In the Format and Action Diagram Options screen you also choose whether or not to write action diagrams. Action diagrams document the structure of a program by using graphical symbols to group related statements together. These diagrams allow you to identify unusual loop exits easily, like `LOOP` and `RETURN`. You can often identify subtle, hard-to-locate bugs by studying the action diagram for the program.

Action diagram files are very much like program files and have the same capitalization as the output program files do. However, action diagram files cannot be compiled or executed because of the additional line-drawing characters they contain and the optional line numbers that can be added.

FoxDoc will identify certain syntax errors, such as a `LOOP` statement that is not within a `DO WHILE` loop, *only if you choose to prepare action diagrams*. Certain error checking is only performed within this module.

File Extension for Action Diagrams

FoxDoc allows you to specify the extension that action diagram files will have. Action diagrams always have the same main file name, that is the first eight characters, as the source code file from which it is drawn but the extension can be anything you specify except .PRG.

You can use the “?” character as a wildcard to match the corresponding characters in the original file extension. Therefore, if you choose an action file extension of “??T”, and your original source file was named MYPROG.PRG, the action diagram file will be named MYPROG.PRT: the PR is drawn from the original filename and the T comes from the action diagram file mask.

The default extensions for action diagrams are

| | |
|-----|---------------|
| ACT | for PRG files |
| AC1 | for SPR files |
| AC2 | for MPR files |
| AC3 | for QPR files |

Adding Line Numbers to Action Diagrams

By default, FoxDoc will add line numbers to the beginning of every line in the action diagram. This option allows you to quickly locate any single line of code.

Graphics, ASCII or Other Characters

Different extended ASCII characters are used to mark the different control structures. Loops (DO WHILE, FOR) use the double-line vertical bar, while conditional statements (IF, DO CASE) use a single vertical bar. The structures are distinguished further by the horizontal symbol used to connect the control structure keyword with the vertical bar.

By default, FoxDoc uses the IBM[®] extended ASCII characters in the action diagrams it prepares. Some printers do not support the IBM graphics character set. If that's your case, you can have FoxDoc prepare the action diagrams using only ASCII symbols by selecting the ASCII symbol option.

For information about how you can create your own “graphics” chart for action diagrams, refer to Action Diagram Symbols later in this chapter.

FoxDoc Xref (Cross-Reference) Options Screen

Choose Xref from the Main Menu to bring you to the Xref (Cross-Reference) Options Screen. If you elect to create the variable cross-reference report by responding Yes to the question on the Report options screen, you may want to set some of the options in the FoxDoc Xref (Cross-Reference) Options Menu screen. This screen allows you to select those items that you'd like to include in the cross-reference report.

| | | |
|--|--|----------|
| 04/18/91 | FoxDoc Xref (Cross-Reference) Options Screen | 17:09:39 |
| <p> Cross reference public variables? <input checked="" type="checkbox"/> Cross-reference other variables and tokens? <input checked="" type="checkbox"/> Cross-reference FoxPro key words? <input checked="" type="checkbox"/> Cross-reference numeric constants? <input checked="" type="checkbox"/> Local cross-references only? <input checked="" type="checkbox"/> </p> | | |

FoxDoc Xref (Cross-Reference) Options Screen

Defaults for each option are shown in the picture.

Cross-Referencing Public Variables

FoxDoc allows you to restrict cross-referencing to PUBLIC variables only. If you select this option, FoxDoc only puts PUBLIC variables on the report. However, FoxDoc does not look for PRIVATE statements that redefine the variable, so if a variable is declared PUBLIC anywhere in the system, FoxDoc documents all subsequent occurrences of it. FoxDoc does not backtrack to see if the variable was referenced before it was defined as PUBLIC.

Cross-Referencing Other Variables and Tokens

The cross-reference report normally includes all variables and tokens that appear in your source code provided that they are not comments, FoxPro keywords, numeric constants, punctuation and quoted strings.

Accordingly, file names, field names and so on will appear in the cross-reference listing. You can, however, prevent these other variables and tokens from being referenced by responding No to this option.

Cross-Referencing FoxPro Keywords

This option determines whether FoxPro keywords will be included in the cross-reference report. Normally, all words in the keywords file will not be referenced on the report. You might, however, want to select this option to see where you used certain keywords, like the REPLACE command.

Large FoxPro systems will probably exceed the number of allowable references to such commonly-used keywords as SAY, STORE, TO and others. If this happens, FoxDoc displays a warning message and continues processing your files discarding any excess references, which will not appear on the report.

Cross-Referencing Numeric Constants

This option determines whether numbers used in your system will appear on the cross-reference report. Under normal circumstances, these constants will only get in the way. There occasionally may be times, however, when you'll want to see where a particular constant has been used, especially if you need to change it later.

Local Cross-Referencing

This option determines whether FoxDoc will prepare a local cross-reference report or the standard global system-wide cross-reference report. The local report shows cross reference information for a single module at a time. In other words, the report will show where the variable "APPLE" was used in a particular program but will not show each occurrence of "APPLE" throughout the system.



Running global system-wide cross-reference report can use a lot of memory. FoxDoc can document larger programs and projects if the local cross-reference report option is chosen.

Local and global cross-reference reports are mutually exclusive. You cannot choose to prepare both reports.

FoxDoc Headings Options Screen

Choose Headings from the Main Menu to bring you to the Heading Options Screen. This screen lets you determine what is put into the program and procedure headings.

| | | |
|----------|--------------------------------|----------|
| 04/18/91 | FoxDoc Headings Options Screen | 17:10:42 |
|----------|--------------------------------|----------|

| | |
|-----------------------------------|-------------------------------------|
| Include copyright data? | <input checked="" type="checkbox"/> |
| Include procedures and functions? | <input checked="" type="checkbox"/> |
| Include called programs? | <input checked="" type="checkbox"/> |
| Include calling programs? | <input checked="" type="checkbox"/> |
| Include databases? | <input checked="" type="checkbox"/> |
| Include indexes? | <input checked="" type="checkbox"/> |
| Include format files? | <input checked="" type="checkbox"/> |
| Include report forms? | <input checked="" type="checkbox"/> |
| Include label forms? | <input checked="" type="checkbox"/> |
| Include memory files? | <input checked="" type="checkbox"/> |
| Include binary files? | <input checked="" type="checkbox"/> |
| Include other files? | <input checked="" type="checkbox"/> |
| Write procedure headings? | <input checked="" type="checkbox"/> |
| Heading insert file name? | |

FoxDoc Headings Options Screen

Defaults for each option are shown in the picture.

Copyright Data

Copyright information includes the application name, author, copyright holder and copyright date.

Procedures and Functions

FoxDoc can list the procedures and functions in your application when it writes the headings to each file.

Called Programs

FoxDoc can list the programs, procedures and function that *are called by* each program in your application in the program headings.

Calling Programs

FoxDoc can list the programs, procedures and functions that *call* each program in your application in the program heading.

Including Databases

FoxDoc can list the databases used in your application in the program heading. For information about how FoxDoc determines which databases are used in your application, refer to FoxDoc File Types Identification later in this chapter.

Including Indexes

FoxDoc can list the indexes, structural indexes and index tags used in your application in the program heading. For more information about how FoxDoc determines which indexes are used in your application, refer to FoxDoc File Types Identification later in this chapter.

Including Format Files

FoxDoc can list the format files used in your application in the program heading. For information about how FoxDoc determines which format files are used in your application, refer to FoxDoc File Types Identification later in this chapter.

Including Report Forms

FoxDoc can list the report forms used in your application in the program heading. For information about how FoxDoc determines which report forms are used in your application, refer to FoxDoc File Types Identification later in this chapter.

Including Label Forms

FoxDoc can list the label forms used in your application in the program heading.

Including Memory Files

FoxDoc can list the memory files used in your application in the program heading.

Including Binary Files

FoxDoc can list the binary files used in your application in the program heading.

Including Other Files

FoxDoc can list the other files, for example TXT files, used in your application in the program heading.

Writing Procedure Headings

FoxDoc can include separate headings for each procedure and function in the file. These headings generally include called and calling programs but you can change its contents on the Heading Options screen.

Heading Insert File Name

Enter the file name for FoxDoc to insert into your program file headings. If this field is blank, FoxDoc will not insert text from any file but the program heading will still have the regular file heading. This option is designed to allow you to create some customized heading information, for example the version number, that you would like to include in each program file heading.

This file will only be included in program and procedure file headings, and not in procedure and function headings. The text will be inserted toward the bottom of the heading, after all the other file names and references.

FoxDoc Tree Diagram Screen

Choose **Tree** from the **Main Menu** to bring you to the Tree Diagram Screen. This screen contains the options that affect the tree diagram such as procedures, functions, format files, databases, and so forth.

| | | |
|---|----------------------------|----------|
| 04/18/91 | FoxDoc Tree Diagram Screen | 17:24:03 |
| <div> <div> Create tree diagram? Include procedures? Include functions? Include format files? Include databases? Include indexes? Include report forms? Include label forms? Include memory files? Characters for tree (G/A/N)? </div> <div> Y Y Y Y Y Y N N N N G </div> <div> Filename TREE.DOC <Graphics, ASCII, None> </div> </div> | | |

FoxDoc Tree Diagram Screen

Defaults for each option are shown in the picture.

Creating a Tree Diagram

By default, FoxDoc creates a program tree for your system that contains all procedures, functions, format files and databases. In addition, you can specify that index files, report forms, label forms and memory files also be included.

This report is a diagram of the system tree structure, showing which programs call which other programs, and which programs use which databases. The diagram also indicates which of the programs are FoxPro procedures.

Including Procedures

Procedures are included in the tree diagram by default. However, you can exclude this file type from the tree diagram by setting the option to No. While you ordinarily would not want to do this, you might want to exclude the file type if there are a lot of procedures that would otherwise clutter up the diagram. If you exclude this file type you cannot use graphics or ASCII characters to depict the tree hierarchy.

Including Functions

Functions are included in the tree diagram by default. However, you can exclude this file type from the tree diagram by setting the option to No. While you ordinarily would not want to do this, you might want to exclude the file type if there are a lot of functions that would otherwise clutter up the diagram. If you exclude this file type you cannot use graphics or ASCII characters to depict the tree hierarchy.

Including Format Files

Format files are included in the tree diagram by default. However, you can exclude this file type from the tree diagram by setting the option to No. While you ordinarily would not want to do this, you might want to exclude the file type if there are a lot of format files that would otherwise clutter up the diagram. If you exclude this file type you cannot use graphics or ASCII characters to depict the tree hierarchy.

Including Databases

Databases are included in the tree diagram by default. However, you can exclude this file type from the tree diagram by setting the option to No. While you ordinarily would not want to do this, you might want to exclude the file type if there are a lot of databases that would otherwise clutter up the diagram. If you exclude this file type you cannot use graphics or ASCII characters to depict the tree hierarchy.

If you include databases, they are displayed in the tree diagram beneath the programs that USE them. Alias names are also shown.

Including Indexes

FoxDoc does not include indexes in the tree diagram by default. This file type may be included in the tree diagram by setting the option to Yes.

Including Report Forms

FoxDoc does not include report forms in the tree diagram by default. This file type may be included in the tree diagram by setting the option to Yes.

Including Label Forms

FoxDoc does not include label forms in the tree diagram by default. This file type may be included in the tree diagram by setting the option to Yes.

Including Memory Files

FoxDoc does not include memory files in the tree diagram by default. This file type may be included in the tree diagram by setting the option to Yes.

Characters For Tree

The action diagram tree is created using IBM box-graphics characters (G) by default. Because some printers cannot print these graphics characters, two other options are available:

- The ASCII option (A) uses ASCII characters only for creating the tree so that no graphics characters are used. With this option, the plus sign is used for joining lines, while hyphens and pipe symbols are used for horizontal and vertical lines.
- The None option (N) causes no characters to appear at all.

FoxDoc Printing Options Screen

Choose **Print** from the **Main Menu** to bring you to the Printing Options Screen. This screen contains the options used for printing source code and action diagram files.

FoxDoc prints a three-line heading on each page of the printout, showing program name, system name, copyright holder, date, time and page number. The heading begins on the line immediately following the top margin you specify. If you use a top margin of 8 and a bottom margin of 8 on 66-line paper, only 47 lines of code will be printed on each page. These 66 lines also include 8 lines for a top margin, 8 lines for a bottom margin, and 3 lines for the heading. The programs are printed in alphabetical order.

If the sum of your left and right margins exceeds your line width, FoxDoc will set the line width and the margins to their default values. Similarly, FoxDoc will reject top and bottom margins that are greater than the page length, any negative values, tab expansions greater than 12 spaces and so on.

New Function Key Options

There are two new function key options associated with this screen: **F2-List** and **F9-Print now**. Pressing F2 or choosing **F2-List** with your mouse will cause a list of printer types to appear. Pressing F9 or choosing **F9-Print now** will allow you to print files without going through the full documentation process. For more information, see Printing Without Documenting below.

Printing Without Documenting

You can use the source code printing facilities of FoxDoc without going through the full documentation process. If you press the F9 key while viewing the Print screen, FoxDoc will prompt you for a file name containing the names of files to print. FoxDoc searches the file you name and prints the contents of the files listed therein. FoxDoc will not print a file with an extension of .MEM, .FRM, .DBF, .CDX, .IDX, .SCX, .FRX, .MNX, .LBX, .FPT, .EXE, .COM and other common non-ASCII names. Also, FoxDoc tries to figure out if a particular file is not an ASCII file and warn you about it before printing.

FoxDoc will not try to print anything that isn't a file name, so the file you specify can contain all sorts of garbage without causing a problem. The main effect of all this is that you can give the Print routines the name of a FoxDoc FILELIST.DOC file and it will print all the source code in the system without going through the full

FoxDoc documentation process again. Of course, you can also create your own file containing file names to print as well as use this feature to print documentation files in a nicely-formatted way.

| | | | | | |
|----------------------------|-------------------------------------|--------------------------------|-------------------------------------|----------|--|
| 04/18/91 | | FoxDoc Printing Options Screen | | 17:24:16 | |
| Print source code files: | <input checked="" type="checkbox"/> | Print action diagrams: | <input checked="" type="checkbox"/> | | |
| Line width: | 132 | Page length: | 66 | | |
| Top margin: | 8 | Bottom margin: | 8 | | |
| Left margin: | 12 | Right margin: | 1 | | |
| Tab expansion: | 3 | Print line numbers: | <input checked="" type="checkbox"/> | | |
| Form feed before printing: | <input checked="" type="checkbox"/> | Form feed after printing: | <input checked="" type="checkbox"/> | | |
| Printer setup string: | | | | | |
| Printer reset string: | | | | | |

FoxDoc Printing Options Screen

Defaults for each option are shown in the picture.

Print Source Code Files

Once the new source code files are created, you can have FoxDoc automatically print them. FoxDoc expects to find these files in the output directory you specified before you began documenting.

If a line of code exceeds the line width minus left and right margins, FoxDoc will wrap it on the printout without messing up the page breaks. FoxDoc also appropriately counts the wrapped line as one source code line, so that your cross-reference report still matches the printout.

Print Action Diagrams

Once the action diagrams have been created, you can have FoxDoc automatically print them. FoxDoc expects to find this file in the output directory you specified before you began documenting.

If you are printing from a file, the ACT files must be listed in the file for FoxDoc to find them. Note that FILELIST.DOC does not list these files.

Line numbers are never added to an action diagram at print time. If you would like action diagrams to be printed with line numbers, choose the option to add the line numbers directly to the action diagram file in the Format screen.

Line Width

The line width should be set between 40 and 255, and should be the total number of characters your printer can print across one line. The standard line width of most printers is 80 with 8.5 x 11 inch paper, although you can use wider paper and set the options accordingly. The actual length of the printed source code lines will be reduced by the left and right margins that you specify. Source code lines longer than the line width will be wrapped to the next line.

Page Length

The page length should be greater than or equal to zero. If you enter zero, the program assumes continuous form paper and writes page headings on only the first page. Page length must also be larger than the top and bottom margins.

Top Margin

The top margin must be greater than three and smaller than the page length. The first three lines of each page contain a heading.

Bottom Margin

The bottom margin must be greater than or equal to zero. The bottom margin is the number of lines at the bottom of each page that will be left blank.

Left Margin

The left margin must be greater than zero and less than the line width. Line numbers, or source code if you elect not to print line numbers, will begin printing this many spaces from the left edge of the paper. A left margin of 12 usually leaves room for three-ring binder holes.

Right Margin

The right margin must be greater than zero and less than the line width. This margin is the number of spaces that remain between the source code and the right edge of the paper.

Tab Expansion

The tab expansion number must be between 1 and 12. Each tab character in the source code will be converted to this many spaces as the source code is printed. The source code file will be unaffected as the expansion is performed for printing purposes only. Tab expansions of 3 or 4 make the code easy to follow without causing it to extend beyond the right side of the page.

Print Line Numbers

FoxDoc will print line numbers next to each line of source code by default. Line numbers are useful references for following up on error messages or for using the variable cross-reference report.

This option does not add line numbers to action diagrams. If you want action diagrams with line numbers on them, you must specify this option before the diagrams are created.

Form Feed Before Printing

FoxDoc does not send a form feed before printing the first source code file by default. To change this, enter Yes.

Form Feed After Printing

FoxDoc sends a form feed after printing the last source code file by default. To change this option, enter No.

Printer Setup String

This option allows you to specify setup codes to be sent to your printer before printing begins. The default codes shift an Epson[®] and many other printers into compressed print mode. The format for setup codes is three-digit decimal numbers separated by backslashes (\). For example, the default code is \015. This field is optional

Printer Reset String

This option allows you to specify reset codes to be sent to your printer after FoxDoc finishes printing. The default codes shift an Epson and many other printers into regular print mode. The format for reset codes is three-digit decimal numbers separated by backslashes (\). For example, the default code is \018. This field is optional.

FoxDoc Other Options Screen

Choose **Other** on the **Main Menu** to display the Other Options Screen. This screen contains miscellaneous options.

04/18/91

FoxDoc Other Options Screen

17:24:29

```

      Ignore drive designations?  Y
      Search tree (Y/N)?         Y
Associate DBFs with other files (Y/N)? Y
      Default extension for index files?  IDX
      Default extension for report forms? FRX
      Default extension for label forms?  LEX

```

FoxDoc Other Options Screen

Ignore Drive Designations

Sometimes you may want FoxDoc to disregard explicit drive and path designations when searching for programs or other files. This option instructs FoxDoc to drop any drive or path designations before attempting to find a file.

This option is useful if your program tests for the existence of files on various drives and makes a decision based on the results. If you choose not to ignore drive designations, FoxDoc will attempt to find the program files on the specified drive. Depending on the equipment you have installed, MS-DOS may prompt you to insert a disk in another drive.

For example, you may have written a backup routine to copy a database to B:BACKUP.DBF. If you would like FoxDoc not to try to find that file on the B: drive, choose the option to ignore drive designations.

Search Tree

FoxDoc assumes that you want to document not only the main program file, but all of the programs it calls, all of the programs called by programs that main program calls, and so on. FoxDoc searches the program tree for all programs, databases, index files, report forms, format files, label forms and memory files as it prepares application documentation. You never need to specify more than the main program file name.

If you choose not to search the tree, only the specific file you enter will be documented. Thus, you can limit documentation to a particular file or a branch of the program tree by varying either the file you input as main program file or the search tree parameter.



FoxDoc does not track SET DEFAULT TO statements. Files that are named without drive designations are assumed to be on the default drive or in the source file subdirectory.

Associate DBFs With Other Files

The default setting *attempts* to figure out how your databases, indexes, report forms, etc., are related.

Default Extension for Index Files

Enter the default extension for index files. For example, you might enter IDX for a FoxBASE+ application. FoxDoc assumes that index files have this extension unless you specifically use another extension as part of the file name.



FoxDoc searches for all .CDX indexes automatically, regardless of this setting.

Default Extension for Report Forms

Enter the default extension for index files. For example, you might enter FRM for a FoxBASE+ application or FRX for FoxPro. FoxDoc assumes that report forms have this extension unless you specifically use another extension as part of the file name.

Default Extension for Label Forms

Enter the default extension for index files. For example, you might enter LBL for a FoxBASE+ application. FoxDoc assumes that label forms have this extension unless you specifically use another extension as part of the file name.

FoxDoc Commands

FoxDoc supports several commands that you can insert in your source code files. These commands will look like comments to FoxPro, but have special meaning for FoxDoc. The commands allow you to define macros, insert imaginary program statements into your system, turn FoxDoc features on and off, and so forth.

Macros

Using macro substitution in a FoxPro program is one way to increase the application's flexibility. However, macro substitution "hides" a variable's value. While FoxDoc does have a limited ability to handle macro substitution, macro-substituted variables are generally next to impossible to properly document.

For documentation purposes, you might want to replace occurrences of macro substitution with actual filenames or values. By doing this, some FoxDoc output, such as the Xref report, will be more complete and, therefore, more useful.

To define a macro, place the following line in your source code:

```
*# DOCMACRO source target1 target2 ...
```

The DOCMACRO statement begins with an asterisk and a pound sign. (See Program Limitations and Miscellaneous Notes.)

The source string does not include the ampersand (&) symbol for the macro, therefore use `source`, not `&source`. The rest of the line, minus leading and trailing blanks, will be used as a substitution string.

The source must be a single word, delimited by spaces or tabs. The target can be a single word or several words. Everything on the line, both source and target, will be converted to upper-case.



You cannot use `&&` comments on a DOCMACRO line since FoxDoc will think they are part of the target. If you want to document the purpose of the DOCMACRO statement, put a comment on the line above or below it.

You can also define macro substitutions in a separate file. Specify the `/mfilename` switch on the command line to tell FoxDoc where the substitution strings are. For example, the following command

will start FoxDoc and read macro definitions from the file MYMACROS.MAC:

```
FOXDOC /mMYMACROS.MAC
```

If you do not specify a filename with the /M switch, FoxDoc will look for MACRO.FXD.

You cannot abbreviate FoxDoc keyword DOCMACRO. You must use the full command verb on each line of the file. Macro substitutions defined in source code files take precedence over those defined in a separate macro file.

The /O switch on the FoxDoc command line disables *all* macro processing, both from DOCMACRO statements in source code and in a named macro file.

The DOCMACRO statement can appear anywhere, as long as FoxDoc sees it before encountering the macro that you want to replace. If the target will have the same value throughout the system, for example, DOG will always be replaced by CAT, it is a good idea to put all of the macros in one place, perhaps at the top of the main program file or in a separate macro file.

If you would like FoxDoc to use different macro values during documentation, you may want to place the DOCMACRO statement immediately before the specific location of each macro substitution. If you have multiple definitions for a single macro, only the last one FoxDoc sees will be effective. For example, if the following series of statements are in your program,

```
*# DOCMACRO apple target1
*# DOCMACRO apple target2
*# DOCMACRO apple target3
```

only target3 will be effective and will be substituted for each occurrence of &apple. (See DOCCODE: Pseudo Program Statements below for an alternative approach.)

FoxDoc will try to remove macros that look like drive or path designations. For example, if the following statement is in one of your programs:

```
USE &MPATH.DATABASE
```

FoxDoc will detect the macro and remove it before trying to find DATABASE.DBF. One consequence of this limit is that you cannot use macros to refer to program files in different directories. All source code program files, that is programs, format files, memory

files, and so forth, must reside in the single program directory you specify in the FoxDoc System Menu. Databases and indexes can be in either the program or data directories you specify.

FoxDoc will not substitute the macro in the code itself, but will use the substitution value everywhere else. For example, assume your program uses a database whose name is contained in a macro variable named “dataname.” Assume further that you have included the following line in your code:

```
*# DOCMACRO dataname ABC.DBF
```

If your code has this line in it:

```
USE &dataname
```

The source code will be unchanged after running FoxDoc, but the program headings and all other FoxDoc reports will show this program using the ABC.DBF database.

DOCCODE: Pseudo Program Statements

Sometimes you may want to enter pseudo program source code statements — statements that FoxDoc treats as real but which do not interfere with your FoxPro program when it executes. A good example is a macro that calls one of a number of programs, depending on the value it has at run time. For example,

```
DO &program
```

The macro substitution discussed in the previous section provides a way for you to specify one value for a program. But suppose you want FoxDoc to assign a program several values and to document each one in turn. The DOCCODE directive provides a way.

The DOCCODE directive causes FoxDoc to treat any text on the line as if it were a program source code statement. Using the example above, you could enter the following lines where the macro was called:

```
DO &program
*# DOCCODE DO proc1
*# DOCCODE DO proc2
*# DOCCODE DO proc3
```

FoxDoc acts as if these were perfectly straightforward calls to PROC1, PROC2 and PROC3 and documents them accordingly, while FoxPro ignores the statements. However, FoxDoc does not know that these DOCCODE statements have anything to do with the DO

statement above. FoxDoc does not associate PROC1 with the program. The next time FoxDoc sees `DO &program` it will not assume that `program` has the value PROC1. If you need FoxDoc to give values to macros, use DOCMACRO.

As a side benefit, DOCCODE statements help document your code by specifying the values that particular macros can take.

Other FoxDoc Directives

Other FoxDoc directives, or commands, allow you to turn cross-referencing and program formatting on or off. For example, you may have some thoroughly debugged and formatted boilerplate code which you insert into many other systems. It would be time-consuming to reformat this code every time you run FoxDoc.

FoxDoc provides commands that temporarily suspend cross-referencing and formatting: XREF, FORMAT, INDENT, CAPITAL and EXPAND. Each of these commands accepts three settings: ON, OFF and SUSPEND. The ON and OFF commands are effective until FoxDoc sees the other directive, while SUSPEND is in effect only until the end of the current program or procedure.

For example, if you insert this line in your code:

```
*# FOXDOC XREF OFF
```

no tokens will be cross-referenced until this line is encountered:

```
*# FOXDOC XREF ON
```

On the other hand, the following line

```
*# FOXDOC XREF SUSPEND
```

suspends cross-referencing only for the current file. Cross-referencing restarts when the next program or procedure is documented.

FoxDoc FORMAT works in the same way. If format is off, no indenting or capitalization will be done. A FoxDoc FORMAT statement affects indenting, capitalization and keyword expansion or contraction. The three commands INDENT, CAPITAL and EXPAND turn specific features on or off.

FORMAT is a shorthand way of referring to CAPITAL, INDENT and EXPAND all at once. Thus, the following sequence suspends indenting and capitalization, but enables keyword expansion:

```
*# FOXDOC FORMAT SUSPEND
*# FOXDOC EXPAND ON
```

The `*#` sequence can be changed with the `/T` switch. For example, if you invoke FoxDoc with the switch

```
/T*$
```

FoxDoc will use `*$` to designate DOCMACRO, DOCCODE and other FoxDoc directives. *You must start this sequence with an asterisk so that FoxPro knows it is a comment*, and it cannot be more than three characters long.

Using FoxDoc in a Batch Environment

If you have several systems to document, you may want to set up a batch file to invoke FoxDoc with the appropriate parameters for each system. FoxDoc supports Immediate Mode for batch documentation through the /X switch on the command line. Used in conjunction with named configuration and macro files, you can sequentially document a series of FoxPro systems. FoxDoc will take its input from the configuration files or from CONFIG.FXD if no configuration file is named, and begin documenting the system immediately.

To create a configuration file, modify the values of the necessary FoxDoc parameters then press F5. FoxDoc will ask for the filename to use for this specific configuration file and will then write it to disk.

You have to have the configuration file ready before you invoke the system. If you do not specify a configuration file, FoxDoc tries to use CONFIG.FXD in the current directory.

To document three FoxPro systems without pausing for user input, set up a batch file with these lines:

```
FOXDOC /x /fssystem1.fxd  
FOXDOC /x /fssystem2.fxd  
FOXDOC /x
```

FoxDoc will document the first system and take input from the SYSTEM1.FXD configuration file. Then, without prompting for further user action, it will read SYSTEM2.FXD and begin documenting the second system. Input for the third system is in the CONFIG.FXD, the default file name. When the batch file is invoked, FoxDoc will begin documenting and will not pause for user input at any point in the process.

If an error occurs anywhere during the processing of these three systems, FoxDoc will display an error message and stop documenting the system in which the error occurred. The batch file, however, will continue running and will move on to the next system to be documented.

Program Limitations and Miscellaneous Notes

For information about the maximum limitations in FoxDoc, press F1 three times at the sign-on screen.

Memory Usage

FoxDoc uses about 225K of memory as “overhead.” This much is used as soon as the program is invoked. It also allocates additional memory dynamically as it is needed. It will probably not be of much use if you don’t have at least 512K.

Continuation Lines

Continuation lines may be a problem in some situations. FoxDoc actually reads each source file twice, once to check for files and once to handle formatting such as indentation, capitalization, and so forth, and a second time for cross-referencing. FoxDoc tries to deal with continuation lines during the first scan and will properly parse statements such as:

```
SET INDEX TO apple, orange, pear, ;  
          grape, peach
```

and

```
SUM hatsize ;  
  FOR level = "MANAGEMENT"
```

However, FoxDoc cannot tell that

```
abracadabra = ;  
  opensesame
```

is an assignment statement. This variation of the assignment statement is unique in that it does not begin with a keyword. The alternate form of the assignment statement

```
STORE abracadabra TO opensesame
```

would be properly parsed. Accordingly, the cross-reference report will not contain the correct flag for an assignment statement. While “abracadabra” and “opensesame” will properly appear on the report, the proper cross-reference marker will not be present.

Multiple Procedure Files

If you use multiple procedure files and if they contain procedures that have the same name, FoxDoc may document your system incorrectly. For example, assume you have two procedure files that are SET at different points in the system. Both files contain a procedure called PROC1, but PROC1 is not the same in each file.

When FoxDoc searches for procedures, it will always find the first PROC1 and assume it is the one being called. FoxDoc does not track which procedure file is currently active and cannot distinguish between identically-named procedures in different procedure files. This limitation has no effect on applications that use only one procedure file, or on applications that have multiple procedure files containing uniquely-named procedures.

Command Line Switches

The following table shows all of the command line switches available for use with FoxDoc. This information can also be found by pressing F1 twice at the FoxDoc startup screen. These switches can be used in the Command window when invoking FoxDoc or put in the CONFIG.FP file.

FoxDoc allows command line switches to begin with either a hyphen (-) or a forward slash (/).

| Switch | Action |
|-----------|--|
| A | Insert an extra indent before a CASE statement. |
| BW | Black and white. Disable color. |
| Ffilename | Use configuration data in "filename." |
| Lfilename | Use standard link data in "filename". |
| Mfilename | Use macro substitution strings in "filename." |
| net | Disable standard printing to print through the network |
| O | Omit all macro substitutions. |
| Pport | Specify the print destination, for example /PLPT2, /Pfilename, /PLPT2(net). Note that the last example is the same as the net command line argument above. |
| Rnnnn | Reserve <i>nnnn</i> bytes of memory. |
| S | Show reports on screen as they are created. |
| T*: | Use '*' as Tag for FoxDoc directives. The '*' can be replaced by characters of your choice. |
| U | Turn off the mouse. |
| Wnnn | Hyphens for reports. |
| X | Run FoxDoc without waiting for input. |
| Y | Do not use EMS. |

Changing, Saving and Restoring Default Options

When you start FoxDoc, the information stored in CONFIG.FXD is used to establish the default field values. FoxDoc looks for this file in the current directory. If it cannot find it, FoxDoc will use the built-in default values.

You can specify another configuration file by using the /F switch on the command line. *Do not include a space between the "/F" switch and the file name.* For example, the command

```
FOXDOC /FC:\MYDIR\MYCONFIG.BAZ
```

tells FoxDoc to look for a configuration file named MYCONFIG.BAZ in the \MYDIR subdirectory on drive C.

As you use FoxDoc, you may wish to modify the default values for certain fields. For example, you may want to set the path for program source code to C:\FOXPRO25\PRG, or you may always want the author and copyright holder fields to display your name. You can make these modifications by entering the data in the appropriate fields and saving the information as the new default.



Save a separate configuration file for each system you use in the same subdirectory with the system. This helps when running FoxDoc in batch mode and for keeping subdirectories, file names, etc., straight.

Default File Names for Report Output

Below is a table listing all of the default file names used by FoxDoc in the creation of reports. Any of these file names can be changed.

| File Name | Description |
|------------------|--|
| TREE.DOC | Tree structure diagram. |
| FILELIST.DOC | List of files that make up the system. |
| NDXSUMRY.DOC | Summary of index files used. |
| DATADICT.DOC | Summary of databases and fields used. |
| FMTSUMRY.DOC | Summary of all format files used. |
| FRMSUMRY.DOC | Summary of all report forms used. |
| PRCSUMRY.DOC | Summary of all procedures and functions. |
| LBLSUMRY.DOC | Summary of all label forms used. |
| SCRNSMRY.DOC | Summary of all screen files used. |
| MENUSMRY.DOC | Summary of all menu files used. |
| BINSUMRY.DOC | Summary of any binary file used. |
| MEMSUMRY.DOC | Summary of all memory files used. |
| STATS.DOC | Summary of the system statistics. |
| XREF.DOC | The cross-reference report. |

FoxDoc File Types Identification

FoxDoc uses the following form identification when it creates Database, Index, Format File and Report Form Summaries.

Index Identification

FoxDoc identifies indexes in the following forms:

- INDEX ON xyz TO index
- USE abc INDEX index1,index2,index3 ...
- SET INDEX TO index1,index2,index3 ...
- DELETE FILE xyz.IDX or DELETE FILE xyz.CDX

In the second and third cases, each index file will be separately identified and documented. A statement that tests for the existence of an index, for example

```
IF FILE("xyz.IDX")
```

will not by itself be identified as an index reference. FoxDoc also lists the database associated with each index file if it can be determined. For more information on the assumptions that FoxDoc makes about which database is active, see the Database Identification section that follows.

FoxDoc tries to determine which indexes, report forms and label forms are associated with each database by tracking the following statements:

```
USE dbfname
SELECT A (or B, C, etc.)
SELECT aliasname
CLOSE DATABASES
CLOSE ALL
USE
```

FoxDoc generally treats IDX and CDX files similarly and can reference to CDX files such as INDEX ON xyz TAG index.

Database Identification

FoxDoc identifies databases in the following forms:

- **USE statements**
Only those USE statements followed by a database name.
- **COPY TO statements**
Including COPY TO ... SDF. If the command copies to an SDF file without an explicit extension, FoxDoc supplies .TXT; otherwise, FoxDoc assumes databases have .DBF extensions.
- **DELETE FILE xxx.DBF**
- **CREATE datafile2 FROM datafile1**
FoxDoc picks up both datafile2 and datafile1.
- **SORT ON [key] TO xxx.DBF**
- **CREATE xxx.DBF FROM xyz.DBF**

A statement that tests for the existence of a database, for example

```
IF FILE("xyz.dbf")
```

will not by itself be identified as a database reference. Currently, FoxDoc imposes an overall limit of 1,024 fields for all databases used in the application. If you exceed this number, FoxDoc returns an error message and does not include the excess fields in the database report summary.

FoxDoc knows how to account for macros in each of these constructions, assuming you have defined the macro. For example,

```
USE &temp
```

will be interpreted to mean

```
USE abcfile
```

if you have previously defined &temp to mean abcfile. (See FoxDoc Commands for more information.)

FoxDoc is reasonably accurate when determining which database is active at a particular point in the code. However, there are some limitations:

- FoxDoc does not track databases or work areas across program files or procedures.

- When FoxDoc begins documenting a file, it assumes that no databases are in use and that work area A is active.
- FoxDoc does not try to interpret conditional structures, for example:

```
IF .T.
    USE apple
ELSE
    USE pear
ENDIF
```

FoxDoc looks at this code one line at a time. The last USE statement it sees is:

```
USE pear
```

That statement will never actually be executed because of the IF test, but FoxDoc doesn't know that. You will need to use one or more DOCCODE statements to show FoxDoc what's happening in cases such as this one. (See the section titled FoxDoc Commands for more information.)

Format File Identification

FoxDoc identifies format files in the following forms:

- SET FORMAT TO xyz
- DELETE FILE xyz.FMT

A command that tests for the existence of a format file, for example:

```
IF FILE("xyz.FMT")
```

will not by itself be identified as a format file reference.

Report Form Identification

FoxDoc identifies report forms in the following forms:

- REPORT FORM xyz ...
- DELETE FILE xyz.FRM

A command that tests for the existence of a report form, for example:

```
IF FILE("xyz.FRM")
```

will not by itself be identified as a report form reference.

Label Form Identification

FoxDoc identifies report forms in the following forms:

- LABEL FORM xyz ...
- DELETE FILE xyz.LBL

A command that tests for the existence of a report form, for example:

```
IF FILE("xyz.LBL")
```

will not by itself be identified as a report form reference.

Screen File Identification

FoxDoc identifies screen files in four ways:

- If you specify a project file as the main file, FoxDoc will scan it for any screens included in the application.
- FoxDoc will infer the existence of a screen file if it sees a reference to an SPR file.
- You can tell FoxDoc to document a screen file by including a statement like this:

```
*# FOXDOC SCREEN scrnname
```

- CREATE SCREEN scrnname FROM xyz.DBF

Menu File Identification

FoxDoc identifies menu files in three ways:

- If you specify a project file as the main file, FoxDoc will scan it for any screens included in the application.
- FoxDoc will infer the existence of a screen file if it sees a reference to an MPR file.
- You can tell FoxDoc to document a screen file by including a statement like this:

```
*# FOXDOC MENU menuname
```


Cross-Reference Codes

The cross-reference report adds certain codes to the end of a line number reference when the reference has particular significance. The following table displays the statements that FoxDoc looks for to tell if a variable or field name (assume its name is "abc") is significant, as well as the code that will appear with the cross-reference listing. The table also shows the code that will appear with the cross-reference listing to indicate the reference's significance.

| Reference Type | Code |
|---------------------------------|------|
| abc = 4 | = |
| STORE 4 TO abc | = |
| WAIT to abc | = |
| @ 1,1 GET abc | G |
| ACCEPT "something" TO abc | G |
| INPUT abc | G |
| REPLACE abc WITH something | R |
| RELEASE abc | x |
| PUBLIC abc | P |
| PRIVATE abc | V |
| ? &abc | & |
| DIMENSION abc(100,200) | A |
| USE abc [INDEX ...] [ALIAS ...] | U |
| DO proc WITH @abc | @ |

As the following excerpt from a cross-reference report illustrates, you can see at a glance what's happening to the variables:

```

ABC
      TODO.PRG          30P  41    43G  44    45=  47G  72x
      TDINPUT.PRG 123V 234= 235  237
      SETFILT.PRG  16   24   28   32
      TDEDIT.PRG  107= 133  134  135  136  138x

```

This report shows that variable ABC was used in the four programs TODO, TDINPUT, SETFILT and TDEDIT. Within TODO, it was declared a PUBLIC variable on line 30, referenced but not changed on

lines 41 and 44, used in a GET statement on lines 43 and 47, modified in line 45, and released on line 72. TDINPUT declared it to be PRIVATE on line 123, assigned it a value on line 234, then referred to it on lines 235 and 237.

A legend explaining these flags is printed at the top of each cross-reference report.

FoxDoc will not flag a RELEASE ALL statement as a reference, nor will it figure out that a RESTORE FROM XYZ.MEM may overwrite other variables. Neither of these statements will generate a cross-reference listing for any variables though XYZ.MEM will be referenced. Of course, you can have FoxDoc cross-reference the RELEASE and RESTORE statements themselves.

References to a database field prefaced by the alias for example, DATAFILE.FIELD, will be shown as one token in the cross reference file.

Batch Programs

While FoxDoc is documenting your application, it can produce several MS-DOS batch files that can serve as useful utilities.

UPDATE.BAT

Copies all program files from the FoxDoc output directory to the original source directory. You use it to copy the new documented versions of your source files over the original undocumented source files. The syntax for UPDATE is:

```
UPDATE <drive:dir>
```

where <drive:dir> is a drive and/or directory name. If it is omitted the original source directory is assumed.



Be careful with this file because it *overwrites* the original source code files. You should backup your original files and review the new output files carefully before executing UPDATE

BACKPRG.BAT

Backs up the programs, format files and report forms in your system. Its syntax is:

```
BACKPRG <drive:dir>
```

<drive:dir> is a drive and/or directory name and is required. If you do not specify a target drive or directory, the batch file returns an error message and stops.

BACKDBF.BAT

Backs up the databases, index files and memory files in your system. Its syntax is:

```
BACKDBF <drive:dir>
```

<drive:dir> is a drive or directory name and is required. If you do not specify a target drive or directory, the batch file returns an error message and stops.

PRINTDOC.BAT

Sends all documentation files, not including source code or action diagrams, to the printer. PRINTDOC calls the MS-DOS PRINT utility for background printing. Thus, PRINT must be available somewhere on the path.

Key Word File List Information

The outcome of capitalization, cross-referencing and keyword expansion and compression is determined by the words in the specified keyword file. Specifically, the keyword file tells FoxDoc what is to be considered a keyword and what is not.

The keyword files are standard ASCII files that you can edit with the FoxPro text editor or most any word processor. The keyword file should contain one keyword per line. Capitalization and order do not matter, except that if two identical keywords have different flags, the last keyword takes precedence.

By inserting certain characters in the file immediately before a keyword or variable, you can cause them to be handled differently than normal. The following characters have a special meaning when inserted in the keyword file:

- * Comments out the word. FoxDoc acts as if it were not in the keyword file at all.
- ! Capitalize, but do no cross-reference even when the option to cross-reference keywords is in effect. You may want to use this character for often-used but uninteresting keywords such as TO or SAY.
- @ Capitalize and always cross-reference this word, even when the option to cross-reference keywords is not in effect. You might want to use this for important keywords such REPLACE, SAVE or QUIT.
- % Do not capitalize or cross-reference, regardless of whether the options to capitalize or cross-reference keywords are in effect. You may want to use this character for variables that you use frequently, but that you are not interested in cross-referencing. This will keep them from cluttering up the cross-reference report. Such words are not affected by key word expansion or contraction.
- () If the *last two characters* in a keyword are (), the keyword will be considered a function and will ordinarily have an initial capital letter in the formatted output. Functions are never affected by keyword expansion or contraction.

The following examples show how to use these special characters:

```
*note  
!SAY  
@REPLACE  
%CHOICE  
SOME_FUNCTION()
```

Indentation

FoxDoc recognizes FoxPro control structures and can indent program statements in your source code to make it more readable. Indentation is, by default, three spaces. If you would prefer indenting with tab characters instead of spaces, you can set this option also. You can also instruct FoxDoc to use a different number of spaces for each indentation level or you can choose not to indent source code at all.

If you use tab characters when indenting your source code, and you choose to use FoxDoc's source code printing routines, you can also select the number of spaces to be inserted for each tab as the code prints.

If you choose to indent source code or create action diagrams, FoxDoc will scan your code for mismatched control structure terminators. For example, if your code has the following sequence:

```
DO WHILE T statements
    ...
    Program statements
    ...
    IF X = Y
        ...
        Program statements
        ...
    ENDIF
ENDIF <— (incorrect)
```

FoxDoc will detect that the final statement should have been an ENDDO instead of an ENDIF and will display an appropriate error message. FoxDoc will accept END as a valid abbreviation of any control structure terminator.

Symbols

You can replace any of the default symbols with others of your choosing. In fact, you can assign your own complete set of action diagram symbols. You must enter O (for Other Symbols) to be able to change the action diagram symbols. If you enter G, then change the symbols, the changes will not take effect. Any user-defined symbols will be saved in the configuration file.

The table on the following page describes the symbols in order from left to right, explaining where each one is used in the action diagram.

To restore the default set of symbols, select G for graphics characters, move to another input screen, then come back again. The symbols are reset when the Format screen is presented.

| Position | Description |
|----------|--|
| 1 | Horizontal symbol for IF/ELSE/ENDIF |
| 2 | Vertical symbol for IF/ELSE/ENDIF |
| 3 | Corner for IF |
| 4 | Corner for ENDIF |
| 5 | Join symbol for ELSE |
| 6 | Horizontal symbol for DO WHILE/ENDDO |
| 7 | Vertical symbol for DO WHILE/ENDDO |
| 8 | Corner for DO WHILE |
| 9 | Corner for ENDDO |
| 10 | Symbol for LOOP |
| 11 | Symbol for EXIT |
| 12 | Horizontal symbol for DO CASE/OTHERWISE/ENDCASE |
| 13 | Vertical symbol for DO CASE/OTHERWISE/ENDCASE |
| 14 | Corner for DO CASE |
| 15 | Corner for ENDCASE |
| 16 | Join symbol for CASE and OTHERWISE |
| 17 | Horizontal symbol for FOR/NEXT |
| 18 | Vertical symbol for FOR/NEXT |
| 19 | Corner for FOR |
| 20 | Corner for NEXT |
| 21 | Arrow symbol for RETURN/CANCEL/QUIT |
| 22 | Horizontal symbol for RETURN/CANCEL/QUIT/LOOP/EXIT |

Sample Reports

The following sample reports illustrate the formats of some FoxDoc output. Because the documentation for a full system is voluminous, only limited portions of the documentation are presented here. This saves much space, but has the disadvantage of making some reports “out of sync.” Thus, the internal consistency of a complete package of documentation is missing here. For example, if you try to trace the entries in the tree back to TODO.PRG, you’ll discover they don’t track since much of TODO.PRG has been trimmed out to reduce the bulk of the documentation. Similarly, the system summary shows a lot of databases that aren’t in the DATADICT.DOC file.

Of course, the best way to see sample FoxDoc reports is to run FoxDoc on some of your own programs.

Sample Main Program/Project File

```

*:*****
*:
*:      Program: TODO.PRG
*:
*:      System: ToDo - To Do Management System
*:      Author: Walter J. Kennamer
*:      Copyright (c) 1988, Walter J. Kennamer
*:      Last modified: 02/26/88      23:00
*:
*:      Calls: HELP.PRG
*:              : F2_HANDLER      (procedure in TODOPRC.PRG)
*:              : HELPEEDIT      (procedure in TODOPRC.PRG)
*:              : TDDEFLT.PRG
*:              : ERRORMSG      (procedure in TODOPRC.PRG)
*:              : SETDATE.PRG
*:              : TDLOGO.PRG
*:              : TDSETUP.PRG
*:              : TDINPUT.PRG
*:
*:      Memory Files: ACESSES.MEM
*:              : DEFAULT.MEM
*:
*:      Documented: 03/02/88 at 18:39
*:*****
PARAMETERS c_line      && command line
*# DOCMACRO s_tdfilename todo
*# DOCMACRO s_ddndx      tododd
*# DOCMACRO s0_adfname address external subject
PUBLIC fox,s_cmdline
IF pcount() < > 0
    s_cmdline = Alltrim(UPPER(c_line))
ELSE
    s_cmdline = ""
ENDIF
.
.
.
* read defaults
USE
IF FILE("default.mem")
    REST FROM DEFAULT ADDITIVE
    IF s0_bw
        s0_montype = "M"
    ELSE
        s0_montype = "C"
    ENDIF
ELSE
    DO tddeflt
ENDIF
DO tdinput
*: EOF: TODO.PRG

```

System Summary

System: ToDo - To Do Management System
Author: Walter J. Kennamer
03/02/88 18:56:29
System Summary

This system has:

- 8636 lines of code
- 46 program files 2 procedure files
- 84 procedures and functions
- 18 databases
- 8 index files
- 3 report forms
- 0 format files
- 0 label forms
- 5 memory variable files
- 622 cross-referenced tokens

See the tree diagram for programs, procedures, functions and format files

| Databases | Index Files | Report Forms | Label Forms | Memory Files |
|---------------|----------------|-----------------|----------------|-----------------|
| ----- | ----- | ----- | ----- | ----- |
| HELP.DBF | HELPKEY.IDX | SUBREPT.FRM | | ACCESSES.MEM |
| TODO.SKL | SUBJECT.IDX | TDSUMIN.FRM | | DEFAULT.MEM |
| TODO.DBF | TODODD.IDX | TDEDETIN.FRM | | LASTFILE.MEM |
| SUBJECT.DBF | &NIXNAME | | | PRINTER.MEM |
| ADDRESS.DBF | HISTDD.IDX | | | CLIP.MEM |
| &B_FILE.DBF | PRIORITY.IDX | | | |
| NOTEPAD.DBF | DATEDUE.IDX | | | |
| PRTCODES.DBF | DATEDUE.IDX | | | |
| &FILE | | | | |
| HIST.DBF | | | | |
| TEMP.DBF | | | | |
| &FNAME | | | | |
| &BAK_NAME.DBF | | | | |
| &FNAME.DBF | | | | |
| AREACODE.DBF | | | | |
| DATEDUE.DBF | | | | |
| PRM.SKL | | | | |
| ADDRESS.ASC | | | | |

FoxDoc created the following documentation files:

- C:\SCRATCH\STATS.DOC
- C:\SCRATCH\TREE.DOC
- C:\SCRATCH\FILELIST.DOC
- C:\SCRATCH\NDXSUMRY.DOC
- C:\SCRATCH\DATADICT.DOC
- C:\SCRATCH\FRMSUMRY.DOC
- C:\SCRATCH\PRCSUMRY.DOC

C:\SCRATCH\XREF.DOC
C:\SCRATCH\TODO.LNK
C:\SCRATCH\TODO.TLK
C:\SCRATCH\TODO.MLK
C:\SCRATCH\MAKEFILE
C:\SCRATCH\ERROR.DOC

Action diagram files

UPDATE.BAT to update program source files in C:\TODO
BACKDBF.BAT to backup databases, indexes and memory files
BACKPRG.BAT to backup program files, report forms and format files
PRINTDOC.BAT to print documentation files

Tree Diagram

System: ToDo - To Do Management System
Author: Walter J. Kenamer
03/02/88 18:56:09
Tree Diagram

```
TODO.PRG
HELP.PRG
    HELP.DBF (database)
    SHOW_HELP (procedure in HELP.PRG)
        SETCOLOR (procedure in TODOPRC.PRG)
        CENTER (procedure in TODOPRC.PRG)
    SETCOLOR (procedure in TODOPRC.PRG)
    CENTER (procedure in TODOPRC.PRG)
F2_HANDLER (procedure in TODOPRC.PRG)
    ADDRLIST.PRG
        SETCOLOR (procedure in TODOPRC.PRG)
    NUMLIST.PRG
        SETCOLOR (procedure in TODOPRC.PRG)
HELPEEDIT (procedure in TODOPRC.PRG)
TDDEFAULT.PRG
    SETCOLOR (procedure in TODOPRC.PRG)
    SCRHEAD (procedure in TODOPRC.PRG)
        SETCOLOR (procedure in TODOPRC.PRG)
    TDSETUP.PRG
        TODO.SKL (database)
        TODO.DBF (database)
        SUBJECT.DBF (database)
        ADDRESS.DBF (database)
    CENTER (procedure in TODOPRC.PRG)
    SETDATE.PRG
    SETPRT.PRG
        PRTCODES.DBF (database)
    SCRHEAD (procedure in TODOPRC.PRG)
        SETCOLOR (procedure in TODOPRC.PRG)
    ERRORMSG (procedure in TODOPRC.PRG)
        SETCOLOR (procedure in TODOPRC.PRG)
        CENTER (procedure in TODOPRC.PRG)
    F2_HANDLER (procedure in TODOPRC.PRG)
        ADDRLIST.PRG
            SETCOLOR (procedure in TODOPRC.PRG)
        NUMLIST.PRG
            SETCOLOR (procedure in TODOPRC.PRG)
    F3_HANDLER (procedure in TODOPRC.PRG)
```

and so forth

Procedure and Function Summary

System: ToDo — To Do Management System
 Author: Walter J. Kenamer
 03/02/88 18:55:52
 Procedure and Function Summary

 2 procedure files in the system
 SUBJECT.PRG
 TODOPRC.PRG

SUBJECT.PRG — Last updated: 12/30/87 at 8:47

Contains: SUBOK()
 Calls: SETCOLOR (procedure in TODOPRC.PRG)
 Calls: CENTER (procedure in TODOPRC.PRG)
 Calls: PUTSUB (procedure in TODOPRC.PRG)
 Contains: SUBLOOK()
 Calls: SETCOLOR (procedure in TODOPRC.PRG)
 .
 .
 .
 Contains: SUBEDIT
 Called by: ED() (function in SUBJECT.PRG)
 Calls: SETCOLOR (procedure in TODOPRC.PRG)

 TODOPRC.PRG — Last updated: 12/28/87 at 14:20

Contains: F2_HANDLER
 Called by: TODO.PRG
 Called by: TDDEFLT.PRG
 Calls: ADDRLIST.PRG
 Calls: NUMLIST.PRG
 Contains: F3_HANDLER
 Called by: TODO.PRG
 Called by: TDDEFLT.PRG
 Calls: SETCOLOR (procedure in TODOPRC.PRG)
 Contains: F4_HANDLER
 Called by: TODO.PRG
 Called by: TDDEFLT.PRG
 .
 .
 .
 Contains: ISCTRL()
 Contains: SCREEN_ON Called by: ROLODEX.PRG
 Contains: PRINT_ON
 Called by: ROLODEX.PRG

Database Structure Summary

System: ToDo - To Do Management System
Author: Walter J. Kenamer
03/02/88 18:55:42
Database Structure Summary

3 databases in the system

HELP.DBF
TODO.DBF
SUBJECT.DBF

Structure for database : HELP.DBF

Number of data records : 37

Last updated : 09/09/87 at 11:06

| Field | Field name | Type | Width | Dec | Start | End |
|-------------|------------|-----------|-------|-----|-------|-----|
| 1 | HCALLPRG | Character | 8 | | 1 | 8 |
| 2 | HINPUTVAR | Character | 12 | | 9 | 20 |
| 3 | HSCRNUM | Character | 4 | | 21 | 24 |
| 4 | HELPMSG | Memo | 10 | | 25 | 34 |
| ** Total ** | | | 35 | | | |

This database is associated with the memo file: HELP.DBT

This database appears to be associated with index file(s):
: HELPKEY.NTX (UPPER(hcallprg+hscrnum+hinputvar))

FoxDoc did not find any associated report forms

Used by: HELP.PRG
: TDFIX.PRG
: TDREINDX.PRG
: TDREPAIR (procedure in TDFIX.PRG)

Structure for database : TODO.DBF

Number of data records : 112

Last updated : 03/01/88 at 17:35

| Field | Field name | Type | Width | Dec | Start | End |
|-------------|------------|-----------|-------|-----|-------|-----|
| 1 | ITEM | Character | 55 | | 1 | 55 |
| 2 | PRIORITY | Character | 1 | | 56 | 56 |
| 3 | DATE_DUE | Date | 8 | | 57 | 64 |
| 4 | CALTIME | Character | 5 | | 65 | 69 |
| 5 | COMPLETE | Character | 1 | | 70 | 70 |
| 6 | ADVANCE | Numeric | 3 | | 71 | 73 |
| 7 | DATE_ASGN | Date | 8 | | 74 | 81 |
| 8 | DATE_COMP | Date | 8 | | 82 | 89 |
| 9 | LATE | Numeric | 3 | | 90 | 92 |
| 10 | ITEMTYPE | Character | 1 | | 93 | 93 |
| 11 | ALARM | Character | 1 | | 94 | 94 |
| 12 | SUBJECT | Character | 30 | | 95 | 124 |
| 13 | DURATION | Numeric | 8 | 2 | 125 | 132 |
| 14 | VERSION | Character | 5 | | 133 | 137 |
| ** Total ** | | | 138 | | | |

Database Structure Summary

This database appears to be associated with index file(s):
: TODODD.NTX (DTOS(date_due)+priority+caltime)
: DATEDUE.NDX (index key not found)

FoxDoc did not find any associated report forms

Used by: TDSETUP.PRG
: TDINPUT.PRG
: OPENDATA (procedure in TDSETUP.PRG)
: EDITEXIT.PRG
: TDREDATE.PRG
: TDPURGE.PRG
: TDFIX.PRG
: TDCAL.PRG
: TDREINDX.PRG

Structure for database : SUBJECT.DBF
Number of data records : 41
Last updated : 01/12/88 at 9:34

| Field | Field name | Type | Width | Dec | Start | End |
|-------------|------------|-----------|-------|-----|-------|-----|
| 1 | SUBCODE | Character | 20 | | 1 | 20 |
| 2 | SUBJECT | Character | 30 | | 21 | 50 |
| ** Total ** | | | 51 | | | |

This database appears to be associated with index file(s):
: SUBJECT.NTX (UPPER(subject))

FoxDoc did not find any associated report forms

Used by: TDSETUP.PRG
: TDINPUT.PRG : OPENDATA (procedure in TDSETUP.PRG)
: TDREINDX.PRG

Database Summary

System: ToDo - To Do Management System
Author: Walter J. Kennamer
03/02/88 18:55:49
Database Summary

Note: the actual system used more than 3 databases. All but three were removed above to save space. This portion of the report shows all of them.

| Field Name | Type | Len | Dec | Database |
|------------|------|-----|-----|--------------|
| ABBREV | C | 2 | 0 | AREACODE.DBF |
| ADDRESS | C | 53 | 0 | ADDRESS.DBF |
| ADVANCE | N | 3 | 0 | TODO.SKL |
| | | | | TODO.DBF |
| | | | | HIST.DBF |
| ALARM | C | 1 | 0 | TODO.SKL |
| | | | | TODO.DBF |
| | | | | HIST.DBF |
| AREACODE | N | 3 | 0 | AREACODE.DBF |
| BPHONE | C | 12 | 0 | ADDRESS.DBF |
| CALTIME | C | 5 | 0 | TODO.SKL |
| | | | | TODO.DBF |
| | | | | HIST.DBF |
| CITIES | C | 78 | 0 | AREACODE.DBF |
| CITY | C | 25 | 0 | ADDRESS.DBF |
| COL | N | 2 | 0 | PRTCODES.DBF |
| COMMENT | C | 50 | 0 | ADDRESS.DBF |
| COMPANY | C | 53 | 0 | ADDRESS.DBF |
| COMPLETE | C | 1 | 0 | TODO.SKL |
| | | | | TODO.DBF |
| | | | | HIST.DBF |
| COMPRESS | C | 13 | 0 | PRTCODES.DBF |
| COUNTRY | C | 20 | 0 | ADDRESS.DBF |
| DATE_ASGN | D | 8 | 0 | TODO.SKL |
| | | | | TODO.DBF |
| | | | | HIST.DBF |
| DATE_COMP | D | 8 | 0 | TODO.SKL |
| | | | | TODO.DBF |
| | | | | HIST.DBF |
| DATE_DUE | D | 8 | 0 | TODO.SKL |
| | | | | TODO.DBF |
| | | | | HIST.DBF |
| DURATION | N | 8 | 2 | TODO.SKL |
| | | | | TODO.DBF |
| ELITE | C | 13 | 0 | PRTCODES.DBF |
| FORMFEED | C | 13 | 0 | PRTCODES.DBF |
| HCALLPRG | C | 8 | 0 | HELP.DBF |
| HELPMMSG | M | 10 | 0 | HELP.DBF |
| HINPUTVAR | C | 12 | 0 | HELP.DBF |
| HSCRNUM | C | 4 | 0 | HELP.DBF |
| ITEM | C | 55 | 0 | TODO.SKL |
| | | | | TODO.DBF |
| | | | | HIST.DBF |
| ITEMTYPE | C | 1 | 0 | TODO.SKL |

| | | | | |
|------------|---|----|---|--------------|
| | | | | TODO.DBF |
| | | | | HIST.DBF |
| LATE | N | 3 | 0 | TODO.SKL |
| | | | | TODO.DBF |
| | | | | HIST.DBF |
| LINE | C | 78 | 0 | TEMP.DBF |
| | | | | PRM.SKL |
| NAME | C | 30 | 0 | ADDRESS.DBF |
| NAME | C | 25 | 0 | PRTCODES.DBF |
| NOTES | M | 10 | 0 | ADDRESS.DBF |
| | | | | NOTEPAD.DBF |
| PHONE | C | 12 | 0 | ADDRESS.DBF |
| POSITION | C | 40 | 0 | ADDRESS.DBF |
| PRIORITY | C | 1 | 0 | TODO.SKL |
| | | | | TODO.DBF |
| | | | | HIST.DBF |
| PRTNUM | N | 2 | 0 | PRTCODES.DBF |
| RESET | C | 13 | 0 | PRTCODES.DBF |
| ROW | N | 2 | 0 | PRTCODES.DBF |
| SECONDLINE | C | 53 | 0 | ADDRESS.DBF |
| STATE | C | 2 | 0 | ADDRESS.DBF |
| STATE | C | 15 | 0 | AREACODE.DBF |
| SUBCODE | C | 20 | 0 | SUBJECT.DBF |
| SUBJECT | C | 30 | 0 | TODO.SKL |
| | | | | TODO.DBF |
| | | | | SUBJECT.DBF |
| | | | | HIST.DBF |
| VERSION | C | 5 | 0 | TODO.SKL |
| | | | | TODO.DBF |
| | | | | NOTEPAD.DBF |
| ZIP | C | 10 | 0 | ADDRESS.DBF |

Index File Summary

System: ToDo - To Do Management System
Author: Walter J. Kennamer
03/02/88 18:55:15
Index File Summary

5 index files in the system
HELPKEY.NDX
SUBJECT.NDX
TODODD.NDX
&NTXNAME
DATEDUE.NDX

HELPKEY.IDX - Indexed on: UPPER(hcallprg+hscrnum+hinputvar)
Last updated: 09/09/87 at 11:06

This index file appears to be associated with database(s):
: HELP.DBF

Used by: HELP.PRG
: TDFIX.PRG
: TDREINDEX.PRG
: TDREPAIR (procedure in TDFIX.PRG)

SUBJECT.IDX - Indexed on: UPPER(subject)
Last updated: 01/12/88 at 9:16

This index file appears to be associated with database(s):
: SUBJECT.DBF

Used by: SUBLOOK() (function in SUBJECT.PRG)
: PART_MATCH() (function in SUBJECT.PRG)
: TDSETUP.PRG
: TDINPUT.PRG
: OPENDATA (procedure in TDSETUP.PRG)
: TDFIX.PRG
: TDREINDEX.PRG
: TDREPAIR (procedure in TDFIX.PRG)

TODODD.IDX - Indexed on: DTOS(date_due)+priority+caltme
Last updated: 03/01/88 at 17:35

This index file appears to be associated with database(s):
: TODO.DBF

Used by: TDSETUP.PRG
: TDINPUT.PRG
: EDITEXIT.PRG
: EDITSRCH.PRG

```
: TDREDATE.PRG
: TDPURGE.PRG
: EDITCHGE.PRG
: TDFIX.PRG
: PRTUNCMP.PRG
: TDREINDX.PRG
: TDREPAIR      (procedure in TDFIX.PRG)
```

&NTXNAME is a macro unknown to FoxDoc

This index file appears to be associated with database(s):
: ADDRESS.DBF

Used by: TDSETUP.PRG
: OPENDATA (procedure in TDSETUP.PRG)
: ADDRESS.PRG
: TDFIX.PRG
: TDREINDX.PRG
: TDREPAIR (procedure in TDFIX.PRG)
: ADDRBOOK.PRG
: ROLODEX.PRG

File not found-DATEDUE.NDX

This index file appears to be associated with database(s):
: TODO.DBF

Used by: TDCAL.PRG
: TDCALDAY.PRG

Report Form File Summary

Documenting Applications with FoxDoc

Report Form File Summary

Summary report? No
Eject page before printing? Yes
Double space report? Yes
Left margin: 1
Eject page after printing? Yes
Plain page? No
Right Margin: 0

Report Contents

| No. | Field | Length | Decimals | Totaled? |
|-----|--------------------|--------|----------|----------|
| 1 | "_____" | 6 | 0 | No |
| 2 | RECNO() | 4 | 0 | No |
| 3 | Item | 55 | 0 | No |
| 4 | " "+dtoc(date_due) | 9 | 0 | No |
| 5 | Caltime | 6 | 0 | No |
| 6 | " "+priority | 5 | 0 | No |
| | | 85 | | |
| | | ===== | | |

Report Layout

Page No. 1
00/00/00

Uncompleted Items

| No. | Item | Due Date | Time | Pri. |
|-----|------|----------|------|------|
| 1 | 2 3 | 4 | 5 | 6 |

Database and Program References

FoxDoc could not find an associated database

Used by: PRTUNCMP.PRG

Token Cross-Reference Report

System: ToDo - To Do Management System
Author: Walter J. Kennamer
03/02/88 18:55:58
Token Cross-Reference Report

622 tokens are included in this report.

Legend for context symbols: (blank) reference does not change the variable or field value.

= variable or field is changed in an assignment statement.

x variable is released.

A array is declared.

G GET statement changes variable or field.

P variable is declared PUBLIC.

R field is replaced.

U database is USED

V variable is declared PRIVATE.

& variable is referenced in a macro-takes preference over all others.

? reference is of unknown type.

ABBREV

| | | | |
|--------------|----|-----|-----|
| AREACODE.PRG | 90 | 134 | 178 |
|--------------|----|-----|-----|

ABORT

| | | | |
|--------------|-----|-----|------|
| TDINPUT.PRG | 62x | 67P | 306= |
| EDITSRCH.PRG | 35= | 69= | 99= |
| EDITCHGE.PRG | 32= | | |

ACCESSES

| | |
|------------|-----|
| TODO.PRG | 130 |
| TDEXIT.PRG | 24 |

.
. .
.

YESNO

| | | | | | | | | | |
|-------------|------|------|------|-----|------|-----|------|------|-----|
| ADDRESS.PRG | 476= | 482G | 482 | 484 | 663= | 664 | 665= | 668G | 668 |
| 672 | | 748= | 751G | 751 | 753 | | | | |

YR

| | | | | | | | | | |
|------------|------|------|-----|-----|-----|-----|-----|-----|------|
| TDCOPY.PRG | 109= | 116= | 116 | 122 | 126 | 128 | 224 | 226 | 241= |
| 248= | | 248 | 255 | 262 | 264 | 277 | | | |

ZIP

| | | | | | | |
|--------------|-----|-----|-----|------|------|-----|
| ADDRESS.PRG | 130 | 227 | 282 | 301R | 336R | 363 |
| ADDRBOOK.PRG | 85 | 86 | 88 | | | |
| ROLODEX.PRG | 180 | 181 | 183 | | | |

Public Variable Summary

System: ToDo - To Do Management System
Author: Walter J. Kennamer
03/02/88 18:56:06
Public Variable Summary

These variables were declared PUBLIC somewhere in the system.
Some may also be used as private variables in some parts of the code.

| | |
|------------|-----------|
| ABORT | ADDR_MROW |
| ANSWER | CARD_ROWS |
| FOX | CMD_LINE |
| C_OFFSET | DATE_STR |
| . | |
| . | |
| S_TESTDATE | S_VERSION |
| T_COMP | T_DATED |
| T_DUR | T_ITEM |
| T_ITEMTYPE | T_PRIOR |
| T_SUBJ | T_TIME |

Macro Summary

System: ToDo - To Do Management System
Author: Walter J. Kennamer
03/02/88 18:56:06
Macro Summary

Macros Defined to FoxDoc

| Variable | Expansion |
|------------|-----------|
| S_TDFILE | TODO |
| S_DDNDX | TODODD |
| S0_ADFNAME | ADDRESS |

Macros Not Defined to FoxDoc

| | |
|-------------|--------------|
| &BAK_NAME | &B_FILE |
| &COLR_STR | &FIELD1 |
| &FIELD2 | &FIELD_NAME |
| &FILT_STR | &FLDNAME |
| &FNAME | &IN_DB |
| &IN_SELECT | &IN_VAL |
| &M_WORKAREA | &NTXNAME |
| &NUM | &PROGNAME |
| &REP_FNAME | &REV_COLR |
| &S0_COLR1 | &S0_COLR2 |
| &S0_DATA | &S0_PRMNAME |
| &S0_PROGRAM | &S0_USERFILT |
| &SRCHFLD | &TEMPIN |

Array Summary

System: ToDo - To Do Management System
Author: Walter J. Kennamer
03/02/88 18:56:06
Array Summary

An array declared with a variable (e.g., DECLARE foo[bar])
will be shown as having a size of [var].
FIELD_LENGTH[var]
FIELD_LIST[2]

File List

System: ToDo - To Do Management System
Author: Walter J. Kennamer
03/02/88 18:56:29
File List

Programs and procedures:

ADDRBOOK.PRG
ADDRESS.PRG
ADDRFILT() (function in ADDRESS.PRG)
ADDRLIST.PRG
.
.
.
TIMEFORM() (function in TDCALDAY.PRG)
TODO.PRG
UNHIGHLIGHT (procedure in SHOWCAL.PRG) VERPRT.PRG

Procedure files:

SUBJECT.PRG
TODOPRC.PRG

Databases:

.
.
.
HELP.DBF
HELP.DBT
SUBJECT.DBF
TODO.DBF

Index files:

&NTXNAME
DATEDUE.IDX
DATEDUE.IDX
HELPKEY.IDX
HISTDD.IDX
PRIORITY.IDX
SUBJECT.IDX
TODODD.IDX

Report forms:

SUBREPT.FRM
TDDDETIN.FRM
TDSUMIN.FRM

Memory files:

ACCESSES.MEM
CLIP.MEM
DEFAULT.MEM
LASTFILE.MEM
PRINTER.MEM

Advanced Topics

14 Optimizing Your Application

This chapter provides tips for optimizing your application, including using Rushmore and other general performance hints such as:

- Memory use
- Opening and closing files
- SET TALK OFF and SET DOHISTORY OFF
- Using arrays instead of macro substitution
- Gathering files into procedures
- SQL SELECT considerations
- Additional considerations
- Performance considerations for FoxBASE+ applications

The Rushmore Technology

The Rushmore technology is a data access technique that permits sets of records to be accessed very efficiently, at speeds comparable to single-record indexed access. It is called “Rushmore” because that was its internal project name selected after viewing Hitchcock’s “North By Northwest” the preceding evening.

With Rushmore, some complex database operations run hundreds or even thousands of times faster than before. FoxPro version 2.5 enables personal computers to handle truly gigantic databases, containing millions of records, at speeds comparable to mainframe database systems.

Rushmore utilizes standard FoxPro B-tree indexes and does not require any new type of file or index. It may be utilized with any FoxPro index: standard (.IDX) indexes as utilized in FoxPro versions 1.xx, compact (.IDX) indexes, or compound (.CDX) indexes.

In particular, Rushmore does not depend on the new compact index format. Compact indexes, both (.CDX) and (.IDX) format, utilize a compression technique that produces indexes as small as 1/6th the size of comparable old-format indexes. Compact indexes can be processed faster solely because they are physically smaller. This means that fewer disk accesses are required to process them and larger portions of the index can be retained in FoxPro’s memory buffers.

Although Rushmore benefits from the smaller size of compact indexes, as does all file access, it also functions very well with old format indexes.

When very large databases are being processed, Rushmore may not have sufficient memory to operate on smaller machines. In this circumstance, a warning message appears (“Not enough memory for optimization”) and execution proceeds as in earlier versions of FoxPro. Although no data will be lost and your program will function correctly, the query will not benefit from Rushmore.

Therefore, if you are processing large databases, we suggest that you use the Extended version of FoxPro version 2.5 (provided at no additional charge with the standard package).



A good rule of thumb is to use the Extended version if your databases have, in aggregate, more than 500,000 records.

In its simplest form, Rushmore speeds up single-database commands utilizing FOR clauses that specify sets of records in terms of existing indexes. Also, Rushmore can speed the operation of the commands listed in the table on the next page titled Potentially Optimizable Commands With FOR Clauses when SET FILTER is in effect and the filtering condition is specified in terms of existing indexes.

To take advantage of Rushmore with multiple databases, you must use the SQL SELECT command. FoxPro's SQL facility makes use of Rushmore as a basic tool in multi-database query optimization, utilizing Rushmore with existing indexes and even creating new ad-hoc indexes to speed queries.

Rushmore with Multiple Databases

To take advantage of Rushmore’s optimization when you are retrieving data from more than one database, you must use the SQL SELECT command. SQL uses Rushmore as a basic technology to optimize its queries.

When you use SELECT, all rules that you must normally follow to benefit from Rushmore are no longer in effect. SQL decides what is needed to optimize a query and does the work for you. You don’t need to open databases or indexes. If SQL decides it needs indexes, it creates temporary indexes for its own use.

Rushmore with Single Databases

With single databases, you can take advantage of Rushmore anywhere that a FOR clause appears. Rushmore is designed so that its speed is proportional to the number of records retrieved.

| Potentially Optimizable Commands With FOR Clauses | | | |
|---|---------|---------|-------|
| AVERAGE | COUNT | LIST | SORT |
| BROWSE | DELETE | LOCATE | SUM |
| CALCULATE | DISPLAY | RECALL | TOTAL |
| CHANGE | EDIT | REPLACE | |
| COPY TO | EXPORT | REPORT | |
| COPY TO ARRAY | LABEL | SCAN | |

In addition to an optimizable FOR clause expression, the commands in the table above must have a scope clause of ALL or NEXT to take advantage of Rushmore. Rushmore also works when you allow the scope to default to ALL.

When optimizing, Rushmore can utilize any open indexes except for *filtered* and *unique* indexes.



For optimal performance, don’t set the order of the database.

If you create indexes or tags, remember that this automatically sets the order. If you want to take maximum advantage of Rushmore with a large data set and you require the data in a specific order, issue SET ORDER TO to turn off index control, then use the SORT command.

Basic Optimizable Expressions

Rushmore technology depends on the presence of a *basic optimizable expression* in a FOR clause. A basic optimizable expression can form an entire expression or can appear as part of an expression. The rules for combining basic optimizable expressions appear in the section titled Combining Basic Optimizable Expressions.

A basic optimizable expression takes one of the following forms:

<index expression> <relational operator> <constant expression>

or

<constant expression> <relational operator> <index expression>

In a basic optimizable expression:

- <index expression> must exactly match the expression on which an index is constructed and <index expression> must not contain aliases
- <relational operator> must be one of the following: <, >, =, <=, >=, <>, #, !=
- <constant expression> may be any expression, including memory variables and fields from other *unrelated* databases

For example, if you have indexes on the following expressions:

```
FIRSTNAME
CUSTNO
UPPER (LASTNAME)
HIREDATE
ADDR
```

then the following are basic optimizable expressions:

```
FIRSTNAME = 'Fred'
CUSTNO >= 1000
UPPER (LASTNAME) = 'SMITH'
HIREDATE < {12/30/90}
```

If you issue the command STORE 'WASHINGTON AVENUE' TO X then the following are also basic optimizable expressions:

```
ADDR = X
ADDR = SUBSTR(X,8,3)
```

Combining Basic Optimizable Expressions

Rushmore’s data retrieval optimization is dependent on the FOR clause expression. With a simple or complex FOR clause expression, data retrieval speeds can be enhanced if the FOR expression is optimizable. This section explains the rules for combining basic expressions to create a FOR expression.

Basic expressions may be optimizable. Basic expressions can be combined with the AND, OR and NOT logical operators to form a complex FOR clause expression that may also be optimizable.

An expression created with a combination of optimizable basic expressions is fully optimizable. If one or more of the basic expressions are not optimizable, the complex expression may be partially optimizable or not optimizable.

A set of rules determines if an expression composed of basic optimizable or non-optimizable expressions is fully optimizable, partially optimizable or not optimizable. The rules for determining query optimization are outlined in the table below, followed by another table with corresponding examples.

| Combining Basic Expressions | | | |
|-----------------------------|----------|------------------|-----------------------|
| Basic Expression | Operator | Basic Expression | Query Result |
| Optimizable | AND | Optimizable | Fully Optimizable |
| Optimizable | OR | Optimizable | Fully Optimizable |
| Optimizable | AND | Not Optimizable | Partially Optimizable |
| Optimizable | OR | Not Optimizable | Not Optimizable |
| Not Optimizable | AND | Not Optimizable | Not Optimizable |
| Not Optimizable | OR | Not Optimizable | Not Optimizable |
| — | NOT | Optimizable | Fully Optimizable |
| — | NOT | Not Optimizable | Not Optimizable |

| Examples of Combined Basic Expressions | |
|--|--|
| Example | Expression Types, Operator and Result |
| FIRSTNAME = 'FRED' AND HIREDATE < {12/30/89} | Optimizable AND Optimizable = Fully Optimizable |
| FIRSTNAME = 'FRED' OR HIREDATE < {12/30/89} | Optimizable OR Optimizable = Fully Optimizable |
| FIRSTNAME = 'FRED' AND 'S' \$ LASTNAME | Optimizable AND Not Optimizable = Partially Optimizable |
| FIRSTNAME = 'FRED' OR 'S' \$ LASTNAME | Optimizable OR Not Optimizable = Not Optimizable |
| 'FRED' \$ FIRSTNAME AND 'S' \$ LASTNAME | Not Optimizable AND Not Optimizable = Not Optimizable |
| 'FRED' \$ FIRSTNAME OR 'S' \$ LASTNAME | Not Optimizable OR Not Optimizable = Not Optimizable |
| NOT FIRSTNAME = 'FRED' | The NOT Operator with Optimizable = Fully Optimizable |
| NOT 'FRED' \$ FIRSTNAME | The NOT Operator with Not Optimizable = Not Optimizable |

You can also use parentheses to group combinations of basic expressions. The rules above also apply to combinations of expressions grouped with parentheses.

Combining Complex Expressions

You can combine complex expressions to create a more complex expression that is fully optimizable, partially optimizable or not optimizable, as shown in the table above. These more complex expressions can, in turn, be combined to create expressions that again may be fully or partially optimizable, or not optimizable at all. The results of combining these more complex expressions are shown in the following table. These rules also apply to expressions grouped with parentheses.

| Combining Complex Expressions | | | |
|-------------------------------|----------|-----------------------|-----------------------|
| Expression | Operator | Expression | Result |
| Fully Optimizable | AND | Fully Optimizable | Fully Optimizable |
| Fully Optimizable | OR | Fully Optimizable | Fully Optimizable |
| Fully Optimizable | AND | Partially Optimizable | Partially Optimizable |
| Fully Optimizable | OR | Partially Optimizable | Partially Optimizable |
| Fully Optimizable | AND | Not Optimizable | Partially Optimizable |

| Combining Complex Expressions | | | |
|-------------------------------|----------|-----------------------|-----------------------|
| Expression | Operator | Expression | Result |
| Fully Optimizable | OR | Not Optimizable | Not Optimizable |
| — | NOT | Fully Optimizable | Fully Optimizable |
| Partially Optimizable | AND | Partially Optimizable | Partially Optimizable |
| Partially Optimizable | OR | Partially Optimizable | Partially Optimizable |
| Partially Optimizable | AND | Not Optimizable | Partially Optimizable |
| Partially Optimizable | OR | Not Optimizable | Not Optimizable |
| — | NOT | Partially Optimizable | Partially Optimizable |
| Not Optimizable | AND | Not Optimizable | Not Optimizable |
| Not Optimizable | OR | Not Optimizable | Not Optimizable |
| — | NOT | Not Optimizable | Not Optimizable |

The table below gives examples of how complex expressions may be combined and the extent to which the result is optimized.

| Examples of Combined Complex Expressions | |
|--|---|
| Example | Expression Types, Operator and Result |
| (FIRSTNAME = 'FRED' AND HIREDATE < {12/30/89}) OR (LASTNAME = '' AND HIREDATE > {12/30/88}) | Fully Optimizable OR Fully Optimizable = Fully Optimizable |
| (FIRSTNAME = 'FRED' AND HIREDATE < {12/30/89}) AND 'S' \$ LASTNAME | Fully Optimizable AND Not Optimizable = Partially Optimizable |
| (FIRSTNAME = 'FRED' AND 'S' \$ LASTNAME) OR (FIRSTNAME = 'DAVE' AND 'T' \$ LASTNAME) | Partially Optimizable OR Partially Optimizable = Partially Optimizable |
| ('FRED' \$ FIRSTNAME OR 'S' \$ LASTNAME) OR ('MAIN' \$ STREET OR 'AVE' \$ STREET) | Not Optimizable OR Not Optimizable = Not Optimizable |

When Rushmore Is Not Available

In rare cases, Rushmore may not be available to enhance data retrieval operations. In these cases, execution proceeds as in earlier versions of FoxPro.

Rushmore is disabled whenever it cannot optimize the FOR clause expression in a potentially optimizable command. See the earlier section, *Combining Basic Optimizable Expressions*, for guidelines on creating an optimizable FOR expression.

Rushmore is also disabled whenever a WHILE clause is included in a command that benefits from Rushmore.

In the Standard version of FoxPro, FoxPro may disable Rushmore when the total number of records in all open databases exceeds 500,000 records. However, the Extended version of FoxPro can use Rushmore to enhance performance with database record totals of more than 500,000 records, as well as with database record totals less than 500,000.

In low memory situations, Rushmore cannot optimize data retrieval. However, data retrieval performance will be on par with earlier versions of FoxPro.

Disabling Rushmore

In rare cases, you should disable Rushmore. When you issue a command that utilizes Rushmore, Rushmore immediately determines which records match the FOR clause expression. These records are then manipulated by the command.

If a potentially optimizable command modifies the index key in the FOR clause, Rushmore's record set can become outdated. In a case like this, you can disable Rushmore to ensure that you have the most current information from the database.

To disable Rushmore for an individual command, include the `NOOPTIMIZE` keyword with the command. To globally disable (or enable) Rushmore for all commands that benefit from Rushmore, use `SET OPTIMIZE`. The command `SET OPTIMIZE OFF` disables Rushmore and `SET OPTIMIZE ON` enables Rushmore. The default setting is `ON`.

For more information about the `SET OPTIMIZE` command, refer to the *FoxPro Language Reference*.

General Performance Hints

FoxPro can use extra memory for dynamic memory allocation — more available memory means faster execution and better performance.

Here are some additional suggestions to help you maximize FoxPro's performance. For information about optimizing your system, refer to the chapter titled *Optimizing Your System* in the FoxPro *Installation and Configuration* manual.

Memory Use

FoxPro is designed to take advantage of the latest memory technology. It can effectively make use of lots of memory. So, one of the best ways to optimize FoxPro's performance is to give it *lots* of memory in which to work.

As you create windows, menus, screens, memory variables and other objects, you use available memory. To maximize performance, avoid creating objects before you need them and remember to clear objects when you finish with them to free memory for FoxPro. SYS(1016) returns the amount of memory being used by objects that you control — windows, menus, screens, memory variables, open databases, etc.

Opening and Closing Files

In applications, opening and closing files frequently slows execution. FoxPro offers 25 work areas so that you can keep databases open when they're used frequently in an application.

SET TALK OFF and SET DOHISTORY OFF

FoxPro can display information on your computer's screen much faster than any competing product. However, FoxPro may not operate at its fastest when you SET TALK ON.

Of course, the amount by which FoxPro is slowed down depends on the particular operation and the amount of talking generated. Under MS-DOS, we have observed situations where activating the TALK option slowed FoxPro down by a factor of two to three times.

SET DOHISTORY ON is useful when debugging because it displays commands from programs in the Command window as they are executed. However, DOHISTORY is intended as a *temporary* debugging aid and causes programs to execute many times slower.

Name Expressions Instead of Macro Substitution

FoxPro supports name expressions. These should be used in many contexts which previously required the use of macro substitution.

If you can use name expressions instead of macro substitution, program performance will greatly improve.

```
STORE "CUST" TO file
USE &file      ← slow
USE (file)     ← fast
```

```
STORE "OUTPUT" TO newwin
DEFINE WINDOW &newwin FROM 2,1 TO 13,75 CLOSE FLOAT GROW ← slow
DEFINE WINDOW (newwin) FROM 2,1 TO 13,75 CLOSE FLOAT GROW ← fast
```

Gathering Files into a Project

FoxPro permits an unlimited number of programs and procedures to be combined in a single file. The Project Manager provides an easy way to do this. Gathering an application's programs and procedures together into one or two files can greatly increase program execution speed for a couple of reasons.

First, after FoxPro opens a program file, it leaves it open. When you later DO a FoxPro program that's contained in the file, no additional file opening or searching is required.

Second, having only one or two files reduces the number of files that are contained in the working directory. With fewer directory entries for MS-DOS to examine when opening, renaming or deleting files, the speed of all file operations is increased.

SQL SELECT Performance Considerations

When using a SQL SELECT command, the following situations can degrade performance and produce unexpected results:

- If you include two databases in a query and don't specify a join condition, every field in the first database will be joined with every field in the second database as long as the filter conditions are met. This can produce enormous query results.
- Use caution when joining databases with empty fields because FoxPro will match empty fields. For example, if you join on CUSTOMER.ZIP and INVOICE.ZIP and CUSTOMER contains 100 empty zipcodes and INVOICE contains 400 empty zipcodes, the query output will contain 40,000 extra records resulting from the empty fields. To avoid this, you can use the EMPTY() function.

Additional Considerations

In general, the following are true:

- Sending output to any window except the topmost window is slower. Causing display to scroll behind a window is a near-worst case.
- FOR ... ENDFOR loops are faster than DO WHILE ... ENDDO loops.
- SQL INSERT is much faster than using APPEND BLANK then REPLACE, particularly with an indexed database in a multi-user environment.
- When you are scattering multiple fields, SCATTER TO ARRAY is faster than SCATTER MEMVAR.
- If you need to append a large number of records to an indexed database, it may be faster to remove the index, append the records and reset the index.
- If you usually use a certain index order, you'll notice improved performance if you periodically sort the database in this order.
- CDXs improve multi-user performance because one CDX can be updated faster than multiple IDXs.
- Remember that *all* CDX tags are always open (when the associated database is in use) and must be updated whenever a key value changes. If you have lots of tags, this can significantly degrade the speed at which records can be added.



Remember that there are always exceptions so you must determine what's best for your situation.

Performance Considerations for FoxBASE+ Applications

If you are running a program in FoxPro that uses only FoxBASE+ features, the following features can be disabled for improved performance:

- Do not load a mouse
- SET SYSMENU OFF
- SET RESOURCE OFF

15 Compatibility

This chapter contains information for those who are upgrading from earlier versions of FoxPro or FoxBASE+, or who have database applications to import from other software programs.

Topics covered include:

- Approaches to achieving FoxBASE+ compatibility
- Importing files from earlier versions of FoxPro

FoxBASE+ Compatibility

FoxPro runs most FoxBASE+ programs without change. When compatibility with FoxBASE+ is needed, you should inform FoxPro of the desired compatibility by using the statement:

```
SET COMPATIBLE FOXPLUS
```

We've also provided a special configuration file that will configure FoxPro to be nearly 100% compatible with FoxBASE+. This configuration file is in the FOXPRO25\GOODIES\MISC directory and is called CONFIG.FOX. To make use of it, simply copy the file as CONFIG.FP into the home directory or incorporate the commands it contains into your current CONFIG.FP file.

Emulating FoxBASE+ Keystroke Assignments

FoxPro's built-in keystroke shortcuts are very similar to those in FoxBASE+ for the Macintosh and are also similar to those of IBM's System Application Architecture (SAA) specification. They are *very different* from those of FoxBASE+.

If your programs rely on the exact keystrokes used to operate FoxBASE+ and the READKEY() key codes that they generate, you will need to use another file that we've provided: FOXPLUS.FKY. This file is a collection of keyboard macros, created with FoxPro's macro creation facility, that emulate the behavior of FoxBASE+. FOXPLUS.FKY can be found in the FOXPRO25\GOODIES\MISC directory.

The macros in this file are activated by restoring the macro file through the Keyboard Macros dialog or by using the command

```
RESTORE MACROS FROM FOXPLUS.FKY
```

These macros can also be restored automatically at startup by renaming the file to DEFAULT.FKY and placing it in the directory where FoxPro resides.

| FOXPLUS.FKY Control Key Definitions | |
|--|--|
| Key(s) | Action |
| MOVING FORWARD | |
| Ctrl+D, Ctrl+L | Next character (right) |
| Ctrl+X | Next line |
| Ctrl+F, End | Next word |
| Ctrl+C, PgDn | Next screen |
| Ctrl+B, Ctrl+Right Arrow | Pan right (in text) |
| Ctrl+M, Carriage return | Next field |
| MOVING BACKWARD | |
| Ctrl+S | Previous character (left) |
| Ctrl+E, Ctrl+K | Previous line |
| Ctrl+A, Home | Previous word |
| Ctrl+R, PgUp | Previous screen |
| Ctrl+Z, Ctrl+Left Arrow | Pan left (in text) |
| INSERT MODE | |
| Ctrl+V | Toggle insert mode on/off |
| DELETING | |
| Ctrl+G | Delete character at cursor |
| Ctrl+H | Delete last character entered |
| Ctrl+T | Delete word |
| Ctrl+U | Delete record toggle (only in BROWSE, CHANGE and EDIT) |
| Ctrl+Y | Delete to end of line |
| EXITING | |
| Ctrl+Q, Escape | Abort and exit |
| Ctrl+W, Ctrl+End | Save changes and exit |
| MISCELLANEOUS | |
| Ctrl+PgDn, Ctrl+Hyphen | Open memo field for editing |
| Ctrl+PgUp, Ctrl+^ | Exit from memo field |

Additional SET Options for FoxBASE+ Emulation

Some of the SET commands in the following table may be required for complete compatibility with FoxBASE+, but are not really desirable in the new context of FoxPro. A good example is SET STATUS ON, which activates the old-style status bar — not really useful in FoxPro but provided for compatibility with FoxBASE+.

| SET Options for FoxBASE+ Compatibility | |
|--|--|
| Command | Action |
| SET BRSTATUS ON | Causes the status bar to appear automatically with BROWSE. |
| SET MACKEY TO | Disables the key which displays the Macro Definition dialog. |
| SET NOTIFY OFF | Turns off FoxPro system messages. |
| SET SCOREBOARD ON | Turns on the old-style scoreboard. |
| SET STATUS ON | Turns on the old-style status bar. |

For compatibility with FoxBASE+, you should include the following statements in your CONFIG.FP file:

```
BRSTATUS = ON
MACKEY = <expC>
NOTIFY = OFF
SCOREBOARD = ON
STATUS = ON
```

Data and Program Files

When FoxPro creates a database file that contains a memo field, the associated memo file uses an extension of .FPT, as opposed to the .DBT file extension used for memo files in FoxBASE+. FoxPro *does* read and write the old format .DBT memo files when they are encountered and can create a new database in the old format as long as you include the TYPE FOXPLUS clause with the COPY TO command.

Also, take note of the difference between compiled FoxPro program files (with an .FXP extension) and those used by FoxBASE+ (.FOX files).

Unavoidable Differences

While FoxPro is as perfectly compatible with FoxBASE+ as possible, the following areas should be avoided or modified if you plan to run existing FoxBASE+ programs in FoxPro.

Error Reporting

FoxPro detects a few error conditions which are ignored by FoxBASE+. If your FoxBASE+ programs contain these errors, they will not operate under FoxPro as they do under FoxBASE+, but will generate error messages.

Many New Functions and Keywords

FoxPro has many new functions and, therefore, many new keywords. If the name of one of these built-in FoxPro functions has been used in a FoxBASE+ application as a UDF (user-defined function), FoxPro will interpret the name as a built-in function and will not execute the UDF. An example is the EVALUATE() function.

SET COLOR TO

In FoxBASE+, issuing the command SET COLOR TO with no arguments resets the screen to the default colors of black and white. In FoxPro, no color changes are made when SET COLOR TO is issued with no arguments.

Color Commands

In color commands, specifying the color U causes underlining on monochrome monitors. FoxBASE+ ignores the color U in a color pair and uses the color black in its place. In FoxPro:

- If you specify U anywhere in a color pair, the foreground will be blue.
- If you specify a background color of U or you don't specify a background color, the background will be black.
- If you specify a background color other than U, you will get the expected color.

In color commands, specifying the color I causes inverse video on monochrome monitors. FoxBASE+ ignores the color I in a color pair and uses the color black in its place. In FoxPro, specifying I as the background or the foreground color in a color pair always gives a black foreground and a white background.

SET DEFAULT

In FoxBASE+, when you issue the SET DEFAULT command with a drive and directory, only the default drive is set. The following command gives a default of C: in FoxBASE+:

```
SET DEFAULT TO C:\FOXBASE\PROGRAMS
```

In FoxPro, the SET DEFAULT command sets the MS-DOS drive *and* directory defaults exactly as you specify.

Using the Null Character

Any program statements that have been written to rely on the fact that null characters cannot be placed in a string will not operate under FoxPro as under FoxBASE+. For instance, if a FoxBASE+ application initializes a string to the null string (containing no characters) with the command:

```
mystring = CHR(0)
```

as opposed to using the normal technique of

```
mystring = ""
```

the application won't work under FoxPro as it did in FoxBASE+. In the previous example, FoxPro (which permits the null character to be stored to strings) would create a string that contains one character, the null character.

.VUE Files

FoxBASE+ .VUE files contain information about databases, indexes, aliases, format files, relations, fields lists, filters and ON/OFF settings at the time the .VUE file was saved. FoxPro .VUE files contain this information and additional information about the default drive, directory path, alternate file, procedure file, help file, resource file, index order of the database, clock position and setting, currency string and ON/OFF settings.

Interface Facilities

Because FoxPro's interface and interactive facilities (like BROWSE and CHANGE/EDIT) look entirely different from those of FoxBASE+, programs that rely on the precise physical appearance of these facilities in FoxBASE+ will not operate identically in FoxPro.

SET COMPATIBLE

The FoxPro SET COMPATIBLE command allows you to specify command compatibility with FoxBASE+ or dBASE IV. For more information, refer to the SET COMPATIBLE command in the FoxPro *Language Reference*.

Converting Files from FoxBASE+ 2.10

Because FoxPro is upwardly compatible with FoxBASE+ 2.10, virtually no changes need to be made to existing FoxBASE+ programs to DO them in FoxPro. Just run your programs as you did before.

FoxPro recognizes the following FoxBASE+ file types:

- Source programs (.PRGs) – These are automatically compiled to FoxPro compatible object programs (.FXPs). See the section that follows about .FOX program files.
- Database files (.DBFs)
- Memo files (.DBTs) – While FoxPro recognizes .DBT-style memo files, it only converts them to FoxPro compatible memo files, with an extension of .FPT, when a new file is created from the old structure. FoxPro can read and write to .DBT memo files without converting them to the new format, and it *does not automatically replace* the original .DBT files.
- Index files (.IDXs) – See the section that follows about .NDX files.
- Memory variable save files (.MEM files)
- Screen format files (.FMT files) – FoxPro automatically compiles .FMT files into .PRX files whenever you execute the SET FORMAT TO command.
- Report form files (.FRM files) – While the FoxPro Report Writer creates report form files in a different format than the FoxBASE+ format, FoxPro can print reports using the old .FRM format. The Report Writer can also read .FRM type reports for modification, although it cannot save report files in the old .FRM format.
- Label definition files (.LBL files).

The file types listed above require no modification before using them with FoxPro — FoxPro handles all file conversion for you.

.NDX Index Files

As in previous Fox products, the indexes used to access FoxPro databases are different in format and usually much smaller than comparable dBase indexes. This is due to the state-of-the-art indexing technique used by FoxPro.

If you're running applications that contain dBASE® .NDX index files, you don't have to worry about converting your index files to FoxPro format — we take care of it for you!

When you run a dBASE application in FoxPro and use a dBASE-style index file, FoxPro immediately and automatically reindexes the database, creating a FoxPro index file. The following message appears:

```
dBASE III index - rebuilding
```

By default, these new index files are created with an extension of IDX. Some applications, however, may be written to require that index files have an .NDX extension. If that's the case, add the following line to the CONFIG.FP file:

```
INDEX = NDX
```

This causes FoxPro to use the .NDX extension with *all* index files it creates. Of course, using this option will cause the original dBASE indexes to be destroyed during the automatic reindexing process.

.DBT Memo Files

FoxPro memo files (.FPTs) are slightly different than those used by FoxBASE+, dBASE III PLUS® and dBASE IV® (.DBTs) in that you can store *any type of data in them*. The old style .DBT files only hold ASCII text data. FoxPro, however, can read and write to .DBT memo files without converting them to the new format.

FoxPro creates a new .FPT type memo file from an existing .DBT file in two situations:

1. Whenever you create a new database structure from an existing database that contains a memo field and has an existing memo file. The COPY TO command is an example.
2. Whenever you modify the structure of a database that has an associated memo file. When the structure changes are saved, the .DBT file is converted to a new .FPT file.

To create a copy of a FoxPro database file that contains a memo field in the form that FoxBASE+ can understand, issue the following command:

```
COPY TO <filename> TYPE FOXPLUS
```

You should do this when your memo fields contain only characters which can be processed by FoxBASE+. The forbidden characters, such as Ctrl+Z (CHR(26)) and Null (CHR(0)), should not be included.

FOX Program Files

FoxPro handles program compilation and management somewhat differently than in previous versions.

If you have existing FoxBASE+ programs that you'd like to execute, take note that FoxPro does not recognize or execute .FOX compiled programs. Existing source files need to be compiled into .FXPs before they can be executed. FoxPro automatically compiles any source .PRG file for which it cannot find a compiled .FXP object file with the same name. You can also use one of the manual compile options.

Compiling Programs

When you execute a program, FoxPro looks for a file with an .FXP extension. If it cannot find one, FoxPro automatically compiles the source program, creating an .FXP object file. Also, if you SET DEVELOPMENT ON (the highly-recommended default), FoxPro automatically recompiles the source program if it has a creation date (or time) that's newer than the existing .FXP file.

If you want to manually compile your program files, you can do so by using the **Compile** option on the **Program** menu popup or by issuing the following FoxPro command:

```
COMPILE <program>
```

Both approaches instruct FoxPro to compile one or more source program files (.PRGs) into object program files (.FXPs). FoxPro assumes an extension of .PRG if one is not explicitly given; you must add a file extension only when compiling source files that do not have a .PRG extension.

For complete information on the syntax and options that are available when compiling programs, see the COMPILE command in the FoxPro *Language Reference* or the Program Menu chapter in the FoxPro *User's Guide*.

Executing Programs

If you're an application developer or a user with existing programs, you can start your programs from the interactive interface or under program control. For information about compiling programs, see the previous section.

A FoxPro program may be executed using any of several different methods.

From the Menu System

You can start a FoxPro program by choosing **Do...** from the **Program** menu popup, then choosing the program from the dialog that appears. For complete information on executing programs from the menu system, refer to the Program Menu chapter in the *FoxPro User's Guide*.

From the Command Window

You can also start a FoxPro program by typing the appropriate command (DO <filename>) in the Command window.

From the MS-DOS Prompt

Another method for executing a program under FoxPro is to include the name of the program on the MS-DOS command line when you execute FoxPro, as in:

```
FOXPRO <filename>
```

This causes FoxPro to start, then automatically execute the specified program file. You don't have to specify the .FXP or .PRG file extension. FoxPro will load and execute the compiled .FXP version of the program file if it's available; otherwise, the source file will be compiled and the resulting .FXP file will be executed. If the file cannot be found, the "File does not exist" message appears.

Using the CONFIG.FP File

You can also include the following statement in the CONFIG.FP:

```
COMMAND = DO <filename>
```

If an executable program name is not provided at the MS-DOS prompt when FoxPro is started, the file named in this command is executed. As with the previous technique, if the named file cannot be found, FoxPro displays the “File does not exist” alert.

Using a Batch Command File

Rather than always including a program file name when executing FoxPro, you can create a batch command file to perform the same program startup. This may be desirable for any number of reasons, including simplicity of use.

A batch command file is nothing more than a file containing one or more commands that the computer’s operating system will execute automatically. In MS-DOS, batch command files have a file name extension of BAT.

Batch command files may be created using any text editor. Complete information on creating batch commands can be found in your MS-DOS manuals.

As an example, to create a batch command file called ACCNT.BAT which will execute FoxPro and start the program named ACCT1, you would need to enter the following line into a file created with the FoxPro text editor:

```
FOXPRO ACCT1
```

Then save the file as ACCNT.BAT.

You could then type the following command at the system prompt to load FoxPro and execute the program file named ACCT1:

```
ACCNT
```

Converting Files from FoxPro 1.XX

Files you created in FoxPro 1.XX generally work in FoxPro version 2.5 without any conversion. However, please read the following tips:

- Keyboard macros – Keyboard macros defined in earlier FoxPro versions may not work in FoxPro version 2.5, since some of the menu structures and dialogs in the interface have changed.
- Programs with arrays – The command DIMENSION formerly recreated the array and initialized all elements to false (.F.). Now, array elements remain intact. Programs relying on the earlier behavior of this command should be revised by first releasing the array, then redimensioning.
- .FXP files – Programs compiled in FoxPro 1.02 are automatically recompiled to run under version 2.5. FoxPro 1.XX will not automatically recompile FoxPro version 2.5 object files. You must explicitly compile the programs or delete the FoxPro object files before running in FoxPro 1.XX.
- FOXUSER resource file – The FIXUSER.PRG program included with FoxPro version 2.5 should be run to allow 2.5 compatibility. FIXUSER.PRG is in FOXPRO25\GOODIES\MISC. The program will add one preference to the FOXUSER file for each entry that corresponds to a FoxPro 1.XX entry. The FoxPro 1.XX preferences will not be overwritten. This process duplicates the entries for these parameters: Color sets, Calculator preferences, Puzzle, Modify memo windows, and Label Layout windows. FIXUSER.PRG does not create 2.5 Browse preferences — you must recreate each 1.XX Browse preference.
- Labels – Label files from 1.XX can be opened transparently in FoxPro version 2.5. For labels in 2.5 you can also add aliases to fields. In addition, the environment is saved differently.

Once you save a label file in 2.5, attempting to open the same file in 1.XX generates the message “Label file invalid.”

- Reports – Reports and their environments created in 1.XX can be loaded into FoxPro version 2.5 without conversion. Once saved in 2.5, attempting to open the same report file in 1.XX generates the message “Report file invalid.” If you need to use a report in 2.5 and in an earlier version of FoxPro, save it under a different name for each.

- CDX, Compact IDX files – Index files created in earlier versions of FoxPro are still valid in 2.5. For better performance, you may wish to recreate your existing (.IDX) indexes in 2.5. The choices are:
 - Create a *compact* .IDX index, for improved speed. Compact single entry (.IDX) index files are useful if you are using an index for a limited time and plan to delete the file when you're finished using it. The number of .IDX files you can have open per database file is limited by file handles.

If you are maintaining compatibility with older versions of FoxPro, or sharing files between FoxPro and FoxBASE+ or FoxBASE+ for the Macintosh, you must use non-compact .IDX index files. Otherwise, if you build .IDX files always include COMPACT to benefit from FoxPro 2.5's faster access index technology.
 - Create a *compound* .CDX index (automatically compact). Compound .CDX index files contain multiple index entries called tags.
 - Create a *structural* .CDX index. Structural .CDX index files are automatically opened when the database is opened, so generally they are the preferred index type. Non-structural .CDX indexes are *independent* compound indexes.
- Related files respect the setting of deleted.
- Arrays are always passed by reference. Passing an array in FoxPro 1.XX would pass the first array element by value.
- Recompiling program applications from FoxPro 1.XX: Procedure and program names beginning with the "@" character generate an error when recompiled in FoxPro version 2.5.

16 FoxPro in a Multi-User Environment

FoxPro in a multi-user environment has all the features of FoxPro in a single-user environment while allowing users to share database files.

This chapter describes the features and functions of FoxPro in a network environment. However, it does not describe how to install FoxPro for use in such an environment. For details about installing FoxPro, refer to the *FoxPro Installation and Configuration* manual.

Information regarding the various networks supported by FoxPro is outside the scope of this chapter. If you have questions about the maximum number of users supported by a network, the minimum amount of memory needed or additional hardware that might be required by such systems, refer to your network documentation.

System Configuration

Network performance can be greatly enhanced by using some or all of the FoxPro system configuration options.

Temporary Work Files

During execution, FoxPro creates temporary work files. These files typically have names that contain an arbitrary string of digits or characters and a .TMP extension. The three categories of work files are:

Program Cache One of the work files is called the “program cache.” FoxPro tries to limit the size of this file to 256K, but it can become larger. The speed of access to this file is critical to FoxPro’s performance. If possible, put this file on the local work station, *not* the file server, and on the fastest storage device. Because file size is not a major factor, RAM disks are particularly suitable for the program cache.

Text Editor Work Files During editing sessions, the text editor creates work files that can become as large as the documents being edited. These files should be placed on a fast storage device, one with sufficient disk space for copies of all documents being edited. Again, it’s preferable that they *not* be situated on the network server, but on the local work station.

Sort and Index Work Files These files are created when databases are sorted and indexed. It’s possible for them to be up to three times the size of a database being sorted, so they should be situated on the fastest device with plenty of free disk space — how much free disk space depends on the size of your databases and indexes. It’s preferable that they *not* be situated on the network server, but on the local work station.

All three types of temporary work files can be independently situated in different directories using the configuration options that follow.

CONFIG.FP

With a CONFIG.FP file, you can redirect temporary files, establish initial SET command defaults and change other settings that can improve network performance. On a network, different work stations typically require different startup configurations. For example, different work stations might require different color sets, certain users might want to automatically execute a certain program at startup or a personalized keyboard macro definition file might need to be restored. To handle this wide range of configuration possibilities, individual CONFIG.FP files should be used.

If all users on the network require the same settings at startup, one CONFIG.FP file can be created and placed in the file server's directory containing FoxPro. FoxPro looks here first.

If FoxPro cannot find a CONFIG file, the default FoxPro settings are used.

To use a configuration file that resides in a directory other than the one from which FoxPro is started, do one of the following:

- Include the directory in your PATH. FoxPro automatically searches the MS-DOS PATH for a copy of CONFIG.FP if it cannot be found in the current working directory. FoxPro will use the first CONFIG.FP file it encounters. (See your MS-DOS manual for instructions on setting a MS-DOS PATH.)
- Use the MS-DOS SET command to specify a MS-DOS environmental variable that tells FoxPro which configuration file to use:

```
SET FOXPROCFG=<path name>
```

Place this command in the work station's AUTOEXEC.BAT file. (See your MS-DOS manual for full details on the SET command.)

- Specify the configuration file to use at startup by using the -C command line option:

```
FOXPRO -C<path name>
```

The command line option consists of a hyphen, followed by an upper or lower case C followed by a filename (with its full path, if necessary). No embedded spaces are allowed. This option will override the MS-DOS FOXPROCFG environment variable if one has been created.

Special CONFIG Options

Several CONFIG.FP network configuration options are available.

EDITWORK = <directory>

This option specifies the directory where the text editor places its temporary work files. Under some circumstances, these temporary work files can become as large as the original file, so be sure there's plenty of room on the disk containing this directory.

SORTWORK = <directory>

This option specifies where commands that use temporary work files, such as SORT and INDEX, will place their work files. SORT and INDEX can require disk space up to twice the size of the file being sorted or the index being constructed, so be sure there's plenty of room on the disk containing this directory.

PROGWORK = <directory>

This option specifies where the temporary program cache file is placed. You might want to put this file in a RAM disk or on a local work station drive. FoxPro tries to keep the size of this file less than 256K, but it can grow larger.

TMPFILES = <drive>

This option sets the drive to which the EDITWORK, SORTWORK and PROGWORK files will be stored if the other options are not included.

The configuration file and the configuration settings are explained in greater detail in the Customizing FoxPro chapter of the FoxPro *Installation and Configuration* manual.

FOXUSER Resource File

The FOXUSER resource file contains information that can be specific to each user. This information includes color sets, keyboard macros, preferences, system window locations and sizes, diary entries and so on. When FoxPro is started, it looks for the resource file that's specified in the CONFIG.FP file with:

RESOURCE = <pathname>

<pathname> can be either a directory or a fully-qualified path with the resource file name. If it's a directory, FoxPro looks for a resource file named FOXUSER in that directory. If a fully-qualified path and file name is included, FoxPro looks for the specified file.

If the CONFIG.FP configuration file does not contain a resource specification, FoxPro then searches the default directory for the resource file. If it cannot find the resource file in the default directory, FoxPro then searches the MS-DOS path. If a FOXUSER file cannot be located, one is created in the default directory using default settings.

Programming in a Multi-User Environment

On a network you must manage the collisions that occur when users share database files. To help you manage contention for database files, FoxPro provides complete record and file locking.

Exclusive Use versus Shared Use

FoxPro provides two methods of accessing database files: exclusive use and shared use.

Exclusive Use

When a database is opened for exclusive use on a workstation, only *one* user has access to the data. No other user can open the database for reading or writing. Because exclusive use defeats many of the benefits of sharing data on a network, it should be used sparingly and only when it's absolutely necessary.

A database file is opened for exclusive use with one of the following:

```
SET EXCLUSIVE ON  
USE <filename>
```

or

```
USE <filename> EXCLUSIVE
```

Opening a file for exclusive use is the only way to prevent other users from gaining read access to the database. Locking the database with `FLOCK()` prevents other users from writing to the file but does not prevent other users from *reading* the database.

SET EXCLUSIVE is ON by default.

Commands that Require Exclusive Use

The following commands require you to open a database for exclusive use.

- INDEX when creating, adding or deleting a compound index tag
- INSERT [BLANK] (not SQL INSERT)
- MODIFY STRUCTURE (Also operates in read-only mode when the file is not opened exclusively.)
- PACK
- REINDEX
- ZAP

The error “Exclusive open of file is required” is returned if you try to execute one of these commands on a shared database (a database not opened exclusively).

Shared Use

When a database is opened for shared use, more than one work station can have the same database open at the same time. Commands that write to a shared database require that a record in the database or the entire database be locked before the command is executed.

You can lock a record or a database opened for shared use in the following ways:

- Use a command that performs an automatic record or file lock. (A table of commands that perform automatic locking can be found in the section Commands that Perform Automatic Locking.)
- Manually lock one or more records or an entire database file with the record and file locking functions.

Associated memo and index files are always opened with the same share status as their database.



If a database is included in your application that is opened for lookup purposes only and is accessed by all users of the application, flag the database read-only with the MS-DOS ATTRIB command for faster performance.

Write Access versus Read-only Access

Commands that modify a database file require write access to a database. With write access, a record or the entire database must be locked before the command can be executed. In most commands, locking is handled automatically. However, you can manually lock the database or record before executing the command.

By comparison, FoxPro commands that read but do not modify data do not require that any portion of the database be locked. Also, read-only commands will operate on a database if another user has a record or the entire file locked. However, if a database file is opened exclusively neither write nor read-only access is available.

For example, REPORT FORM, which only reads a database file, will operate on any database that has not been opened exclusively by another user. This is true for other commands, such as TOTAL, SUM, SQL SELECT and SORT, that only read files.

In cases where completely current information is required, for example with general ledger reports, lock the file before reporting from it.

A list of read-only commands where automatic file locking can be enabled can be found in the SET LOCK topic in the *FoxPro Language Reference*.

Record and File Locking

A command that writes to a database record or records must lock the record or the entire database. This prevents two users from modifying the same record (or file) at the same time.

Record locking, whether automatic or manual, prevents one user from writing to a record that's currently being written to by another user. File locking prevents other users from writing to (but not reading from) a database file. File locking prohibits other users from updating records in a database and should only be used sparingly.

Automatic versus Manual Locking

Many FoxPro commands automatically attempt to lock a record or a database file before the command is executed. If the record or database is successfully locked, the command is executed and the lock is released. See the table on the following page for a list of commands that automatically lock the database.

You can manually lock a record or a database file with one of the manual locking functions (RLOCK(), LOCK(), FLOCK()). Once a record or database is locked, be sure to release the lock as quickly as possible to provide access to other users.

If an attempt to lock a record or file fails, the setting of SET REPROCESS and the current ON ERROR routine determine if the lock is attempted again. For more information about these commands, see the *FoxPro Language Reference*.

Unlocking Records and Database Files

The following commands release manual and automatic record and file locks:

- UNLOCK releases record and file locks in the current work area.
- UNLOCK ALL releases all locks in all work areas.
- If MULTLOCKS is SET OFF, locking another record (either manually or automatically) will release a record lock.
- Toggling MULTLOCKS from ON to OFF or from OFF to ON implicitly performs an UNLOCK ALL, releasing all record and file locks in all work areas.
- Locking a file releases all record locks within that file.
- Closing the database with USE, CLOSE ALL, CLOSE DATABASE or QUIT releases all record and file locks.

Moving the record pointer off a manually locked record does not remove the lock from the record. Moving the record pointer off an automatically locked record releases the lock even if MULTLOCKS is SET ON.



If a record was autolocked in a UDF and you move the record pointer off the record and back on the record, the autolock will be released.

Commands that Perform Automatic Locking

The following table lists the commands that automatically lock records and database files.

| Commands that Automatically Lock Records and Files | |
|--|---|
| Command | What's Locked |
| APPEND | Entire database |
| APPEND BLANK | Database header (briefly) |
| APPEND FROM | Entire database |
| APPEND FROM ARRAY | Database header |
| APPEND MEMO | Current record |
| BROWSE, CHANGE and EDIT | Current record and all records from fields in related databases (specified by alias) once editing of a field begins |
| DELETE | Current record |
| DELETE NEXT 1 | Current record |
| DELETE RECORD <n> | Record <n> |
| DELETE <scope beyond one> | Entire database |
| GATHER | Current record |
| INSERT – SQL | Database header |
| MODIFY MEMO | Current record when editing begins |

| Commands that Automatically Lock Records and Files | |
|---|--|
| Command | What's Locked |
| READ | Current record (and all records from aliased fields) |
| RECALL | Current record |
| RECALL NEXT 1 | Current record |
| RECALL RECORD <n> | Record <n> |
| RECALL <scope beyond one> | Entire database |
| REPLACE | Current record (and all records from aliased fields) |
| REPLACE NEXT 1 | Current record (and all records from aliased fields) |
| REPLACE RECORD <n> | Record <n> (and all records from aliased fields) |
| REPLACE <scope beyond one> | Entire database (and all files from aliased fields) |
| SHOW GETS | Current record and all records from aliased fields. |
| UPDATE | Entire database |

If a record or database is locked by another user, or if the database is opened exclusively by another user, the record or file lock will fail. Commands that lock the current record return the error "Record is in use by another" if the record cannot be locked. Commands that lock an entire database return the error "File is in use by another" if the file cannot be locked.

BROWSE, CHANGE, EDIT and MODIFY MEMO do not lock a record until you edit the record. If a BROWSE, CHANGE or EDIT window contain fields from records in related databases, the related records are locked if possible. The lock attempt will fail if the current record or any of the related records are also autolocked by

another user. If the lock attempt is successful, you are allowed to edit the record, and the lock is released when you move to another record or another window is brought forward.

APPEND BLANK automatically locks the file header (briefly, while the record is being added). Because the file header is locked, special consideration must be given to the APPEND BLANK command.

Other users can share the file without causing a collision when the APPEND BLANK command is executed. However, if another user is also appending a BLANK record to the database file, an error can occur. The error returned when two or more users execute APPEND BLANK simultaneously is "File is in use by another."

SET REPROCESS

SET REPROCESS affects the result of an unsuccessful lock attempt. You can control the number of LOCK attempts or the length of time a lock is attempted with SET REPROCESS. Refer to the command later in this chapter.

Program Example

The following example shows how an automatic locking command can be used in conjunction with SET REPROCESS TO AUTOMATIC and an ON ERROR routine. Because REPROCESS is set to AUTOMATIC, FoxPro attempts to APPEND a blank record until it's successful or until the you press Escape. If you press Escape, the ON ERROR routine ERR_MSG is executed.

```
ON ERROR DO err_msg
SET REPROCESS TO AUTOMATIC
USE customer
APPEND BLANK
ON ERROR
* EOF: TEST.PRG

*** Err_Msg: A simple error routine
PROCEDURE err_msg
?? CHR(7) * Initialize the message strings
IF ERROR( ) = 108
    * Error: 'File is in use by another'
    WAIT WINDOW MESSAGE( ) + "YOU MAY TRY AGAIN LATER."
ELSE
    * Error is not: 'File is in use by another'
    WAIT WINDOW MESSAGE( ) + "SEE YOUR SYSTEM ADMINISTRATOR"
ENDIF
* End of procedure: Err_Msg
```

Manual Locking Functions

FoxPro has three manual locking functions: FLOCK(), RLOCK() and LOCK(). FLOCK() locks a file. RLOCK() and LOCK() are identical and can lock one or more records.

These three functions:

- Test the lock status of the record or file.
- If the record or file is unlocked, it is locked by the function and a logical true value (.T.) is returned.
- If the record or file cannot be locked, the function might attempt to lock the record or file again depending on the current setting of SET REPROCESS.

SET REPROCESS determines if the lock attempts continue a limited number of times or indefinitely (until the lock is successful placed or until the you cancel the lock attempts).

After the final lock attempt, a logical value true or false value is returned indicating if the lock was successfully placed.

If you want to test the lock status of a record without locking the record, use the SYS(2011) function.



Program Example

In the following example, the CUSTOMER database is opened for shared access. FLOCK() is used to lock the file. If the file is successfully locked, REPLACE ALL is used to update every record in the database. UNLOCK issued to release the file lock. If the file cannot be locked because another user has locked the file or a record in the file, a message is displayed.

```
SET EXCLUSIVE OFF
SET REPROCESS TO 0
* Open the customer database non-exclusively
USE customer
* If the file can be locked
IF FLOCK( )
    * Do replacements and unlock the file
    REPLACE ALL contact WITH PROPER(contact)
    UNLOCK
ELSE
    * The file cannot be locked, so output message
    WAIT "File in use by another." WINDOW NOWAIT
ENDIF
```

Collision Management

When developing multi-user applications, you must anticipate and manage the inevitable collisions that will result. A collision occurs when one user tries to lock a record or file that's currently locked by another user. Two users cannot lock the same record or database at the same time.

Your application should contain a routine to manage these collisions. If your application does not have a collision routine, the system can lock up in a deadly embrace. A *deadly embrace* occurs when one user has locked a record (or a file) and tries to lock another record that's locked by a second user who, in turn, is trying to lock the record that's locked by the first user. While such occurrences are rare, the longer that a record (or file) is locked the greater the chance of a deadly embrace.

Error Handling Routines

Designing a multi-user application or adding network support to a single-user system requires that you deal with collisions and trap for errors.

If a record or file is locked and an attempt is made to lock the record or file, an error message is returned. Your application should contain an ON ERROR routine to trap these errors and provide options if the record or file cannot be locked.

Your multi-user applications can use SET REPROCESS to automatically deal with unsuccessful lock attempts. This command, in combination with an ON ERROR routine and the RETRY command, can let you continue or cancel the lock attempts.

Program Example

```
* This sample program demonstrates how SET REPROCESS
* and ON ERROR can be used to manage user collisions
* in FoxPro.

* Error routine to execute if error occurs
ON ERROR DO err_fix WITH ERROR( ),MESSAGE( )

* Open files non-exclusively
SET EXCLUSIVE OFF

* Reprocessing of unsuccessful locks is automatic
SET REPROCESS TO AUTOMATIC
```

```
* Open database
USE customer

* Create APPEND FROM file if needed
IF !FILE('cus_copy.dbf')
    COPY TO cus_copy
ENDIF

* Main routine
DO app_blank
DO rep_next
DO rep_all
DO rep_curr
DO add_recs
ON ERROR
* End of main program

PROCEDURE app_blank
* Routine to append a blank record
APPEND BLANK
RETURN

PROCEDURE rep_next
* Routine to replace data in current record
REPLACE NEXT 1 contact WITH PROPER(contact)
RETURN

PROCEDURE rep_all
* Routine to replace data in all records
REPLACE ALL contact WITH PROPER(contact)
GO TOP
RETURN

PROCEDURE rep_curr
* Routine to replace data in current record
REPLACE contact WITH PROPER(contact)
RETURN

PROCEDURE add_recs
* Routine to append records from another file
APPEND FROM cus_copy
RETURN
```

Programming in a Multi-User Environment

```
*****
* Program: Err_fix.prg
* This program is called when an error is encountered
* and the user escapes from the wait process

PROCEDURE err_fix
PARAMETERS ermun, msg

* Define and activate error message window
DEFINE WINDOW err_win FROM 21,00 TO 24,79 ;
    COLOR SCHEME 7

DO CASE
    * Error: File in use by another.
    CASE ermun = 108
        line1 = "File cannot be locked."
        line2 = "Try again later..."

    * Error: Record in use by another.
    CASE ermun = 109 .OR. ermun = 130
        line1 = "Record cannot be locked."
        line2 = "Try again later..."

    *Unknown error
    OTHERWISE
        line1 = msg
        line2 = "SEE YOUR SYSTEM ADMINISTRATOR"
ENDCASE

* Activate the error window
ACTIVATE WINDOW err_win
* Report the error
@ 0, (WCOLS( )-LEN(line1))/2 SAY line1
@ 1, (WCOLS( )-LEN(line2))/2 SAY line2

* Pause
WAIT WINDOW

* Release the message window
RELEASE WINDOW err_win

RETURN
* End of procedure: Err_fix
```

The Low-Level File Functions

Files opened with Write or Read/Write access by FCREATE() and FOPEN() are opened exclusively.

Optimizing Performance

This section discusses how to get the best performance from FoxPro. Refer to the chapter *Optimizing Your System* in the *FoxPro Installation and Configuration* manual for more information on improving performance.

Place the Temporary Files on a Local Drive

FoxPro creates its temporary files in the default directory. You can specify an alternate location for these files by including the `EDITWORK`, `SORTWORK`, `PROGWORK` and `TMPFILES` statements in your `CONFIG.FP` configuration file.

These temporary files are created in the course of editing, indexing, sorting, and so on. Text editing sessions can also temporarily create a complete copy of the file being edited (if `.BAKs` are being created).

If local work stations have hard drives with plenty of free space, you can improve performance by placing these temporary work files on the local drive or in a RAM drive. Redirecting these files to a local drive or a RAM drive increases performance by reducing access of the network drive.

Sorted Files versus Indexed Files

When the data contained in a database file is relatively static, sequential processing of sorted databases *without an index open* will result in improved performance. This does not mean that sorted databases cannot or should not take advantage of index files — the `SEEK` command, which requires an index, is incomparable for locating records quickly.

`SEEK` can be used to locate a record with the index on. Once a record is located the index can be turned off.

Exclusive Use of Files

Commands that execute when no other users require access to the data, such as with overnight updates, can benefit by opening the data files for exclusive use. When files are open for exclusive use, performance improves because FoxPro does not need to test the status of record or file locks.

Length of Lock

Shorten the amount of time a record or file is locked to reduce contention between users for write access to a file or record.

Multi-User Commands and Functions

The following table summarizes the commands and functions that are used in a multi-user environment.

| Command or Function | Description |
|----------------------------------|--|
| BROWSE/CHANGE/EDIT | Displays, edits, or appends database records. |
| DISPLAY STATUS or LIST STATUS | Shows the status of the FoxPro environment. |
| ERROR() | Returns the number of the error that triggered an ON ERROR routine. |
| FLOCK() | Attempts to lock a database file and returns .T. if successful. |
| MESSAGE() | Returns the current error message or the contents of the line that caused the error. |
| NETWORK () | Returns .T. if you are using FoxPro on a network. |
| RETRY | Re-executes the previous command. |
| RLOCK() or LOCK() | Attempts to lock one or more database records. |
| SET EXCLUSIVE | Specifies that databases be opened for exclusive or shared use. |
| SET LOCK | Enables or disables automatic record or file locking. |
| SET MULTLOCKS | Enables or disables multiple record locking. |

| Command or Function | Description |
|---------------------|---|
| SET NOTIFY | Enables or disables the display of system messages. |
| SET PRINTER | Enables or disables output to the printer and specifies an output port. |
| SET REFRESH | Displays changes made to records by other users. |
| SET REPROCESS | Specifies how FoxPro controls unsuccessful record or file locks. |
| SET STATUS | Enables or disables the display of the status bar. |
| SYS(0) | Returns the machine number. |
| SYS(2011) | Returns the current record or file lock status. |
| UNLOCK | Releases record or file locks. |
| USE . . . EXCLUSIVE | Opens a database file for exclusive use on the network. |

For more information about these commands or functions, see the *FoxPro Language Reference*.

17 Printer Drivers

The FoxPro Report Writer and Label Designer let you specify the style of text that is output to your printer. For example, text in reports or labels can be in bold face, italicized, underlined, subscript or superscript.

For text to appear in different styles in printed output, you must specify a printer driver for your printer. A *printer driver* is an interface between FoxPro and your printer. FoxPro provides a set of printer drivers and complete support for creating your own user-defined printer drivers.

If your printer is supported by a printer driver included with FoxPro, you can interactively specify a printer driver for your printer. In addition to specifying a printer driver, you can create different setups for your printer. For example, you can create individual printer setups for particular page orientations, font sizes and styles, and so on.

If your printer isn't supported by the printer drivers included with FoxPro, you can create your own printer driver. FoxPro printer support features *complete* flexibility and totally open architecture, making it possible to create your own printer driver for any type of printer — even very elaborate drivers that support proportional fonts, generate tables, incorporate graphics, etc.

This chapter is divided into three sections:

- Printer Driver Overview

Explains how printers are supported in FoxPro.

- Using FoxPro's Sample Printer Drivers and Printer Driver Setups

Describes how to specify a printer driver for your printer through the dialog supplied, and how to create different setups for your printer. This information allows you to interface your printer with FoxPro to print custom reports and labels.

- **Creating Custom Printer Drivers and Printer Driver Setup Applications**

Describes how to create your own printer drivers and printer driver setup applications. It is intended for advanced FoxPro users with programming experience and a familiarity with the FoxPro language.

Printer Driver Overview

Included with FoxPro are sample printer driver programs and an application that lets you interactively specify a printer driver for your printer. A printer driver is a program that lets FoxPro control the style of text output by your printer.

If the application provided with FoxPro suits your printing needs, it can be used “as is”. You can also modify the sample printer application and printer driver programs, or use them as a model to create your own printer drivers and printer application.

All the files used to provide printer support in FoxPro are contained in a project named GENPD.PJX. GENPD.APP, an application built from GENPD.PJX, is an executable program that ties together all the related printer support files. The support files include printer driver programs, databases containing printer specific information, and programs created from screen sets.

You can create printer driver setups in FoxPro — a printer driver setup stores printer settings (page orientation, font style and size, etc.). When you specify a printer driver setup in FoxPro, GENPD.APP is executed. Based on the settings in your printer driver setup, GENPD.APP loads printer control codes from a database into a special memory variable array and specifies the proper printer driver program.

When you print a report or labels or use DISPLAY, LIST or TYPE to send output to a file or printer, the printer driver program uses the printer control codes stored in the special memory variable array to format the printed output.

Here is a complete listing of the files included in GENPD.PJX.

| File | Type | Use |
|--|----------------------|---|
| GEN_PD.PRG | Program | Printer driver interface program |
| Screen sets used by GEN_PD.PRG | | |
| PD_EDIT.SCX | Screen Set Database | Printer Setup Editing Dialog |
| PD_EDIT.SCT | Screen Set Memo File | Printer Setup Editing Dialog |
| PD_SETUP.SCX | Screen Set Database | Printer Driver Setup Dialog |
| PD_SETUP.SCT | Screen Set Memo File | Printer Driver Setup Dialog |
| USR_PROC.SCX | Screen Set Database | Printer Application Procedure Dialog |
| USR_PROC.SCT | Screen Set Memo File | Printer Application Procedure Dialog |
| Databases and related files used by GEN_PD.PRG | | |
| P_CODES.DBF | Database | General printer escape code database |
| P_CODES.CDX | Index File | Index for printer escape code database |
| FONTS.DBF | Database | Database with Postscript font information |
| FONTS.CDX | Index | Structural index for the FONTS database |
| Printer Drivers included in GENPD.PJX | | |
| DRIVER | Program | FoxPro language general printer driver program |
| DRIVER2 | Library | API general printer driver |
| PS | Program | FoxPro language Postscript printer driver program |
| PSAPI | Library | API Postscript printer driver |

GENPD.APP – Printer Support Application

GENPD.APP, an application built from GENPD.PJX, is installed in the same directory with FoxPro. GENPD.APP is executed when you:

- Choose the **Printer Driver Setup** check box in the Printer Setup dialog.
- Choose the **Set Printer Driver** check box in the Report Page Layout or Label Environment dialogs.
- Issue the SET PDSETUP command.
- Store the name of a printer driver setup to _PDSETUP.
- Start FoxPro and a default printer driver setup has been specified.

The system memory variable `_GENPD` contains the name of the application or program that is executed when one of these five events occur. `GENPD.APP` is stored in `_GENPD` by default. You can store the name of a different application or program to execute in `_GENPD`, allowing you to replace the FoxPro supplied printer support application with your own. The ability to replace FoxPro's printer support with your own provides unlimited flexibility when creating printer utilities for your applications.

When executed, `GENPD.APP` displays dialogs that let you create, select, modify or delete printer setups. You can also specify a default printer setup or clear the current printer setup. When you specify a printer setup, the name of a printer driver program is stored to the system memory variable `_PDRIVER` and the setup name is stored to `_PDSETUP`. When you print reports or labels or use `DISPLAY`, `LIST` or `TYPE` to send output to a file or printer, procedures in the printer driver program are executed to specify the style of text output to your printer.

Sample Printer Drivers

Four printer drivers are supplied with FoxPro.

| Supplied Driver | Description |
|-----------------|---|
| DRIVER.PRG | FoxPro language general printer driver program |
| DRIVER2.PLB | API general printer driver |
| PS.PRG | FoxPro language Postscript printer driver program |
| PSAPI.PLB | API Postscript printer driver |

The FoxPro language printer driver `DRIVER.PRG` and the C language printer driver `DRIVER2.PLB` are used to support almost 100 different printers. The FoxPro language printer driver `PS.PRG` and the C language driver `PSAPI.PLB` support Postscript printers.

The `_PDRIVER` system memory variable contains the name of the printer driver that is used when you print reports or labels or use `DISPLAY`, `LIST` or `TYPE` to send output to a file or printer. By default, a C language printer driver (`DRIVER2.PLB` or `PSAPI.PLB`) is used.

The FoxPro language general printer drivers, `DRIVER.PRG` and `PS.PRG`, can be modified to suit your printing needs. The source code for the C language printer drivers, `DRIVER2.PLB` and `PSAPI.PLB`, is provided with the optional Microsoft® FoxPro 2.5 Library Construction Kit. In addition to demonstrating how to create a FoxPro printer driver, the C language printer drivers provide insight into how an API library can be used in a practical application.

P_CODES and FONTS Databases

Two databases are used to store printer specific information. P_CODES is based on a public domain database of printer control (escape) codes. Printer control sequences are sent to your printer by FoxPro to enable or disable different printing styles. P_CODES provides support for almost 100 different printers, and is used with the general printer drivers DRIVER.PRG and DRIVER2.PLB.

The FONTS database is used with the Postscript printer drivers PS.PRG and PSAPI.PLB. FONTS provides attribute information for specific Postscript fonts.

You can modify the contents of these databases to suit your needs. For example, let's assume that your dot matrix printer is "100%" compatible with another printer. In the Printer Setup Editing dialog you specify the printer that your printer is compatible with. Your printed output looks great, except you cannot print in italics. After consulting the manual that came with your printer, you find that the printer control code that enables italics is different than the code stored in the P_CODES database. When you change the italics printer control code in the database to the proper value, you find your printed output is now perfect.

Here is the structure of the FONTS and P_CODES databases, along with a description of the contents of each field.

| FONTS.DBF | | | |
|------------|-----------|-------|--------------------------|
| Field Name | Type | Width | Description |
| FONTNAME | Character | 20 | Font family name |
| SEPARATOR | Character | 5 | Separator for type faces |
| REGULAR | Character | 20 | Upright face name |
| BOLD | Character | 10 | Bold face name |
| ITALIC | Character | 20 | Italic face name |

| P_CODES.DBF | | | |
|-------------|-----------|-------|-----------------------------------|
| Field Name | Type | Width | Description |
| P_NAME | Character | 30 | Printer name |
| P_OUTPORT | Character | 4 | Printer output port |
| P_SETUP | Character | 60 | Printer initialization code |
| P_RESET | Character | 60 | Printer reset code |
| P_FLEN | Character | 60 | Form length code |
| P_FF | Character | 60 | Form feed code |
| P_6LPI | Character | 60 | 6 lines per vertical inch |
| P_8LPI | Character | 60 | 8 lines per vertical inch |
| P_10CPI | Character | 60 | 10 characters per horizontal inch |
| P_12CPI | Character | 60 | 12 characters per horizontal inch |
| P_COMPRESS | Character | 60 | Compressed print |
| P_LANDSCAP | Character | 60 | Landscape page orientation |
| P_PORTRAIT | Character | 60 | Portrait page orientation |
| P_BOLDON | Character | 60 | Enable bold face printing |
| P_BOLDOFF | Character | 60 | Disable bold face printing |
| P_ULINEON | Character | 60 | Enable underlined printing |
| P_ULINEOFF | Character | 60 | Disable underlined printing |
| P_ITALON | Character | 60 | Enable italics printing |
| P_ITALOFF | Character | 60 | Disable italics printing |
| P_SUPERON | Character | 60 | Enable superscript printing |
| P_SUPEROFF | Character | 60 | Disable superscript printing |
| P_SUBON | Character | 60 | Enable subscript printing |
| P_SUBOFF | Character | 60 | Disable subscript printing |
| P_FIXED | Character | 60 | Fixed font spacing |
| P_PROPTNAL | Character | 60 | Proportional font spacing |
| P_CRLF | Character | 60 | Line advance method |
| P_HORZMV1 | Character | 60 | Horizontal movement code |
| P_HORZMV2 | Character | 60 | Horizontal movement code |

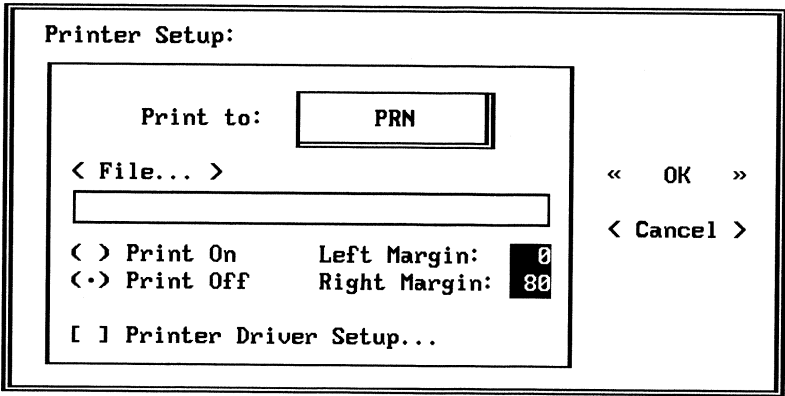
Using FoxPro's Sample Printer Drivers

GENPD.APP, the printer support application included with FoxPro, is installed automatically with FoxPro. When GENPD.APP is installed, you can interactively specify a printer driver for your printer and create different print setups. A default printer setup can be specified — the default setup is loaded when FoxPro is started.

GENPD.APP may well provide all the printer support you need. If you have special printing needs that GENPD.APP cannot provide, you can add another record to the P_CODES database, modify GENPD.APP or create your own printer driver or printer driver application. See the Creating Custom Printer Drivers section that follows for more information on modifying and creating printer drivers.

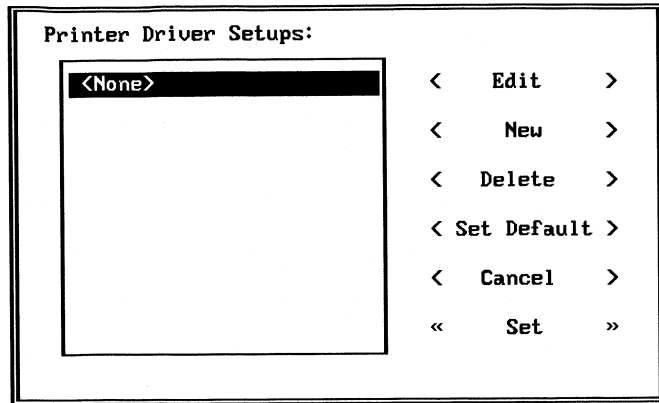
Specifying a Printer Driver

To specify a printer driver, first choose the **Printer Setup...** option from the **File** menu popup. The Printer Setup dialog appears.



Printer Setup Dialog

Now choose the **Printer Driver Setup...** check box in the Printer Setup dialog. The Printer Driver Setup dialog appears.

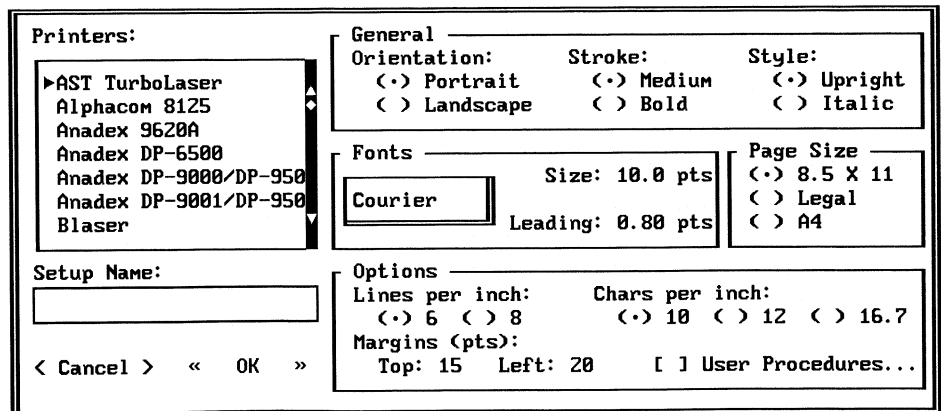


Printer Driver Setup Dialog

From this dialog, you can create a printer driver setup, modify an existing setup, delete a setup, specify a default setup, load a setup and clear the current setup.

Creating a New Printer Driver Setup

Choose the **New** push button in the Printer Driver Setup dialog to display the Printer Setup Editing dialog.



Printer Setup Editing Dialog

To create a new printer driver setup, choose a printer from the **Printers** list. Adjust any print parameters (page orientation, page size, font styles, etc.) if desired.

If you specify the Postscript or HP[®]LaserJet III printer driver, you can:

- Specify a default font (Times, Helvetica, Courier etc.) and font size in points (1 point \approx 1/72 inch) for reports and labels and output from DISPLAY, LIST and TYPE.
- Specify the leading (space between lines) in points.
- Specify the top and left margins in points for reports.

If you use a printer driver other than Postscript, you can:

- Specify the number of lines per vertical inch (6 or 8).
- Specify the number of characters per horizontal inch (10, 12 or 16.7).

Once you have specified your print parameters, save the setup. To save the setup, type a name for the setup in the Setup Name text box and choose the **OK** push button. The message "Setup Saved" is displayed. Printer driver setups are stored in your FoxPro resource file.

The printer driver setup you defined now appears in the Printer Driver Setup dialog. To load this setup for your printer, choose the **Set** push button.

Modifying an Existing Printer Setup

To modify an existing printer setup, choose the name of the printer setup to modify from the **Printer Driver Setups** list in the Printer Driver Setup dialog, then choose the **Edit** push button. The Setup Editing dialog appears.

Deleting a Printer Setup

To delete a printer setup, choose the name of the printer setup to delete from the **Printer Driver Setups** list in the Printer Driver Setup dialog, then choose the **Delete** push button.

Specifying a Default Printer Setup

A default printer setup can be automatically loaded when FoxPro is started. To specify a default printer setup, choose the name of the printer setup from the **Printer Driver Setups** list in the Printer Driver Setup dialog, then choose the **Set Default** push button. The system message "Default was set" is displayed.

You can also specify a default startup setup in your FoxPro configuration file, CONFIG.FP. Include the line

```
PDSETUP = <setup name> WITH <parm list>
```

in your CONFIG.FP file, where <setup name> is the name of the setup you want to load when FoxPro is started. *Be sure to enclose the setup name in quotation marks.* The default printer driver setup specified in your CONFIG.FP takes precedence over a default printer driver setup specified in the Printer Driver Setup dialog.

Loading a Printer Driver Setup

[] A printer driver setup can also be loaded programmatically with SET PDSETUP, or by storing the name of the setup to the _PDSETUP system memory variable. These are described in further detail in the FoxPro *Language Reference*.

Clearing the Current Printer Setup

To clear the currently loaded printer driver setup, choose the **<None>** option in **Printer Driver Setups** list in the Printer Driver Setup dialog, then choose the **Set** push button.

Specifying Printer Procedures Interactively

If GENPD.APP (the printer driver setup application supplied with FoxPro) is loaded, you can specify external programs that are executed at the bottom of the printer procedures called by FoxPro during printing. These procedures allow you to perform additional actions without having to modify the printer driver program.

The dialog box is titled "Printer Application Procedure Dialog". It is divided into two main sections: "Start" and "End". Each section contains a list of checkboxes for different printer procedures: "Document...", "Page...", "Line...", and "Object...". Each checkbox is followed by a text input field for specifying an external program. To the right of the "End" section are two buttons: "< OK >" and "< Cancel >". At the bottom of the dialog, there is a checkbox labeled "Object..." followed by a text input field.

Printer Application Procedure Dialog

When you choose the **User Procedure...** check box in the Printer Driver Setup dialog, the Printer Application Procedure dialog appears.

To specify an external program for a printer driver procedure, choose the check box for the procedure. The Open File dialog is displayed. You can choose an external program from this dialog for the printer driver procedure.

You can specify a program for these procedures:

| | | | |
|----------|-----------|----------|-----------|
| PDDOCST | PDDOCEND | PDPAGEST | PDPAGEEND |
| PDLINEST | PDLINEEND | PDOBJST | PDOBJEND |
| PDOBJECT | | | |

FoxPro passes one value to the procedure program. The value passed to the procedure program are the control codes that would normally be passed back to FoxPro. The procedure program can modify these codes and return the new codes to the driver to be passed back to FoxPro.

For example, a program specified for the document start procedure PDDOCST could append a number of page ejects codes to the codes passed from the PDDOCST routine. This would permit multiple blank pages to be ejected before a report or labels are printed.

Creating Custom Printer Drivers

FoxPro printer driver support extends beyond the sample drivers provided and includes the ability to execute procedures written in the FoxPro language, C, and assembly language through events associated with printing. Events include loading a printer driver, starting a document, starting a page, starting and ending a line or an object, and so on.

The meaning of a particular style code, the action to be taken upon starting a page, what happens at the end of a document, etc., can be any action that you can code in the FoxPro language, C, or assembly language. You can use the predefined style codes or create your own style codes which can trigger any activity that's possible to program.

A FoxPro printer driver is a FoxPro program or an API routine. Because a printer driver is a program, the appearance of your printed output is restricted only by your program and printer, not by FoxPro.

A printer driver program consists of a set of specially named procedures that return values to FoxPro. These values are typically printer control codes or escape sequences that vary from printer to printer. Control codes or escape sequences determine the format of printed output. To create a FoxPro printer driver for your printer, all you need are the control codes or escape sequences for your printer. These are usually listed in your printer manual.

Printer Driver Programs

A FoxPro printer driver is a FoxPro program containing a set of specially named procedures. When you load a printer driver, FoxPro executes the appropriate procedures in the driver program for the object being printed.

The values returned by these procedures specify the print attributes for objects, lines, pages and the entire printed output. The values returned by these procedures are usually printer control codes or escape sequences along with the object.

FoxPro passes values to some of these procedures. Procedures that are passed values from FoxPro *must* include a PARAMETERS statement to accept the values. The PARAMETERS statement must be the first program line following PROCEDURE <procedure name>.

Some examples of values passed from FoxPro to the procedures are report or label height and width, font attributes (bold, italic, etc.) and positions of objects. These values can be ignored, or can be used to determine the value returned to FoxPro by the procedure.

Printer Driver Procedures

Below are the special procedures you can include in a printer driver program. Procedures that are passed values from FoxPro are noted along with information about the values.

| Printer Driver Program Procedures | | |
|-----------------------------------|---|---|
| Procedure Name | Used By | Values Passed from FoxPro to the Procedure |
| PDONLOAD | — | None |
| PDONUNLOAD | — | None |
| PDDOCST | Reports Labels DISPLAY, LIST, TYPE | Report or Label Height and Width, or _PLENGTH and _RMARGIN |
| PDDOCEND | Reports Labels DISPLAY, LIST, TYPE | None |
| PDPAGEST | Reports TYPE | None |
| PDPAGEEND | Reports TYPE | None |
| PDLINEST | Reports Labels DISPLAY, LIST, TYPE | None |
| PDLINEEND | Reports Labels DISPLAY, LIST, TYPE | None |
| PDOBJST | Reports Labels ? and ?? DISPLAY, LIST, TYPE | Style Code String |
| PDOBJECT | Reports Labels ? and ?? DISPLAY, LIST, TYPE | Object, Style Code String |
| PDOBJEND | Reports Labels ? and ?? DISPLAY, LIST, TYPE | Style Code String |
| PDADVPR | Reports Labels ? and ?? DISPLAY, LIST, TYPE | Prior Object Ending Column, Beginning Column of Next Object |

PDONLOAD

The PDONLOAD procedure is immediately executed when a printer driver program is specified with `_PDRIVER`. If you specify a printer driver in your FoxPro `CONFIG.FP` configuration file, this procedure in the `_GENPD` program is executed immediately after FoxPro is loaded.

This procedure can be used to display a dialog to select print options.

Any value returned to FoxPro by PDONLOAD is ignored.

Values passed to PDONLOAD from FoxPro

None.

PDONUNLOAD

This procedure is executed when another printer driver program is loaded with `_PDRIVER` or the current driver is unloaded.

This procedure is typically used to release the special `_PDPARMS` memory variable array from memory. `_PDPARMS` is discussed in greater detail in a following section.

Any value returned to FoxPro by PDONUNLOAD is ignored.

Values passed to PDONUNLOAD from FoxPro

None.

PDDOCST

PDDOCST is the document start procedure. The value returned to FoxPro from PDDOCST is sent to the printer *before* a report or a set of labels is printed, or output from `DISPLAY`, `LIST` or `TYPE` is sent to the printer or a file.

The value returned to FoxPro from PDDOCST typically contains printer control codes or escape sequences that affect the entire document. With PDDOCST you can specify:

- The number of rows and columns in the printed output
- The print quality (draft or letter quality)
- The printed output's orientation (portrait or landscape)
- A font style and/or size for the entire printed output

Values passed to PDDOCST from FoxPro

Numeric Value 1 For reports, the first numeric value passed to PDDOCST from FoxPro is the report height specified in the Page Layout dialog in the Report Writer.

For labels and DISPLAY, LIST and TYPE, the first numeric value passed to PDDOCST from FoxPro is the current value of `_PLENGTH`.

Numeric Value 2 For reports, the second numeric value passed to PDDOCST from FoxPro is the report width specified in the Page Layout dialog in the Report Writer.

For labels, the second numeric value passed to PDDOCST from FoxPro is the total width of the labels plus the space separating them. For example, if you are printing three across labels that are each 20 columns wide, and they are separated by five columns, the total width is 70 ($20 + 5 + 20 + 5 + 20$). In this case the value passed to PDDOCST is 70.

For DISPLAY, LIST and TYPE, the second numeric value passed to PDDOCST is the value contained in `_RMARGIN`.

PDDOCEND

PDDOCEND is the document end procedure. The value returned to FoxPro from PDDOCEND is sent to the printer *after* a report or a set of labels is printed, or after output from DISPLAY, LIST and TYPE is sent to the printer or a file.

The value returned to FoxPro from PDDOCEND typically restores the prior printer settings changed by PDDOCST.

Values passed to PDDOCEND from FoxPro

None.

PDPAGEST

PDPAGEST is the page start procedure. The value returned to FoxPro from PDPAGEST is sent to the printer before each new page in a report is printed. PDPAGEST is not called when you are printing labels, or when output from DISPLAY and LIST is sent to the printer or a file.

Values passed to PDPAGEST from FoxPro

None.

PDPAGEEND

PDPAGEEND is the page end procedure. The value returned to FoxPro from PDPAGEEND is sent to the printer after every page is printed. PDPAGEEND is not called when you are printing labels, or when output from DISPLAY and LIST is sent to the printer or a file.

Values passed to PDPAGEEND from FoxPro

None.

PDLINEST

PDLINEST is the line start procedure. The value returned to FoxPro from PDLINEST is sent to the printer before every report line is printed, and before every line output by DISPLAY, LIST and TYPE. If you are printing labels, the return value from PDLINEST is sent to the printer before each horizontal line in a label or set of labels (for example, three across labels) is printed.

Values passed to PDLINEST from FoxPro

None.

PDLINEEND

PDLINEEND is the line end procedure. The value returned to FoxPro from PDLINEEND is sent to the printer after every report line is printed, and after every line output by DISPLAY, LIST and TYPE. When printing labels, the return value from PDLINEST is sent to the printer after each horizontal line in a label or set of labels is printed.

Values passed to PDLINEEND from FoxPro

None.

PDOBJST

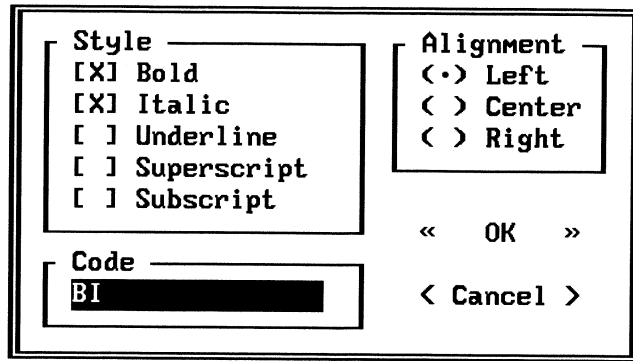
PDOBJST is the object start procedure. The value returned to FoxPro from PDOBJST is sent to the printer once before each object is printed. Objects are text, fields and boxes created in the FoxPro Report Writer and Label Designer, or the expression in ? or ??. Each line output by DISPLAY, LIST and TYPE is an object.

Because the PDOBJST return value is sent to the printer before text, a field, a box or an expression is printed, the value returned to FoxPro is usually a control code or escape sequence that enables print attributes. For example, PDOBJST can send a control code to the printer to change the font style to enable bold face printing for field names.

Values passed to PDOBJST from FoxPro

Character String The character string passed to the PDOBJST procedure from FoxPro contains the style codes specified for an object in the Style dialog in the FoxPro Report Writer or Label Designer, or in a STYLE clause included with ? or ??.

For example, in the Report Writer you've chosen the **Bold** and **Italic** check boxes for an object in the Style dialog. The letters BI appear in the text box — BI is returned in a character string to PDOBJST for the object.



Style Dialog



You are not restricted to the default FoxPro bold, italic, underline, superscript and subscript print attributes offered in the Style dialog in the Report Writer or Label Designer. You can enter additional characters and numbers in the text box in the Style dialog. Your PDOBJST procedure can parse the characters and numbers and send the appropriate control codes to the printer to enable other printer attributes.

The null string is passed to PDOBJST for DISPLAY, LIST and TYPE.

PDOBJECT

PDOBJECT is the object procedure. Objects are text, fields and boxes created in the FoxPro Report Writer and Label Designer, or the expression in ? or ??. Each line output by DISPLAY, LIST and TYPE is an object.

FoxPro sends the object and the object's style codes to this procedure. Because the object is sent to this procedure, you can conditionally return printer control codes or escape sequences for an object, or modify the object.

For example, you can print negative values (losses) surrounded by parentheses. Your PDOBJECT procedure can test the object — if the object is numeric and less than zero, the object can be padded with parentheses and returned to FoxPro.

Values passed to PDOBJECT from FoxPro

Character String 1 The object being printed. Objects are text, fields and boxes created in the FoxPro Report Writer and Label Designer, or the expression in ? or ??. Each line output by DISPLAY, LIST and TYPE is an object.

Character String 2 The style codes specified for an object in the Style dialog in the FoxPro Report Writer or Label Designer, or in a STYLE clause included with ? or ??. The null string is passed for DISPLAY, LIST and TYPE.

PDOBJEND

PDOBJEND is the object end procedure. The value returned to PDOBJEND from FoxPro is sent to the printer once after each object is printed. Objects are text, fields and boxes created in the FoxPro Report Writer and Label Designer, or the expression in ? or ??.

Because the PDOBJEND return value is sent to the printer after text, a field, a box or an expression is printed, the value returned to FoxPro by PDOBJEND is usually a control code or escape sequence that disables a print attribute enabled by PDOBJST.

Values passed to PDOBJEND from FoxPro

Character String The character string sent to PDOBJEND from FoxPro contains the style codes specified for an object in the Style dialog in the FoxPro Report Writer or Label Designer, or in a STYLE clause included with ? or ??. The null string is passed to PDOBJEND for DISPLAY, LIST and TYPE.

PDADVPRT

PDADVPRT is the object advance printer procedure. PDADVPRT determines how the printer advances to the next object to print. The method that printers use to advance varies between printers. For example, some printers advance horizontally by outputting spaces while others require a series of control characters.

Values passed to PDADVPRT from FoxPro

| | |
|------------------------|--|
| Numeric Value 1 | The first numeric value sent to PDADVPRT from FoxPro is the ending column position of the object just printed. |
| Numeric Value 2 | The second numeric value sent to PDADVPRT from FoxPro is the beginning column position of the next object to be printed. |

Printer Procedures Notes

When you create a printer driver program, keep the following in mind:

- A printer driver program does not have to contain all or even *any* of these specially named procedures. If any of these procedures are included in a printer driver program, they do not have to return a value to FoxPro. In these cases no output is generated from the procedures.
- The procedure values returned to FoxPro can contain embedded nulls. Many printer control codes and escape sequences use null characters.
- When printing an expression with ? or ??, FoxPro only calls PDLINEEND, PDOBJST, PDOBJECT, PDOBJEND and PDADVPRT.
- When printing labels, FoxPro does not call PDPAGEST and PDPAGEEND.
- When you DISPLAY MEMORY to a printer and _PDPARMS exists, _PDPARMS will not be printed. _PDPARMS is not displayed on the screen or a window with DISPLAY MEMORY when PRINT is SET ON or you DISPLAY MEMORY TO PRINT.

PDPARMS

A special public memory variable array, `_PDPARMS`, can be created to contain printer control codes and escape sequences. `_PDPARMS` is automatically created when you include the `WITH` clause in `SET PDSETUP`. You can also create `_PDPARMS` with `DIMENSION`, `DECLARE` and `PUBLIC`.

`_PDPARMS`, unlike other arrays, cannot be cleared from memory with `CLEAR MEMORY`, `CLEAR ALL` or `RELEASE MEMORY`. To remove `_PDPARMS` from memory you must use

```
RELEASE _PDPARMS
```

If you create `_PDPARMS` in your main program, any control codes and escape sequences stored in `_PDPARMS` are available to all programs in your application. Because `_PDPARMS` cannot be cleared from memory with the usual memory commands, `_PDPARMS` does not require special consideration when clearing other memory variables or arrays.



`_PDPARMS` is especially convenient for providing printer control codes and escape sequences in printer driver programs. If printer control codes and escape sequences are stored in `_PDPARMS`, they can be returned to FoxPro by the procedures in a printer driver program.

If your application supports multiple printers, printer control codes and escape sequences can be stored in fields in a read-only printer driver database. If your application is created from a project, you can include the printer driver database in the project. When a printer is selected, the fields from the appropriate record in the printer driver database can be `SCATTERed` to `_PDPARMS`.

Refer to the sample printer driver program later in this chapter to see how a printer driver database and `_PDPARMS` can work together. The `GEN_PD.PRG` printer driver interface program provided with FoxPro also demonstrates how to create and manipulate the `_PDPARMS` array.

SET PDSETUP

If you include the WITH <expression list> clause in SET PDSETUP, _PDPARMS is automatically created. The expressions included in the WITH clause become the elements in _PDPARMS.

For example, the following command creates _PDPARMS.

```
SET PDSETUP TO <printer driver setup> WITH 'A', 'B', 1
```

_PDPARMS is a one-dimensional array with three elements. The first element contains A, the second element contains B and the third element contains 1. (If you are using GENPD.APP, the default printer setup interface application, do not include the WITH clause. GENPD.APP fills _PDPARMS automatically.)

If _PDPARMS already exists and you issue SET PDSETUP with a WITH <expression list>, _PDPARMS is automatically redimensioned. Once _PDPARMS exists, you can redimension it with DIMENSION or DECLARE.

Designating a Printer Driver Program

You can designate a printer driver program when FoxPro is started, or by storing the name of the printer driver program to the _PDRIVER system memory variable.

_PDRIVER

You can specify a printer driver program by storing the name of the program to the _PDRIVER system memory variable.

When you store the name of a printer driver program to _PDRIVER:

- The PDONUNLOAD procedure in the currently loaded printer driver program is executed (if it has a PDONUNLOAD procedure).
- The printer driver you stored to _PDRIVER is then loaded and its PDONLOAD procedure is executed (if it has a PDONLOAD procedure).

When you store a printer driver program to _PDRIVER, the system message "Printer driver installed" is displayed. This message can be suppressed by issuing the following command before you store a printer driver program name to _PDRIVER:

```
SET NOTIFY OFF
```

Specifying a Printer Driver in CONFIG.FP

You can specify a printer driver setup in FoxPro's CONFIG.FP configuration file. When FoxPro is started, the PDONLOAD procedure in the printer driver program designated in the printer driver setup is executed.

To specify a printer driver setup in your CONFIG.FP configuration file, include the line

```
PDSETUP = <setup name> WITH <parm list>
```

If you specify a printer driver setup in your CONFIG.FP file, the PDONLOAD procedure (if one is included in the designated printer driver program) is executed immediately after FoxPro is loaded.

If you execute another program from your CONFIG.FP with the line

```
COMMAND = DO <program name>
```

the printer driver program is executed *after* the PDONLOAD procedure is executed.

For additional information on FoxPro's CONFIG.FP file, see the Customizing FoxPro chapter in the FoxPro *Installation and Configuration* manual.

Sample Printer Driver Programs

Four printer driver programs are supplied with FoxPro. FoxPro language and C versions of a general printer driver and a Postscript driver are provided. The C versions of these programs are used by default by FoxPro.

This sample printer driver program demonstrates how to use the special procedure names and how to pass values to and receive values from FoxPro. Also demonstrated is the use of the special `_PDPARMS` array. `_PDPARMS` is used to store printer control codes and escape sequences scattered from the `P_CODES` database.

Here is the sample FoxPro language printer driver program, `DRIVER.PRG`. The program included with your version of FoxPro may differ from this.

```
PROCEDURE PDunload
    RELEASE _pdparms
RETURN
```

Executed when a printer driver is specified with `_PDRIVER`. The `_PDPARMS` array is cleared from memory.

```
PROCEDURE PDdocst
PARAMETER doc_height, doc_width
```

The value returned by this procedure is sent to the printer before a report or labels are printed. You can initialize your printer in this procedure.

```

*
* Build the control characters for the initialization of the printer.
* These are stored in specific positions of the array _pdparms.
* Also, for reports, store the page height and width.
*

ctlchars = _pdparms[3] + _pdparms[10] + ;
           _pdparms[21] + _pdparms[7] + _pdparms[8] + _pdparms[25] + ;
           _pdparms[26]
_pdparms[28] = doc_height
_pdparms[29] = doc_width

IF NOT EMPTY(_pdparms[30])
    DO (LOCFILE(_pdparms[30], "PRG;APP;SPR;FXP;SPX", ;
               "Where is " + _pdparms[30] + "?")) WITH ctlchars
ENDIF

RETURN ctlchars
```

This procedure determines how the printer advances from object to object.

```

PROCEDURE PDadvprt
PARAMETER fromhere, wheretogo

*
* Check to see if the current printer has control codes for horizontal
* movement. If so, then return the movement factor otherwise, use a
* default movement scheme.
*

IF EMPTY(_pdparms[23])
    IF fromhere>wheretogo
        RETURN CHR(13) + SPACE(wheretogo)
    ELSE
        RETURN SPACE(wheretogo-fromhere)
    ENDIF
ELSE
    IF fromhere = wheretogo
        RETURN ""
    ELSE
        RETURN _pdparms[23] + ALLTRIM(STR(wheretogo)) + _pdparms[24]
    ENDIF
ENDIF
RETURN
    
```

The value returned by this procedure is sent to the printer before the start of each new page in a report. It is not executed when printing labels.

```

PROCEDURE PDpagest

*
* Reset the line counter and return the codes to go to the next line.
*

_pdparms[27] = 1

ctlchars = IIF(EMPTY(_pdparms[6]), CHR(12)+CHR(13), _pdparms[6])

IF NOT EMPTY(_pdparms[31])
    DO (LOCFILE(_pdparms[31], "PRG;APP;SPR;FXP;SPX", ;
        "Where is " + _pdparms[31] + "?")) WITH ctlchars
ENDIF

RETURN ctlchars
    
```

```
PROCEDURE PObject
PARAMETERS textline, attribs
```

```
startchars = ""
endchars = ""
ctlchars = ""
```

```
IF LEN(attribs) > 0 && check to see if the object has any attributes
  STORE .F. TO m.bold, m.italic, m.super, m.sub, m.under
  attribs = UPPER(attribs)
  FOR i = 1 TO LEN(attribs) && parse the attribute string
```

```
    char = SUBSTR(attribs,i,1)
    * Build the control characters for the current object's
    * attributes. Build both the turning on and off of the
    * style.
    *
    DO CASE
      CASE char = "B" AND NOT m.bold
        IF NOT EMPTY(_pdparms[26])
          startchars = startchars + _pdparms[11]
          endchars = endchars + _pdparms[12]
        ENDIF
        m.bold = .T.

      CASE char = "I" AND NOT m.italic
        IF NOT EMPTY(_pdparms[25])
          startchars = startchars + _pdparms[15]
          endchars = endchars + _pdparms[16]
        ENDIF
        m.italic = .T.

      CASE char = "R" AND NOT m.super AND NOT m.sub
        startchars = startchars + _pdparms[17]
        endchars = endchars + _pdparms[8] + _pdparms[18]
        m.super = .T.

      CASE char = "L" AND NOT m.sub AND NOT m.super
        startchars = startchars + _pdparms[19]
        endchars = endchars + _pdparms[8] + _pdparms[20]
        m.sub = .T.

      CASE char = "U" AND NOT m.under
        startchars = startchars + _pdparms[13]
        endchars = endchars + _pdparms[14]
        m.under = .T.

    ENDCASE
```

```
  ENDFOR
ENDIF
```

This procedure is passed the object being printed and the style code (if any) for the object. Based on the style code passed, the object is returned surrounded by the appropriate escape characters to enable and disable the specified print style (bold, italic, etc.).

```

ctlchars = startchars + textline + endchars
IF NOT EMPTY(_pdparms[34])
    DO (LOCFILE(_pdparms[34], "PRG;APP;SPR;FXP;SPX", ;
        "Where is " + _pdparms[34] + "?")) WITH ctlchars
ENDIF

RETURN ctlchars

```

The value returned by this procedure is sent to the printer after each line in a report or labels is printed.

```

PROCEDURE PDlineend

```

```

*
* Check to see if the line number is greater than the page length.
* If so, don't return anything and let the page start handle the
* movement.
*
IF _pdparms[27] >= _pdparms[28]
    ctlchars = ""
    _pdparms[27] = 1
ELSE
    ctlchars = IIF(EMPTY(_pdparms[22]), CHR(13)+CHR(10), _pdparms[22])
    _pdparms[27] = _pdparms[27] + 1
ENDIF

IF NOT EMPTY(_pdparms[36])
    DO (LOCFILE(_pdparms[36], "PRG;APP;SPR;FXP;SPX", ;
        "Where is " + _pdparms[36] + "?")) WITH ctlchars
ENDIF

RETURN ctlchars

```

This procedure is executed after the report or labels are printed. It returns characters that reset the printer.

```

PROCEDURE PDdocend

```

```

ctlchars = IIF(EMPTY(_pdparms[3]), "", _pdparms[3])

IF NOT EMPTY(_pdparms[38])
    DO (LOCFILE(_pdparms[38], "PRG;APP;SPR;FXP;SPX", ;
        "Where is " + _pdparms[38] + "?")) WITH ctlchars
ENDIF

RETURN ctlchars

```

Creating Custom Printer Driver Setup Applications

FoxPro version 2.5 includes a printer driver interface application, GENPD.APP, that is executed when you choose the **Printer Driver Setup...** check box in the Printer Setup dialog. GENPD.APP lets you specify a printer driver and printing information. You can create your own custom printer driver interface application or modify GENPD.APP for your use.

The _GENPD system memory variable is used to specify the printer driver interface application. The application whose name is stored in _GENPD is executed when any of these five events occur:

- The **Printer Driver Setup** check box in the Printer Setup dialog is chosen
- The **Set Printer Driver** check box in the Report or Label dialogs is chosen
- SET PDSETUP is issued
- A printer setup name is stored in _PDSETUP
- A default printer setup has been specified and FoxPro is started

To execute your printer driver interface application when one of these events occur, store your application name in _GENPD.

See the _GENPD system memory variable in the FoxPro *Language Reference* for additional information on creating your own printer driver interface applications.

Appendix

A Error Messages

FoxPro is an intuitive, easy-to-use program. There will be times when you'll press the wrong button or key or even give FoxPro a command it doesn't understand. When something like this happens, FoxPro alerts you by displaying an error message.

This appendix lists FoxPro error messages in alphabetical order. The corresponding error number is in parentheses following the text of the error message. At the end of this appendix is a table with the error messages listed in numerical order.



Error messages and their meanings are also located in FoxPro's online help under the topic ► Error Messages.

Error Message Parameters

Certain error messages return the file, field or variable name in as a parameter in the error message. With the function SYS(2018), you can return the error message parameter for the following error messages:

- "<cursor>" must be created with SELECT ... INTO TABLE. (1815)
- "<field>" is not related to the current work area. (1165)
- "<field> | <variable>" is not unique and must be qualified. (1832)
- "<file>" is not a FoxPro EXE file. (1196)
- "<file>" is not an object file. (1309)
- "<name>" is not a file variable. (226)
- "<name>" is not a file variable. (226)
- "<name>" is not a memory variable. (225)
- "<name>" is not an array. (232)
- ALIAS ["<alias>"] not found. (13)
- Application file "<file>" not closed. (1178)
- Cannot create file "<file>". (102)
- Cannot open file "<file>". (1101)

- File [“<file>”] does not exist. (1)
- Key label [“<label>”] is invalid. (1255)
- Not enough disk space for “<file name>”. (1160)
- Object file “<file>” is the wrong version. (1195)
- Procedure “<procedure>” not found. (1162)
- Queue “<queue>” not found. (1716)
- Server “<server>” not found. (1715)
- SQL column “<field> | <variable>” not found. (1806)
- Variable [“<variable>”] not found. (12)
- WINDOW [“<window name>”] has not been defined. (214)



Error messages that have optional error message parameters may not always return a parameter through SYS(2018).

Alphabetical Listing of Error Messages

“<cursor>” must be created with SELECT ... INTO TABLE. (1815)

Some cursors cannot be used in successive queries.

“<field>” is not related to the current work area. (1165)

The field you have specified is not in the database in the current work area.

“<field> | <variable>” is not unique and must be qualified. (1832)

The field or variable in the SQL SELECT command you have specified cannot be found.

“<file>” is not a FoxPro EXE file. (1196)

The EXE file specified was not created by FoxPro.

“<file>” is not an object file. (1309)

An attempt was made to load a compiled FoxPro program which does not have a proper header.

“<name>” is not a file variable. (226)

A memory variable or an array variable was used where a file variable (field) is required.

“<name>” is not a memory variable. (225)

A file variable (field) was used where a memory or array variable was required.

“<name>” is not an array. (232)

A DIMENSION statement contains a variable declaration without the required subscript arguments.

**** or ^ domain error. (78)**

Exponentiation was attempted on a negative number.

bad date

The date expression you used is invalid.

ACOS() : Out of Range. (293)

The expression you have used with the arc cosine function has a value that is out of the range of -1.0 to +1.0.

ALIAS name already in use. (24)

An attempt was made to USE a database with an ALIAS that is associated with another database already in USE.

ALIAS [“<alias>”] not found. (13)

An attempt was made to either SELECT a work area not in the range A through J, or 11 through 25, or to specify an ALIAS that has not been defined.

All work areas in use. (1721)

A database is open in all 25 work areas.

API library revision mismatch. Rebuild library. (1711)

The API library you are using does not match your version of FoxPro.

Application file “<file>” not closed. (1178)

An attempt has been made to build an application while a file in the application is not closed.

Area cannot contain handle. (1011)

If this error occurs, call Microsoft Product Support. You are returned to MS-DOS after FoxPro closes all open files.

Area size exceeded during compaction. (1010)

If this error occurs, call Microsoft Product Support. You are returned to MS-DOS after FoxPro closes all open files.

ASIN() : Out of Range. (291)

The expression you have used with the arc sine function has a value that is out of the range of -1.0 to +1.0.

Attempt to move file to different device. (1153)

A file may not be renamed to a different device.

Attempt to use FoxPro function as an array. (1652)

FoxPro encountered a function when it was expecting an array name. To correct this, replace the function with a valid array name.

Bad array dimensions. (230)

Either an illegal value (such as zero) has been entered during the declaration of an array, or an array has been declared that exceeds 3,600 elements.

Bad drive specifier. (1907)

An invalid drive name was specified in a DIR command.

Bad .LIB file. (1191)

One of the files in the Microsoft FoxPro 2.5 Distribution Kit has become corrupted. Try reinstalling.

Bad width or decimal place argument. (1908)

Either the length or decimal argument was invalid in the STR() function.

Bar position must be a positive number. (167)

An attempt has been made to define bars of a menu or a popup using a negative number.

Beginning of file encountered. (38)

The record pointer was positioned before the first record in the file.

Beyond string. (62)

An attempt was made to access characters beyond the last byte of a string memory variable or database field.

Browse database closed. (1163)

A Browse window's database was closed by a Browse validation routine.

Browse structure changed. (1164)

The structure of a Browse window was changed by a Browse validation routine.

Can't find ESO file.

Standard Runtime version only.

Can't find OVL file.

Standard version only.

CANCEL/SUSPEND is not allowed. (1651)

The dBASE IV report or label file you are running calls upon a SUSPEND.

Cannot access selected database. (1152)

An attempt was made to select a database outside the range 1 to 25, or a reference was made to a file variable in a database that is not open.

Cannot allocate screen map.

There was not enough memory to run FoxPro.

Cannot append from password protected file. (1672)

The file you are attempting to append is encrypted.

Cannot build APP/EXE while suspended. (1719)

A program specified in the project is suspended. Cancel execution of the program to be able to build APP/EXE.

Cannot build APP/EXE without a main program. (1689)

The main program in the specified project cannot be found.

Cannot clear menu in use. (176)

An attempt was made to clear an active menu with the CLEAR MENU or RELEASE MENU command.

Cannot clear popup in use. (177)

An attempt was made to clear an active popup menu with the CLEAR POPUP or RELEASE POPUP command.

Cannot convert Memo file for a read-only database file. (1659)

This message appears when you try to perform a read-only operation on a dBASE IV style memo because FoxPro cannot convert a memo in a read-only operation.

Cannot create file. (1102)

The operating system has returned an error to FoxPro indicating that the new file cannot be created. The inability to create a new file is usually the result of a full disk or directory, entry of an invalid file name, or not having the proper requirements needed to access the directory which is to contain the file.

Cannot create file "<file>". (102)

The operating system has returned an error to FoxPro indicating that the new file cannot be created. The inability to create a new file is usually the result of a full disk or directory, entry of an invalid file name, or not having the proper requirements needed to access the directory which is to contain the file.

Cannot create program workspace.

FoxPro was not able to create enough workspace upon startup. This may be due to insufficient rights to the directory or not enough disk space.

Cannot find screen/menu generation program. (1693)

The GENSCRN or GENMENU programs that create menus and screens cannot be located.

Cannot GROUP by aggregate field. (1846)

There has been an attempt to GROUP by one of the aggregate functions MIN(), MAX(), SUM(), AVG(), COUNT(), NPV(), STD() or VAR().

Cannot import from password protected file. (1671)

The file you are attempting to format is encrypted.

Cannot load PROAPI16.EXE. (1709)

The PROAPI16.EXE file must be located in your FoxPro home directory if you are using API routines with the Extended version of FoxPro.

Cannot locate COMSPEC environment variable. (1412)

An environment variable needed by FoxPro cannot be found. This error usually occurs when you try to run FoxPro from within another program and the other program doesn't properly pass MS-DOS environment variables.

Cannot open configuration file.

FoxPro found a CONFIG.FP file but did not have the proper rights to open it.

Cannot open file "<file>". (1101)

The operating system has returned an error to FoxPro indicating that the file cannot be opened. The inability to open a file is usually the result of attempting to open a file which does not exist, entry of an invalid file name, or not having the proper permission needed to access the file.

Cannot redefine menu in use. (174)

An attempt was made to issue a DEFINE MENU command while there was an active menu in memory. You must first issue a DEACTIVATE MENU command.

Cannot redefine popup in use. (175)

An attempt was made to issue a DEFINE POPUP command while there was an active popup in memory. You must first issue a DEACTIVATE POPUP command.

Cannot rename current directory. (1253)

The directory you have flagged in the Filer could not be renamed.

Cannot run on MS-DOS before version 3.0.

Loaders and compact EXE files cannot run under MS-DOS version 3.0 or earlier.

Cannot SET FORMAT while in a READ with a FORMAT file. (1720)

The format file you have specified calls upon a format file.

Cannot update file. (1157)

This error is very unusual. It appears only if a critical problem occurs when writing to disk, such as space exhausted, total disk failure, etc.

Cannot write to a read-only file. (111)

An attempt was made to write to a file that was created or accessed for read-only purposes.

COLORSET resource not found. (1642)

An attempt has been made to SET COLOR SET TO a color set not found in the resource file.

Column number must be between 0 and 255. (223)

The printer column number specified is off of the page.

Column number must be between 0 and right margin. (1657)

There was an attempt to set _PCOLNO to an invalid number.

COLUMN/FORM/ALIAS/NOOVERWRITE/WIDTH allowed only with FROM clause. (1695)

These options are only available when you create a quick report with the FROM clause.

COLUMN/ROW/ALIAS/NOOVERWRITE/SIZE/SCREEN allowed only with FROM clause. (1698)

These options are only available when you create a quick screen with the FROM clause.

Compiled code for this line too long. (1252)

The object code produced for this statement exceeded the size of the FoxPro internal code buffer. This line contains too many long expressions and must be subdivided into multiple statements.

CONTINUE without LOCATE. (42)

An attempt was made to execute a CONTINUE command without first executing a LOCATE command.

CPU exhibits 32 bit multiply problem.

You are using an early version of INTEL[®] 386. Please check with your hardware vendor before continuing.

Cyclic relation. (44)

A circular relationship was attempted by defining a sequence of relations that eventually points back to a database that is already related.

Data type mismatch. (9)

An expression was considered to be invalid by FoxPro because it is composed of data types that cannot be evaluated together. Either the appropriate data type required by a FoxPro statement was not used, or the two fields specified in an operation between two databases did not have the same data type (e.g., UPDATE, REPLACE, SET RELATION).

Database is not ordered. (26)

One of the following has occurred. An index for the database file was not selected when an UPDATE command, using the RANDOM option, was encountered. A FIND/SEEK was attempted against an unindexed database or while SET ORDER TO 0 was in effect. An attempt was made to SET RELATION with a non-numeric relational expression to an unindexed database or while SET ORDER TO 0 was in effect.

Database operation invalid while indexing. (1690)

There has been an attempt to do a database operation inside of an index UDF.

Descending not permitted on IDX files. (1706)

IDX index files do not support a DESCENDING clause.

Display mode not available. (216)

An attempt was made to select an unavailable display mode.

Division by 0. (1307)

Zero was specified as the second argument in the MOD function.

DO nesting too deep. (103)

The maximum DO program nesting levels of 32 has been exceeded.

Duplicate field names. (1156)

In the FIELDS option for SORT and COPY, you have specified a duplicate name in the list of fields. Correct your program or command and try again.

End of file encountered. (4)

The record pointer was positioned past the last record in the file.

Endtext without text. (1214)

An ENDTEXT statement is missing a corresponding TEXT statement.

Error in label field definition. (1245)

An invalid expression was encountered in a LABEL file (.LBX).

ESL and ESO file do not match.

The time and date of your ESL and ESO files do not match. Reinstall. Standard Runtime version only.

Exclusive open of file is required. (110)

An attempt was made to perform an operation which requires exclusive use of the database.

EXE and OVL file do not match.

The time and date of your EXE and OVL files do not match. Reinstall. Standard version only.

Expression evaluator fault. (67)

This is an internal consistency check failure in the FoxPro expression evaluator. This may be caused by a damaged FoxPro object code file. Recompile the program that caused the error.

Fatal error <expN> reporting error <expN>.

If this error occurs, call Microsoft Product Support. No error number is returned and you are returned to MS-DOS after FoxPro closes all open files.

Feature not available. (1001)

The FoxPro feature which you have attempted to use is not supported under the version of FoxPro being used.

Field must be a Memo field. (350)

An attempt has been made to enter a field name with the APPEND/COPY MEMO command that is not a memo field type.

'Field' phrase not found. (1130)

The field specified in PROMPT <field> was not found.

File access denied. (1705)

An attempt was made to write to a file which is write protected by the MS-DOS ATTRIB command.

File already exists. (7)

The file name to which you are attempting to rename a file already exists. This error is usually generated by the RENAME command.

File close error. (1112)

An error was returned by the operating system while FoxPro was attempting to close a file.

File ["<file>"] does not exist. (1)

The file you have specified does not exist.

File is in use by another. (108)

An attempt was made to USE, DELETE or RENAME a file which is being used by another user in a multi-user system.

File is in use. (3)

An attempt was made to USE, DELETE or RENAME a file which is currently open.

File is open in another work area. (1708)

Commands that require exclusive use of a database (PACK, MODIFY STRUCTURE, ZAP, etc.) cannot be performed on a database that has been opened in multiple work areas with the USE AGAIN command.

File is read only. (1718)

An attempt was made to write to a read only file.

File must be opened exclusively to convert Memo file. (1637)

The database file you are using must be opened exclusively to be able to convert the memo file.

File not open. (1113)

An attempt was made to read from or write to a file which is not currently open.

File read error. (1104)

An error was returned by the operating system while FoxPro was attempting to read a file.

File was not LOADed. (91)

An error occurred when attempting to CALL or RELEASE a module which was not loaded.

File write error. (1105)

An error was returned by the operating system while FoxPro was attempting to write to a file. Most often, this error is the result of an attempt to write to a write-protected diskette.

FILTER expression too long. (1140)

An attempt was made to SET FILTER TO an expression that is longer than the maximum allowable size of 160 characters.

FILTER requires a logical expression. (37)

An attempt was made to SET FILTER TO an expression that is not a logical expression.

For/while need logical expressions. (1127)

A FOR or WHILE clause within a command does not contain a logical expression.

Foreign node found during compaction. (1008)

If this error occurs, call Microsoft Product Support. You are returned to MS-DOS after FoxPro closes all open files.

FOXUSER file is invalid. (1294)

Your FOXUSER file has been corrupted. Reinstall the FOXUSER file.

Free handle not found. (1003)

If this error occurs, call Microsoft Product Support. You are returned to MS-DOS after FoxPro closes all open files.

Function not implemented. (1999)

You have attempted to call a function that is not supported in the current version of FoxPro.

If/else/endif mismatch. (1211)

An ELSE or ENDIF statement does not have a corresponding IF statement.

Illegal operation for MEMO field. (34)

An INDEX command may not specify a MEMO field as its key expression.

Illegal printer driver recursion. (1910)

Your printer driver program calls upon the printer driver.

Illegal seek offset. (1103)

If this error occurs, call Microsoft Product Support.

Illegal value. (46)

An expression specified by SET MEMOWIDTH, BROWSE, or an @ ... TO command evaluated to an illegal value.

Import only Worksheet A for Lotus 1-2-3 version 3.0 files. (1679)

Worksheets B and C cannot be imported into FoxPro because a database is two-dimensional.

Improper data type in field expression. (1647)

A picture data type was encountered by REPORT for a field expression.

Improper data type in group expression. (1241)

A picture data type was encountered by REPORT for the group expression.

Incorrect handle found during compaction. (1009)

If this error occurs, call Microsoft Product Support. You are returned to MS-DOS after FoxPro closes all open files.

Index does not match database file. Recreate Index. (114)

A damaged structure within an index file has been detected.

Index expression is too big. (23)

The index expression for this index exceeds the maximum length of 220 bytes.

Index file does not match database. (19)

The index expression for the current index uses variables which are not contained within the current database.

Index tag not found. (1683)

The tag you have specified could not be found in the index.

Index tag or file name must be specified.

You have attempted to exit the Index dialog without specifying a tag name.

Insufficient memory. (1282)

There was not enough memory for FoxPro to complete an operation.

Insufficient stack space. (1308)

One of the following situations can cause this error message:

- Your program is too recursive.
- Your program is too complex and is nested too deeply.
- An error in FoxPro.

Internal consistency error.

An internal FoxPro table has been corrupted. If this error occurs, call Microsoft Product Support. No error number is returned and you are returned to MS-DOS after FoxPro closes all open files.

Internal error: Too many characters in report. (1243)

The total size of all the headings and expressions defined for the report in CREATE/MODIFY REPORT is too large to be contained in the standard report file.

Internal .LIB undefined symbol error. (1192)

The library file is corrupted or the library version does not match the FoxPro version.

Invalid buffdirty call. (1155)

If this error occurs, call Microsoft Product Support. No error number is returned and you are returned to MS-DOS after FoxPro closes all open files.

Invalid buffpoint call. (1154)

If this error occurs, call Microsoft Product Support. No error number is returned and you are returned to MS-DOS after FoxPro closes all open files.

Invalid box dimensions. (227)

Row 2, column 2 must be larger than row 1, column 1 when using the @ ... BOX command.

Invalid character in command. (1220)

The source line contains an invalid character. This is probably caused by control characters embedded in the source line by an external editor.

Invalid compact EXE file. Rebuild EXE.

Your EXE file has become corrupt. Rebuild the file.

Invalid database number. (17)

Attempt to select a database number not in the range from 1 to 25.

Invalid DIF file header. (115)

The DIF file header of the file you're attempting to import is incorrect.

Invalid DIF type indicator. (117)

The DIF file attempting to import has an invalid data type indicator.

Invalid DIF vector — DBF field mismatch. (116)

The DIF file you are attempting to import has an internal conflict between its header and its data.

Invalid Excel version 2.0 file format. (1661)

The Excel file you are attempting to import is either not version 2.0 or has been corrupted.

Invalid field width or number of decimal places. (1713)

The field width or number of decimal places you have specified is not valid. Check the System Capacities table in the Microsoft FoxPro 2.5 *Update* manual for a list of the requirements.

Invalid file descriptor. (1111)

FoxPro is unable to open a file. If this error occurs, call Microsoft Product Support.

Invalid function argument value, type or count. (11)

A value passed to a function is either not of the expected data type, or the value of the argument is out of range for this function. This may also be caused by passing too many arguments to a function.

Invalid index number. (1141)

An attempt was made to SET ORDER TO a position in the index file list that is not occupied.

Invalid keyboard macro file format. (356)

An attempt has been made to use a macro file with invalid data.

Invalid key length. (112)

An invalid index key length has been specified. The length of a key for an .IDX index must be between 1 and 100 characters. The length of a key for a .CDX index must be between 1 and 254 characters.

Invalid .LIB signature. (1190)

Library file used to construct .EXE was corrupted.

Invalid Lotus 1-2-3 version 1.0 file format. (1662)

The Lotus® 1-2-3® file you are attempting to import is either not version 1.0 or has been corrupted.

Invalid Lotus 1-2-3 version 2.0 file format. (297)

The Lotus 1-2-3 file you are attempting to import is not version 2.0. Use the Lotus conversion utility to convert the file to version 2.0 before attempting to import it.

Invalid Lotus 1-2-3 version 3.0 file format. (1678)

The Lotus 1-2-3 file you are attempting to import is either not version 3.0 or has been corrupted.

Invalid Multiplan version 4.0 file format. (1670)

The Multiplan® file you are attempting to import is either not version 4.0 or has been corrupted.

Invalid or duplicate field name. (1712)

You have attempted to create a field in a database that already has that name or have used the characters ., “, /, \, [,], :, |, <, >, +, =, ;, *, or ? or a blank or a space in a field name.

Invalid or missing EXE file.

The EXE file you have specified either is corrupted or there is not sufficient rights to use it.

Invalid operation for CURSOR. (1115)

An attempt was made to PACK a cursor.

Invalid Paradox version 3.5 file format. (1688)

The Paradox® file you are attempting to import is either not version 3.5 or has been corrupted.

Invalid path or file name. (202)

You've attempted to execute a FoxPro command that contains an invalid path or file name.

Invalid printer redirection. (124)

Either a path to a printer is not established or the print device cannot be shared.

Invalid SET expression. (231)

An invalid argument has been used with the SET function.

Invalid Symphony version 1.0 file format. (1673)

The Symphony file you are attempting to import is either not version 1.0 or has been corrupted.

Invalid Symphony version 1.1 file format. (1674)

The Symphony[®] file you are attempting to import is either not version 1.1 or has been corrupted.

Invalid subscript reference. (31)

A subscript was used that is outside of the valid range of array subscripts defined in the DIMENSION statement. This may also be caused by using two subscripts with a one-dimensional array.

Invalid SYLK file dimension bounds. (120)

The file attempting to be imported is indicating invalid rows or columns — it is out of bounds.

Invalid SYLK file format. (121)

The file you are attempting to import is not in a valid SYLK file format.

Invalid SYLK file header. (119)

The file header of the file you are attempting to import has an incorrect SYLK file header.

Invalid variable reference. (1223)

A function was used where a memory variable was expected. This is caused by using a valid function name or abbreviation as an array name.

Invalid WINDOW file format. (1632)

The window save file (.WIN) being restored contains invalid data.

I/O operation failure. (1002)

MS-DOS is unable to perform a file or hardware operation.

Key label [“<label>”] is invalid. (1255)

The key label you have used is not supported. See ON KEY LABEL in the *FoxPro Language Reference* for a list of supported key labels.

Key string too long. (1257)

The character expression included with KEYBOARD is too long. Reduce its length.

Key too big. (1124)

The compiled version of the index expression has exceeded 150 bytes.

Label file invalid. (54)

An attempt was made to modify or print labels using a label file that was not created with CREATE/MODIFY LABEL.

Label nesting error. (1653)

A user-defined function in a label form has called LABEL FORM.

Left margin plus indent must be less than right margin. (221)

The sum of the specified left margin and paragraph indent exceeds the right margin.

Library file is invalid. (1691)

FoxPro cannot use the library file you’ve specified. Rebuild the library.

Line number must be less than page length. (222)

The line number used falls beyond the page length memory variable.

Line too long. (18)

The maximum length for a command line (2048 bytes) has been exceeded. Macro substitution may have caused the line to expand beyond the 2048 byte limit.

Link command failed. (1194)

A compilation error occurred and an .EXE was not created on the disk.

Localized product required for this environment.

There was an attempt to use a US version of FoxPro with a non-US system.

LOG() : Zero or negative. (58)

A zero or a negative number has been used as the argument of the natural log function.

LOG10() : Zero or negative. (292)

A zero or a negative number has been used as the argument for a base-10 logarithm.

Logical expression required.

The expression provided in the Expression Builder must be of logical type in this instance.

Macro not defined. (355)

An attempt has been made to play a macro that does not exist.

Maximum allowable menu items (128) exceeded. (1607)

The maximum number of items that may be contained in a menu (128) was exceeded.

Maximum allowable menus (25) exceeded. (1608)

The maximum number of menus that may be defined (25) was exceeded.

Maximum record length exceeded in import file. (392)

The file you are attempting to import has a record length greater than 4,001 bytes.

MEMO file is missing/invalid. (41)

An attempt was made to use a database file whose associated memo file (.DBT or .FPT) has been deleted, removed or cannot be found.

Memory Variable file is invalid. (55)

An attempt was made to RESTORE FROM a file that is not a valid memory (.MEM) file.

Menu file invalid. (1687)

The menu file you have specified has become corrupted.

Menu has not been activated. (178)

A attempt has been made to select from a menu before it has been activated.

Menu has not been defined. (168)

An attempt has been made to activate a menu that has not been defined.

Menu is already in use. (181)

An attempt has been made to activate a menu that is already active.

Menu item cannot be defined. (169)

An attempt was made to add a menu bar to a prompt that was defined with files or structure.

Menu item cannot be released. (170)

An attempt was made to release a prompt that was defined with files or structure.

Menu items/titles must be type CHARACTER. (1611)

A menu item or menu title was encountered which was defined with a data type other than character.

Menu/Popup was not pushed. (279)

An attempt has been made to pop a menu or popup that was not pushed.

Mismatched braces in key label. (1256)

A left or right brace is missing from a key label.

Mismatched case structure. (1213)

A CASE, ENDCASE, or OTHERWISE statement does not have a corresponding DO CASE statement.

Missing ((1304)

A function name is missing a left parenthesis.

Missing) (1300)

A function name is missing a right parenthesis. Either the current line contains a left parenthesis which has not been matched, or the function contains too many arguments and thus the right parenthesis is going undetected.

Missing , (1306)

FoxPro was expecting a comma and one was not found. This is usually caused through failure to include the proper number of arguments with a function.

Missing expression. (152)

A valid expression was expected, but not found.

Missing operand. (1231)

An operator that requires two operands was used without the second operand, such as ? (13+4)/.

Missing .RTT section. (1193)

The Microsoft FoxPro 2.5 Distribution Kit was not properly installed.

MULTISELECT/MOVERS not supported for PROMPT style popup. (1710)

Commands which have the clauses PROMPT FIELD, PROMPT FILE or PROMPT STRUCTURE do not allow the clauses MULTISELECT or MOVER.

Must be an array definition. (1232)

A DIMENSION statement contains a variable declaration without the required subscript arguments.

Must be a character, date or numeric key field. (1145)

The key field used to UPDATE ON must be character, date or numeric.

Must be a file variable. (1226)

A file variable (field) was used where a memory or array variable was required.

Must be a memory variable. (1225)

A memory variable or an array variable was used where a file variable (field) is required.

Nested aggregation not allowed. (1844)

Nesting of the aggregate functions MIN(), MAX(), SUM(), AVG(), COUNT(), NPV(), STD() or VAR() is not allowed in FoxPro version 2.5.

Nesting error. (96)

One of these situations has been encountered:

- A DO is present without a matching ENDDO or vice versa
- A SCAN is present without a matching ENDSCAN or vice versa
- A FOR is present without a matching ENDFOR or vice versa
- A LOOP or EXIT clause is present outside of the DO WHILE ... ENDDO, FOR ... ENDFOR or SCAN ... ENDSCAN commands.

Nested key labels are invalid. (1254)

You cannot nest key labels.

No bars have been defined for this popup. (166)

You must define bars for each popup.

No database is in USE. (52)

A database was not in use at the time FoxPro attempted to execute a database command.

No fields to process. (47)

No fields can be found, usually because a SET FIELDS TO was in effect.

No fields were found to copy. (138)

During an attempt to COPY to a file, no visible fields were found.

No memory for buffer. (1149)

It has proven impossible to allocate memory for a buffer. This message is very unusual and will occur only in situations where available memory is extremely limited. You should consider adding memory or removing some memory resident programs to give FoxPro more working memory.

No memory for file map. (1150)

It has proven impossible to allocate memory for a FoxPro internal resource. This message is very unusual and will occur only in situations where available memory is extremely limited. You should consider adding memory or removing some memory resident programs to give FoxPro more working memory.

No memory for file name. (1151)

It has proven impossible to allocate memory for a FoxPro internal resource. This message is very unusual and will occur only in situations where available memory is extremely limited. You should consider adding memory or removing some memory resident programs to give FoxPro more working memory.

No menu bar defined. (1604)

An attempt has been made to READ a menu bar that has not been defined.

No pads defined for this menu. (1621)

An attempt has been made to activate a menu that has no pads.

No PARAMETER statement found. (1238)

After doing a program with a parameter list, the called program must begin with a PARAMETER statement.

No popup menu defined. (1605)

An attempt has been made to activate a light-bar menu before defining a light-bar.

No previous printjob to match this command. (1649)

An ENDPRINTJOB command was encountered without a matching PRINTJOB command preceding it.

No such menu/item is defined. (1612)

The value(s) passed to a READ MENU TO or READ MENU BAR TO command does not correspond to a defined menu or menu item.

Not a character expression. (45)

An attempt was made to CALL a module with an expression whose value is not of the type character.

Not a database file. (15)

The file that FoxPro is attempting to use as a database contains an improper header.

Not a numeric expression. (27)

An attempt was made to use the SUM command on a non-numeric field.

Not a user-defined window. (1682)

You have attempted to perform a command containing a WINDOW clause on window that was not created by a user, for example, the Command window.

Not a valid Framework II database/spreadsheet. (256)

The file you are attempting to import is not a valid Framework II[™] file format.

Not a valid RapidFile database. (255)

The file you are attempting to import is not a valid RapidFile[®] file format.

Not enough disk space for "<file name>". (1160)

The operating system has returned an error to FoxPro indicating that there is no room on the disk to contain the data from the latest WRITE command. This is also erroneously returned under certain versions of Novell[®] when you try to extend a file that has no owner.

Not enough memory to USE database. (1600)

There was not enough memory to open an additional database.

Not suspended. (101)

A RESUME command was used without first suspending a program.

NOWAIT/SAVE/NOENVIRONMENT/IN/WINDOW clauses not allowed with FROM. (1696)

These clauses cannot be included when you create a quick report with the FROM clause.

Numeric overflow (data was lost). (39)

A mathematical operation resulted in a number that was too large to be stored in the field or variable in which it was placed.

Object file "<file>" is the wrong version. (1195)

The object file is out of date and the source code cannot be located.

Operator/operand type mismatch. (107)

An arithmetic, string, or logical operator or function is being used with an invalid data type. This error would occur, for example, if you were to attempt to add two logical values.

OS memory error. (1012)

There is a problem with your MS-DOS free memory chain.

PAD has not been defined. (164)

A reference has been made to a pad that does not exist.

Picture error in GET statement. (1217)

The PICTURE clause within an @ ... SAY ... GET statement contains a picture which cannot include the value to be formatted. This error may be occurring because the picture is in error or because the value to be formatted does not match the picture.

POPUP has not been activated. (179)

An attempt has been made to select a pad from a popup before the popup has been activated.

POPUP has not been defined. (165)

An attempt has been made to activate a popup before it has been defined.

POPUP is already in use. (182)

An attempt has been made to activate a POPUP that is already activated.

POPUP is too small. (287)

The popup window is too small to display selection bars. You'll have to define a larger popup.

Popup too big, first <expN> entries shown.

You have specified too many items to be shown in the popup.

Position is off the screen. (30)

A row or column number specified in an @ command is larger than the number of rows or columns on the screen, window or printer.

PREVIEW clause not allowed with OFF/NOCONSOLE or TO PRINT/FILE. (1681)

Page Preview is not allowed when the console is not on or when printing to the printer or a file.

Printer driver is corrupted. (1643)

An attempt has been made to load a printer driver that is corrupted.

Printer driver not found. (1644)

The specified printer driver could not be located.

Printer not ready. (125)

The printer device specified is currently not accessible or the printer is timing out. The TIME parameter in the CONFIG.FP file may need to be increased.

Printjobs cannot be nested. (1337)

After encountering a PRINTJOB command, an ENDPRINTJOB command was not found before another PRINTJOB command was encountered.

Procedure "<procedure>" not found. (1162)

The procedure specified with the IN clause of the DO command could not be located.

Product has not been properly installed.

Please reinstall your copy of FoxPro.

Program too large. (1202)

The program which FoxPro is attempting to load will not fit into memory. The largest program or individual procedure FoxPro can load is one containing 65,000 bytes.

Project file is invalid. (1685)

The project file specified has become corrupted.

Queue "<queue>" not found. (1716)

The queue specified in the SET PRINTER TO command cannot be found.

Record is in use by another. (109)

An attempt has been made to write to a record that is locked by another user.

Record is in use: Cannot write. (1502)

FoxPro could not write to the selected record because it is in use.

Record is not in index. (20)

A key field of the database in use has been modified without the index having been active. To correct this, REINDEX the database file.

Record is not locked. (130)

An attempt was made to BROWSE, CHANGE, EDIT, DELETE, GATHER, MODIFY MEMO, READ, RECALL or REPLACE a record that was not locked.

Record is out of range. (5)

The record number which you are attempting to access is beyond the actual number of records contained in the current database. Check to see if you are using the demonstration activation key. If

so, reinstall FoxPro with the live key. It is also possible that the index is out of date. REINDEX if this is the case.

Record too long. (1126)

While attempting to create a database file, the maximum length for the data portion of a record (4,000 bytes) was exceeded. The length of the data portion of a record is equal to the sum of the lengths of the individual record fields.

Recursive macro definition. (1206)

The maximum number of macro substitutions (256 in any one statement) has been performed in the current line. The probable cause of this error is that a macro references itself.

Relational expression too long. (1108)

The compiled version of the relational expression has exceeded 60 bytes.

Report file invalid. (50)

A report description contains an error.

Report nesting error. (1645)

A user-defined function in a report form has called REPORT FORM.

Required clause not present in command. (1221)

A command was used without a clause that was required by the FoxPro syntax.

RUN! command failed. (1405)

The operating system has returned an error to FoxPro indicating that it cannot create a process to execute a RUN command. Most often, this error is the result of the inability to find the shell program to be executed, or insufficient free memory to load the shell program into memory. Make sure the COMMAND.COM is accessible via the MS-DOS environment variable COMSPEC.

RUN! command string too long. (1411)

The command string included with the RUN command is too long. The maximum length allowed is approximately 240 characters.

Screen code too large for memory. (1507)

The screen object you are trying to copy has a code snippet larger than 64K.

Screen file invalid. (1686)

The screen file specified has become corrupted.

SELECT's are not UNION compatible. (1851)

Data types or sizes of fields being projected in each SELECT are not identical.

Server "<server>" not found. (1715)

The server specified with the SET PRINTER TO command could not be found.

Source code not found.

The file you have specified cannot be located in the directory specified.

Source code out of date.

The date source code you have specified does not match the compiled (.FXP) file.

SQL aggregate on non-numeric expression. (1811)

There was an attempt to perform an average, sum, standard deviation or variance on a non-numeric field.

SQL cancelled operation. (1839)

There was an attempt to SELECT into a table when SAFETY was ON and the user chose not to overwrite the table.

SQL column "<field> | <variable>" not found. (1806)

The field or variable you have specified cannot be found.

SQL could not locate database. (1802)

The database specified could not be found.

SQL err building temporary index. (1831)

A temporary index could not be built, possibly due to insufficient disk space.

SQL Error Correlating Fields. (1801)

An outer-reference can only be in syntax similar to the following:

$X = Y$

Syntax such as $X = Y + 1$ or $X = 5$ will produce this error.

SQL expression too complex. (1845)

FoxPro ran out of memory when it tried to expand the SELECT statement to analyze it.

SQL illegal GROUP BY in subquery. (1828)

When using one of the six operators +, -, *, /, < and >, there can only be one row of output.

SQL index not found. (1830)

FoxPro could not find the existing index.

SQL Internal Error. (1800)

An internal error has occurred. If this error occurs, call Microsoft Product Support.

SQL invalid * in SELECT. (1820)

There was an attempt to use * in one of the aggregate functions of MAX(), MIN(), AVG(), or SUM().

SQL invalid aggregate field. (1822)

There has been an attempt to do aggregation on a memo field.

SQL invalid DISTINCT. (1819)

There must only be one distinct used per level.

SQL invalid GROUP BY. (1807)

There is an error in the GROUP BY clause.

SQL Invalid HAVING. (1803)

There is an error in HAVING clause, for example the clause does not contain a logical expression.

SQL invalid ORDER BY. (1808)

One of the fields chosen for the order by is not in the SELECT list. This only occurs when using numeric indexing, for example

```
SELECT a,b,c FROM database ORDER BY 4
```

The number chosen exceeds the number of fields selected.

SQL invalid SELECT statement. (1804)

There is an error in the projection list.

SQL invalid SELECT. (1826)

There has been an attempt to select more than one field in a subquery.

SQL invalid subquery. (1825)

Something in the subquery is erroneous.

SQL invalid WHERE clause. (1833)

There is an error in the WHERE clause.

SQL Invalid use of subquery. (1810)

A subquery was used in a place where it wasn't expected.

SQL Invalid use of union in subquery. (1813)

UNION is not supported in subqueries in FoxPro version 2.5.

SQL no FROM clause. (1818)

There must be a FROM clause in the SELECT command.

SQL out of memory. (1809)

FoxPro has run out of memory while trying to process your SELECT command.

SQL queries of this type are not supported at present. (1814)

There was an attempt to perform a query that is not supported by FoxPro version 2.5.

SQL statement too long. (1812)

The object code is too long to be compiled.

SQL subqueries nested too deep. (1842)

FoxPro version 2.5 does not support nested subqueries.

SQL too many columns referenced. (1841)

There must be no more than 256 columns total referenced in any SELECT command.

SQL Too many subqueries. (1805)

The number of subqueries allowed in FoxPro version 2.5 is two per SELECT.

SQL Too many UNIONS. (1834)

The maximum number of UNIONS (10) has been exceeded.

SQRT domain error. (61)

The SQRT argument must not be negative.

Statement not allowed in interactive mode. (95)

The commands IF, ELSE, ENDIF, TEXT, ENDTEXT, EXIT, RETRY, RETURN, SUSPEND, DO WHILE, ENDDO, DO CASE, ENDCASE, SCAN, ENDSCAN, FOR and ENDFOR are not allowed in the interactive mode.

String memory variable area overflow. (21)

The combined length of all memory variable strings has exceeded the amount of memory.

String too long to fit. (1903)

The allowable string length was exceeded.

Structural CDX file not found. (1707)

The structural index file associated with a database file could not be found.

Structure invalid. (1235)

In CREATE FROM, a database was specified whose structure does not match the STRUCTURE EXTENDED format.

Structure nesting too deep. (1212)

The maximum structured programming command nesting of 64 levels has been exceeded.

Subscript out of bounds. (1234)

An attempt was made to reference an array element with a subscript that is outside of the range defined in the DIMENSION statement.

Subquery returned more than one record. (1860)

Subqueries linked with the six normal operators <>, =, <=, >=, < or >, excluding ALL or ANY, can only produce a one row result.

Syntax error. (10)

A command which is syntactically incorrect has been issued. A syntax error may be caused by a misspelled command or variable name, or by use of a clause which is not valid in the current context.

Tab stops must be in ascending order. (228)

The tab stops defined for the system variable _TAB must be in ascending order.

Target is already engaged in relation. (1147)

A situation has occurred where there is more than one relationship between the currently selected database file and one of its targets.

Too few arguments. (1229)

A function call contains less than the required number of arguments.

Too many arguments. (1230)

A function call contains more than the permitted number of arguments.

Too many extensions specified. (1694)

Too many file extensions have been included in a GETFILE(), LOCFILE() or PUTFILE() function. The maximum number of file extensions you may include is 30.

Too many files open. (6)

FoxPro has attempted to open more than its internal limit of files. This may possibly be caused by the CONFIG.SYS file setting not being set high enough.

Too many memory variables. (22)

The maximum number of memory variables which may be created has been exceeded.

Too many PICTURE characters specified. (1310)

The amount of characters permitted in a PICTURE clause is limited by memory.

Too many PROCEDURES. (1250)

The number of procedures is limited to the amount of memory you have.

Too many READs in effect. (1249)

The maximum nesting level for READs (5) in effect has been exceeded.

Too many names used. (1201)

As a program was being loaded or a database was being opened, FoxPro's name table was overflowed. To correct this, divide the program into smaller modules.

Too many record to BROWSE/EDIT in demo version. (1161)

The maximum number of records you can BROWSE or EDIT in the demo version of FoxPro is 120.

Too many relationships. (1148)

Too many relationships have been established between database files. Decrease the number of relations between open database files.

Total field type must be numeric. (1646)

A report expression that includes a total specification is not numeric type.

Total label width exceeds maximum allowable size. (1246)

The LABEL command detected a condition where the sum of the individual label widths plus separating spaces is greater than the maximum width allowed.

Transgressed node found during compaction. (1007)

If this error occurs, call Microsoft Product Support. You are returned to MS-DOS after FoxPro closes all open files.

Unable to create temporary work file(s). (1410)

A SORT or INDEX command which requires temporary work files was not permitted by the operating system to create them. This is caused by a full directory or a permissions problem concerning access to the temporary files directory.

Unable to generate printer driver. (1717)

You have not specified a printer driver through _GENPD and tried to use the command SET PDSETUP.

Unable to locate desired version of FoxPro.

The version of FoxPro requested is not on the designated path.

Unable to process error.

If this error occurs, call Microsoft Product Support. No error number is returned and you are returned to MS-DOS after FoxPro closes all open files.

Unknown error code <expN>.

If this error occurs, call Microsoft Product Support. No error number is returned and you are returned to MS-DOS after FoxPro closes all open files.

Unknown function key. (104)

An attempt was made to SET FUNCTION to a function key that does not match the name or number of an existing function key.

Unrecognized command verb. (16)

A word was used at the beginning of the command line which is not a valid FoxPro command.

Unrecognized phrase/keyword in command. (36)

A phrase beginning with an invalid keyword was used in a command line.

Unresolvable REGIONAL name conflict. (1692)

FoxPro has attempted to create a REGIONAL variable name that already exists.

Use of invalid handle. (1004)

If this error occurs, call Microsoft Product Support. You are returned to MS-DOS after FoxPro closes all open files.

Use of unallocated handle. (1005)

If this error occurs, call Microsoft Product Support. You are returned to MS-DOS after FoxPro closes all open files.

Use of transgressed handle. (1006)

If this error occurs, call Microsoft Product Support. You are returned to MS-DOS after FoxPro closes all open files.

Variable must be in selected database. (1134)

You can only use fields from the currently selected database in quick screens.

Variable ["<variable>"] not found. (12)

The specified variable could not be found.

View file invalid. (127)

An attempt has been made to open a view file with invalid data.

WINDOW coordinates are invalid. (332)

An attempt was made to define a window's coordinates outside the allowable range.

WINDOW has not been activated. (215)

An attempt was made to use a window which has not been activated.

WINDOW ["<window name>"] has not been defined. (214)

An attempt has been made to activate a window that has not been defined.

Worksheet A for Lotus 1-2-3 version 3.0 file is hidden. (1680)

FoxPro cannot import a worksheet that has been hidden.

Wrong length key. (1117)

The key field length does not match for an UPDATE or SET RELATION TO operation between two files.

Wrong number of parameters. (94)

The number of parameters specified in a DO ... WITH statement is greater than the number of parameters defined by the PARAMETER statement in the called program.

Numerical Listing of Error Messages

| Error Number | Text of Error Message |
|--------------|--|
| | Can't find ESO file. |
| | Can't find OVL file. |
| | Cannot allocate screen map. |
| | Cannot create program workspace. |
| | Cannot open configuration file. |
| | Cannot run on MS-DOS before version 3.0. |
| | CPU exhibits 32 bit multiply problem. |
| | ESL and ESO file do not match. |
| | EXE and OVL file do not match. |
| | Fatal error <expN> reporting error <expN>. |
| | Index tag or file name must be specified. |
| | Internal consistency error. |
| | Invalid compact EXE file. |
| | Invalid or missing EXE file. |
| | Localized product required for this environment. |
| | Logical expression required. |
| | Popup too big |
| | Product has not been properly installed. |
| | Source code not found. |
| | Source code out of date. |
| | Unable to locate desired version of FoxPro. |
| | Unable to process error. |
| | Unknown error code <expN>. |
| 1 | File ["<file>"] does not exist. |
| 3 | File is in use. |
| 4 | End of file encountered. |
| 5 | Record is out of range. |
| 6 | Too many files open. |
| 7 | File already exists. |
| 9 | Data type mismatch. |
| 10 | Syntax error. |
| 11 | Invalid function argument value, type or count. |
| 12 | Variable ["<variable>"] not found. |

| Error Number | Text of Error Message |
|--------------|---|
| 13 | ALIAS ["<alias>"] not found. |
| 15 | Not a database file. |
| 16 | Unrecognized command verb. |
| 17 | Invalid database number. |
| 18 | Line too long. |
| 19 | Index file does not match database. |
| 20 | Record is not in index. |
| 21 | String memory variable area overflow. |
| 22 | Too many memory variables. |
| 23 | Index expression is too big. |
| 24 | ALIAS name already in use. |
| 26 | Database is not ordered. |
| 27 | Not a numeric expression. |
| 30 | Position is off the screen. |
| 31 | Invalid subscript reference. |
| 34 | Illegal operation for MEMO field. |
| 36 | Unrecognized phrase/keyword in command. |
| 37 | FILTER requires a logical expression. |
| 38 | Beginning of file encountered. |
| 39 | Numeric overflow (data was lost). |
| 41 | MEMO file is missing/invalid. |
| 42 | CONTINUE without LOCATE. |
| 44 | Cyclic relation. |
| 45 | Not a character expression. |
| 46 | Illegal value. |
| 47 | No fields to process. |
| 50 | Report file invalid. |
| 52 | No database in USE. |
| 54 | Label file invalid. |
| 55 | Memory Variable file is invalid. |
| 58 | LOG() : Zero or negative. |
| 61 | SQRT domain error. |
| 62 | Beyond string. |
| 67 | Expression evaluator fault. |
| 78 | ** or ^ domain error. |

| Error Number | Text of Error Message |
|--------------|---|
| 91 | File was not LOAded. |
| 94 | Wrong number of parameters. |
| 95 | Statement not allowed in interactive mode. |
| 96 | Nesting error. |
| 101 | Not suspended. |
| 102 | Cannot create file "<file>". |
| 103 | DO nesting too deep. |
| 104 | Unknown function key. |
| 107 | Operator/operand type mismatch. |
| 108 | File is in use by another. |
| 109 | Record is in use by another. |
| 110 | Exclusive open of file is required. |
| 111 | Cannot write to a read-only file. |
| 112 | Invalid key length. |
| 114 | Index does not match database file. Recreate index. |
| 115 | Invalid DIF file header. |
| 116 | Invalid DIF vector — DBF field mismatch. |
| 117 | Invalid DIF type indicator. |
| 119 | Invalid SYLK file header. |
| 120 | Invalid SYLK file dimension bounds. |
| 121 | Invalid SYLK file format. |
| 124 | Invalid printer redirection. |
| 125 | Printer not ready. |
| 127 | View file invalid. |
| 130 | Record is not locked. |
| 138 | No fields were found to copy. |
| 152 | Missing expression. |
| 164 | PAD has not been defined. |
| 165 | POPUP has not been defined. |
| 166 | No bars have been defined for this popup. |
| 167 | BAR position must be a positive number. |
| 168 | MENU has not been defined. |
| 169 | Menu item cannot be defined. |
| 170 | Menu item cannot be released. |

| Error Number | Text of Error Message |
|--------------|---|
| 174 | Cannot redefine menu in use. |
| 175 | Cannot redefine popup in use. |
| 176 | Cannot clear menu in use. |
| 177 | Cannot clear popup in use. |
| 178 | MENU has not been activated. |
| 179 | POPUP has not been activated. |
| 181 | MENU is already in use. |
| 182 | POPUP is already in use. |
| 202 | Invalid path or filename. |
| 214 | WINDOW [“<window name>”] has not been defined. |
| 215 | WINDOW has not been activated. |
| 216 | Display mode not available. |
| 221 | Left margin plus indent must be less than right margin. |
| 222 | Line number must be less than page length. |
| 223 | Column number must be between 0 and 255. |
| 225 | “<name>” is not a memory variable. |
| 226 | “<name>” is not a file variable. |
| 227 | Invalid box dimensions. |
| 228 | Tab stops must be in ascending order. |
| 230 | Bad array dimensions. |
| 231 | Invalid SET expression. |
| 232 | “<name>” is not an array. |
| 255 | Not a valid RapidFile database. |
| 256 | Not a valid Framework II database/spreadsheet. |
| 279 | Menu/Popup was not pushed. |
| 287 | POPUP is too small. |
| 291 | ASIN() : Out of Range. |
| 292 | LOG10() : Zero or negative. |
| 293 | ACOS() : Out of Range. |
| 297 | Invalid Lotus 1-2-3 version 2.0 file format. |
| 332 | WINDOW coordinates are invalid. |
| 350 | Field must be a Memo field. |
| 355 | Macro not defined. |
| 356 | Invalid keyboard macro file format. |

| Error Number | Text of Error Message |
|--------------|---|
| 392 | Maximum record length exceeded in import file. |
| 1001 | Feature not available. |
| 1002 | I/O operation failure. |
| 1003 | Free handle not found. |
| 1004 | Use of invalid handle. |
| 1005 | Use of unallocated handle. |
| 1006 | Use of transgressed handle. |
| 1007 | Trangressed node found during compaction. |
| 1008 | Foreign node found during compaction. |
| 1009 | Incorrect handle found during compaction. |
| 1010 | Area size exceeded during compaction. |
| 1011 | Area cannot contain handle. |
| 1012 | OS memory error. |
| 1101 | Cannot open file "<file>". |
| 1102 | Cannot create file. |
| 1103 | Illegal seek offset. |
| 1104 | File read error. |
| 1105 | File write error. |
| 1108 | Relational expression too long. |
| 1111 | Invalid file descriptor. |
| 1112 | File close error. |
| 1113 | File not open. |
| 1115 | Invalid operation for CURSOR. |
| 1117 | Wrong length key. |
| 1124 | Key too big. |
| 1126 | Record too long. |
| 1127 | For/while need logical expressions. |
| 1130 | 'Field' phrase not found. |
| 1134 | Variable must be in selected database. |
| 1140 | FILTER expression too long. |
| 1141 | Invalid index number. |
| 1145 | Must be a character, date or numeric key field. |
| 1147 | Target is already engaged in relation. |
| 1148 | Too many relationships. |
| 1149 | No memory for buffer. |

| Error Number | Text of Error Message |
|--------------|--|
| 1150 | No memory for file map. |
| 1151 | No memory for file name. |
| 1152 | Cannot access selected database. |
| 1153 | Attempt to move file to different device. |
| 1154 | Invalid buffpoint call. |
| 1155 | Invalid buffdirty call. |
| 1156 | Duplicate field names. |
| 1157 | Cannot update file. |
| 1160 | Not enough disk space for "<file name>". |
| 1161 | Too many records to BROWSE/EDIT in demo version. |
| 1162 | Procedure "<procedure>" not found. |
| 1163 | Browse database closed. |
| 1164 | Browse structure changed. |
| 1165 | "<field>" is not related to the current work area. |
| 1178 | Application file "<file>" not closed. |
| 1190 | Invalid .LIB signature. |
| 1191 | Bad .LIB file. |
| 1192 | Internal .LIB undefined symbol error. |
| 1193 | Missing .RTT section. |
| 1194 | Link command failed. |
| 1195 | Object file "<file>" is the wrong version. |
| 1196 | "<file>" is not a FoxPro EXE file. |
| 1201 | Too many names used. |
| 1202 | Program too large. |
| 1206 | Recursive macro definition. |
| 1211 | If/else/endif mismatch. |
| 1212 | Structure nesting too deep. |
| 1213 | Mismatched case structure. |
| 1214 | Endtext without text. |
| 1217 | Picture error in GET statement. |
| 1220 | Invalid character in command. |
| 1221 | Required clause not present in command. |
| 1223 | Invalid variable reference. |
| 1225 | Must be a memory variable. |
| 1226 | Must be a file variable. |

| Error Number | Text of Error Message |
|--------------|---|
| 1229 | Too few arguments. |
| 1230 | Too many arguments. |
| 1231 | Missing operand. |
| 1232 | Must be an array definition. |
| 1234 | Subscript out of bounds. |
| 1235 | Structure invalid. |
| 1238 | No PARAMETER statement found. |
| 1241 | Improper data type in group expression. |
| 1243 | Internal error: Too many characters in report. |
| 1245 | Error in label field definition. |
| 1246 | Total label width exceeds maximum allowable size. |
| 1249 | Too many READs in effect. |
| 1250 | Too many PROCEDURES. |
| 1252 | Compiled code for this line too long. |
| 1253 | Cannot rename current directory. |
| 1254 | Nested key labels are invalid. |
| 1255 | Key label ["<label>"] is invalid. |
| 1256 | Mismatched braces in key label. |
| 1257 | Key string too long. |
| 1282 | Insufficient memory. |
| 1294 | FOXUSER file is invalid. |
| 1300 | Missing) |
| 1304 | Missing (|
| 1306 | Missing , |
| 1307 | Division by 0. |
| 1308 | Insufficient stack space. |
| 1309 | "<file>" is not an object file. |
| 1310 | Too many PICTURE characters specified. |
| 1337 | Printjobs cannot be nested. |
| 1405 | RUN! command failed. |
| 1410 | Unable to create temporary work file(s). |
| 1411 | RUN! command string too long. |
| 1412 | Cannot locate COMSPEC environment variable. |
| 1502 | Record is in use: Cannot write. |
| 1507 | Screen code too large for memory. |

| Error Number | Text of Error Message |
|--------------|--|
| 1600 | Not enough memory to USE database. |
| 1604 | No menu bar defined. |
| 1605 | No popup menu defined. |
| 1607 | Maximum allowable menu items (128) exceeded. |
| 1608 | Maximum allowable menus (25) exceeded. |
| 1611 | Menu item/titles must be type CHARACTER. |
| 1612 | No such menu/item is defined. |
| 1621 | No pads defined for this menu. |
| 1632 | Invalid WINDOW file format. |
| 1637 | File must be opened exclusively to convert Memo file. |
| 1642 | COLORSET resource not found. |
| 1643 | Printer driver is corrupted. |
| 1644 | Printer driver not found. |
| 1645 | Report nesting error. |
| 1646 | Total field type must be numeric. |
| 1647 | Improper data type in field expression. |
| 1649 | No previous printjob to match this command. |
| 1651 | CANCEL/SUSPEND is not allowed. |
| 1652 | Attempt to use FoxPro function as an array. |
| 1653 | Label nesting error. |
| 1657 | Column number must be between 0 and right margin. |
| 1659 | Cannot convert Memo file for a read-only database file. |
| 1662 | Invalid Lotus 1-2-3 version 1.0 file format. |
| 1661 | Invalid Excel version 2.0 file format. |
| 1670 | Invalid Multiplan version 4.0 file format. |
| 1671 | Cannot import from password protected file. |
| 1672 | Cannot append from password protected file. |
| 1673 | Invalid Symphony version 1.0 file format. |
| 1674 | Invalid Symphony version 1.1 file format. |
| 1678 | Invalid Lotus 1-2-3 version 3.0 file format. |
| 1679 | Import only Worksheet A for Lotus 1-2-3 version 3.0 files. |

| Error Number | Text of Error Message |
|--------------|---|
| 1680 | Worksheet A for Lotus 1-2-3 version 3.0 file is hidden. |
| 1681 | PREVIEW clause not allowed with OFF/NOCONSOLE or TO PRINT/FILE. |
| 1682 | Not a user-defined window. |
| 1683 | Index tag not found. |
| 1685 | Project file is invalid. |
| 1686 | Screen file invalid. |
| 1687 | Menu file invalid. |
| 1688 | Invalid Paradox version 3.5 file format. |
| 1689 | Cannot build APP/EXE without a main program. |
| 1690 | Database operation invalid while indexing. |
| 1691 | Library file is invalid. |
| 1692 | Unresolvable REGIONAL name conflict. |
| 1693 | Cannot find screen/menu generation program. |
| 1694 | Too many extensions specified. |
| 1695 | COLUMN/FORM/ALIAS/NOOVERWRITE/WIDTH allowed only with FROM clause. |
| 1696 | NOWAIT/SAVE/NOENVIRONMENT/IN/WINDOW clause not allowed with FROM. |
| 1698 | COLUMN/ROW/ALIAS/NOOVERWRITE/SIZE/SCREEN allowed only with FROM clause. |
| 1705 | File access denied. |
| 1706 | Descending not permitted on IDX files. |
| 1707 | Structural CDX file not found. |
| 1708 | File is open in another work area. |
| 1709 | Cannot load PROAPI16.EXE. |
| 1710 | MULTISELECT/MOVERS not supported for PROMPT style popup. |
| 1711 | API library revision mismatch. Rebuild library. |
| 1712 | Invalid or duplicate field name. |
| 1713 | Invalid field width or number of decimal places. |
| 1715 | Server "<server>" not found. |
| 1716 | Queue "<queue>" not found. |
| 1717 | Unable to generate printer driver. |
| 1718 | File is read only. |

| Error Number | Text of Error Message |
|--------------|---|
| 1719 | Cannot build APP/EXE while suspended. |
| 1720 | Cannot SET FORMAT while in a READ with a FORMAT file. |
| 1721 | All work areas in use. |
| 1800 | SQL Internal Error. |
| 1801 | SQL Error Correlating Fields. |
| 1802 | SQL could not locate database. |
| 1803 | SQL Invalid HAVING. |
| 1804 | SQL invalid SELECT statement. |
| 1805 | SQL Too many subqueries. |
| 1806 | SQL column "<field> <variable>" not found. |
| 1807 | SQL invalid GROUP BY. |
| 1808 | SQL invalid ORDER BY. |
| 1809 | SQL out of memory. |
| 1810 | SQL Invalid use of subquery. |
| 1811 | SQL aggregate on non-numeric expression. |
| 1812 | SQL statement too long. |
| 1813 | SQL Invalid use of union in subquery. |
| 1814 | SQL queries of this type are not supported at present. |
| 1815 | "<cursor>" must be created with SELECT ... INTO TABLE. |
| 1818 | SQL no FROM clause. |
| 1819 | SQL invalid DISTINCT. |
| 1820 | SQL invalid * in SELECT. |
| 1822 | SQL invalid aggregate field. |
| 1825 | SQL invalid subquery. |
| 1826 | SQL invalid SELECT. |
| 1828 | SQL illegal GROUP BY in subquery. |
| 1830 | SQL index not found. |
| 1831 | SQL err building temporary index. |
| 1832 | "<field> <variable>" is not unique and must be qualified. |
| 1833 | SQL invalid WHERE clause. |
| 1834 | SQL Too many UNIONS. |
| 1839 | SQL cancelled operation. |

| Error Number | Text of Error Message |
|--------------|---|
| 1841 | SQL too many columns referenced. |
| 1842 | SQL subqueries nested too deep. |
| 1844 | Nested aggregation not allowed. |
| 1845 | SQL expression too complex. |
| 1846 | Cannot GROUP by aggregate field. |
| 1851 | SELECTs are not UNION compatible. |
| 1860 | Subquery returned more than one record. |
| 1903 | String too long to fit. |
| 1907 | Bad drive specifier. |
| 1908 | Bad width or decimal place argument. |
| 1910 | Illegal printer driver recursion. |
| 1999 | Function not implemented. |

Index

A

- Activating browse windows, 2-90
- Activating a menu system, 3-8, 4-2
- API external routines, 17-13
- .APP file extension, 1-1, 5-2, 17-3
- APPEND
 - BLANK, 16-10, 16-12
 - FROM, 16-10
 - FROM ARRAY, 9-11, 16-10
 - MEMO, 16-10
- Application programs
 - Importing, 15-10
- Array functions, 9-4
- Arrays, 9-1, 15-13
 - And SQL SELECT, 9-13
 - Creating, 9-2
 - Creating multiple, 9-3
 - Initializing, 9-5
 - Limitations, 9-9
 - Manipulating, 9-5
 - Passing to UDFs, 9-10
 - Private, 9-8
 - Public, 9-8
 - Redimensioning, 9-7, 15-13
 - Referencing elements, 9-5
 - Transferring data, 9-11
 - With screen controls, 9-14
- Associated Window list
 - READ WITH command, 2-12, 2-91

B

- Backup
 - Importance of, 13-6
- Basic optimizable expressions, 14-5
- Batch command files
 - Executing, 15-12
- BROWSE, 16-10, 16-19
- Browse with screens, 2-89

C

- C language, and printer drivers, 17-5, 17-13, 17-24
- Calling menu programs, 4-4
- Calling screen programs, 2-10, 4-5
- CHANGE, 16-19, 16-29
- Check boxes (Screens), 2-75
- Cleanup and procedure code (Screens), 2-37
- Cleanup code (Menus), 3-16, 3-24, 3-25
- Code snippets, 2-3, 2-7, 3-1, 3-5
- Collisions, 16-23
- Color
 - Using with menus, 3-29, 3-34
 - Using with screens, 2-17
- See also
 - Input and Output — Data Formatting
- Combining basic optimizable expressions
 - Rushmore, 14-6
- Combining complex optimizable expressions, 14-8
- Command (Menu Builder)
 - Assigning to menu pad, 3-14
- Commands for saving/restoring
 - current environment, 5-14
- Commands for text merge, 11-1
- Commands benefiting from Rushmore, 14-4
- Comment boxes (Menus), 3-41
- Communications ports, Access to, 10-10
- Compact single entry indexes, 15-14
- Compilation errors, 6-3
 - Causes, 6-4
- COMPILE command, 15-10
- Compiled program files, 15-10
- CONFIG.FP, 15-2, 15-11, 16-3
 - EDITWORK, 16-4
 - INDEX, 15-9
 - OVERLAY, 16-4
 - Printer driver, 17-11
 - PROGWORK, 16-4
 - RESOURCE, 16-4
 - Search for, 16-2, 16-4
 - SET commands, 16-3
 - SORTWORK, 16-4
 - TMPFILES, 16-4
- Configuration, 16-2

- Configuration files (FoxDoc)
 - CONFIG.FXD, 13-49
 - Creating, 13-45
 - Multi-user, 16-2
 - Naming, 13-5
 - Retrieving, 13-5
 - Specifying, 13-49
 - Using, 13-45
- Context-sensitive help, 12-1, 13-4
- Control-Z, 15-10
- Converting from FoxBASE+, 15-8
- Converting programs, 15-8 - 15-14
- Coordinating Browse with screens, 2-89
 - Activating Browse windows, 2-90
 - Activating menus during
 - MODAL READ, 2-92
 - Sizing and positioning Browse windows, 2-91
- Coordinating screens and menus, 4-1
- COPY TO ARRAY command, 9-11
- COPY TO command and FoxBASE+, 15-9
- Creating a help database, 12-3
- Creating applications without programming, 3-1
- Creating custom printer drivers, 17-13
- Creating menus, 3-1, 3-20
 - Color scheme, 3-29
 - Placement of menu bar, 3-26
- Creating projects, 5-1, 5-2
- Creating screens, 2-1
- Cross references among help topics, 12-4

D

- .DBT file extension, 15-9
- Deadly embrace, 16-14
- Debugging
 - Generated programs, 2-93
 - Menus, 3-40
 - Tips, 6-6
- DECLARE command, 9-2
- DEFINE BAR/PAD commands (Menus), 3-39
- DEFINE POPUP command
 - COLOR SCHEME clause, 3-34
- DEFINE WINDOW command, 12-9
- DELETE, 16-10
- Delimiters
 - Text merge, 11-2

- Designing menus, 3-12
- DIMENSION command, 9-2, 15-13
- DISPLAY STATUS command, 10-9, 16-30
- DO command in CONFIG.FP, 15-11
- Documentation
 - Output, 13-2
- Documenting Source Code
 - FoxDoc, 13-1
- DOS MODE command, 10-10
- Drive designations, 13-11

E

- EDIT, 16-10 - 16-11
- EMPTY() function, 14-11
- Environment settings, 5-14
- Error handling
 - FoxDoc, 13-8
 - Multi-user, 16-14
- Errors
 - During execution, 6-5
 - When compiling, 6-3
- Exclusive use, 16-6
- Executing batch command files, 15-12
- Executing programs, 15-11
- Extended display, Debugging with, 6-6
- External API routines, 17-13
- EXTERNAL commands, 5-9

F

- FCHSIZE() function, 10-9
- FCLOSE() function, 10-8
- FCREATE() function, 10-3
- FEOF() function, 10-9
- FERROR() function, 10-9
- FFLUSH() function, 10-9
- FGETS() function, 10-6
- File locking, 16-17
- File pointers, 10-6
- FLOCK(), 16-13
- .FMT file extension, 15-8
- FOPEN() function, 10-5
- FOR clause
 - Rushmore use in optimization, 14-4
- Form letter, 11-3
- Fonts database (printer drivers), 17-6
- .FOX file extension, 15-8

FoxBASE+ compatibility

- Additional SET options, 15-3
- Data and program files, 15-4
- Emulating keystrokes, 15-2
- Unavoidable differences, 15-4

FoxBASE+, converting from, 15-8

FoxDoc

- Action diagram, 13-24
- Action diagram symbols, 13-62
- BACKDBF.BAT, 13-57
- BACKPRG.BAT, 13-57
- Batch files, 13-57
- Batch files errors, 13-45
- Batch files, Using, 13-45
- Called programs, 13-28
- Calling programs, 13-29
- CAPITAL command, 13-43
- Commands, 13-40
- Command line switches, 13-47
- CONFIG.FXD, 13-4
- Cross-reference codes, 13-55
- Cross-referencing, 13-26 - 13-27
- Default options, Changing, 13-49
- Default options, Restoring, 13-49
- Default options, Saving, 13-49
- DOCCODE, 13-42
- DOCMACRO, 13-40
- Drive designations, Ignoring, 13-38
- Error handling, 13-8
- EXPAND command, 13-43
- File type identification, 13-51
- Format/Action Diagram Options Screen, 13-19
- FORMAT command, 13-43
- FoxBASE+ keyword file, 13-22
- FOXDOC.EXE, 13-4
- FOXDOC.HLP, 13-4
- FoxPro keyword file, 13-22
- Function key options, 13-4, 13-34
- FXPWORDS.FXD, 13-4, 13-22
- Headings file, 13-30
- Headings Options Screen, 13-28
- INDENT command, 13-43
- Keyword capitalization, 13-21
- Keyword file, 13-59
- Macro substitution, 13-40, 13-52
- Macro substitution, Disabling, 13-41
- Macro substitution files, 13-40
- MACRO.FXD, 13-41
- Main menu, 13-5
- Memory usage, 13-46

FoxDoc (cont'd)

- Moving around, 13-4 - 13-5
- OFF command, 13-43
- ON command, 13-43
- Other Options Screen, 13-38
- Path, 13-6
- Paths, 13-10 - 13-12, 13-21
- PRINTDOC.BAT, 13-58
- Printing Options Screen, 13-34
- Printing w/out documenting, 13-34
- PROWORDS.FXD, 13-4, 13-22
- Pseudo program statements, 13-42
- Report Screen, 13-13
- Sample reports, 13-64
- Source code format, 13-22 - 13-25
- Source code indentation, 13-61
- Status Screen, 13-7
- SUSPEND command, 13-43
- System files, 13-4
- System Screen, 13-9
- Tree Diagram Screen, 13-31
- UPDATE.BAT, 13-57
- With generated programs, 2-90
- Xref (Cross-Reference) Options Screen, 13-26
- XREF command, 13-43
- FOXHELP database structure, 12-2
- FOXPROCFG, 16-3
- FOXPROLOVL, 16-2
- FOXUSER, 15-13, 16-4
- FOXUSER.DBF | .FPT, 15-13
- .FPT file extension, 15-8
- FPUTS() function, 10-7
- FREAD() function, 10-6
- .FRM file extension, 15-8
- FSEEK() function, 10-9
- Functions for text merge, 11-1
- FWRITE() function, 10-8
- .FXP file extension, 15-10

G

GATHER MEMVAR

- command, 2-16, 9-11, 16-10
- General Options dialog (Menus), 3-20
- Generated screen code, 2-3
 - Debugging, 2-93
 - Using FoxDoc with, 2-95
- Generated menu program (.MPR), 3-16
- Generated screen program (.SPR), 2-18
- Generator directives (Screens), 2-31

Generator-named procedure, 2-3, 2-8
GENMENU program generator, 3-16
_GENPD, 17-28
GENPD.APP (Printer support), 17-4
GENSCRN program generator, 2-18

H

Hardware requirements, 16-2
HEADER() function, 10-9
Help

- Context-sensitive, 12-1, 13-4
- Command
 - IN WINDOW option, 12-9
- Database
 - Requirements for creating, 12-3
- Database structure, 12-2

Hierarchical popups, 3-4, 3-14
Home directory, 16-10
Hot keys (Screens), 2-16

I

IBM's System Application

- Architecture, 15-2

.IDX file extension, 15-8
.IDX indexes, 15-61
Importing application programs, 15-8
Index files, 16-11

- Compact single entry, 15-9
- Default extension, 15-9

.NDXs, 15-9
Initializing arrays, 9-5
Input screens

- See Screen Builder

J

Join condition, 14-11

K

Key label assignments for ON KEY
LABEL, 12-7
Keyboard macros, 15-2

Keyboard shortcuts

For menu options, 3-15
For screen controls, 3-15, 4-5

L

.LBL file extension, 15-8
Library Construction Kit, 17-5
LIST STATUS command, 10-9
Lists (Screens), 2-84
Location of menu system (Menus), 3-26
LOCK(), 16-13
Locking operations (multi-user)

- Automatic, 16-10
- Automatic vs. Manual, 16-9
- Files, 16-8
- Manual, 16-13
- Records, 16-8

Low-level file functions

- Summarized, 10-1

Low-level multi-user functions, 16-16

M

Macro substitution, 14-11
Main file (Project Manager), 5-7
Mark (Menus)

- Global to menu system, 3-27

.MEM file extension, 15-8
Memory variables

- Menus, 3-21

Menu Bar Options dialog, 3-28
Menu Builder, 3-1

- Assigning command to menu pad, 3-14
- Assigning procedure to menu pad, 3-14
- Assigning submenu to submenu
 - option, 3-14
- Cleanup code, 3-16, 3-24, 3-25
- Code snippets, 3-5
- Color scheme, 3-29
- Comment boxes, 3-41
- Comments, 3-5
- Creating popups, 3-13
- Debugging menus, 3-40 - 3-41
- Generated menu code, 3-16, 3-30
- Hot keys for prompts, 3-37
- Location of menu system, 3-26
- Mark, global to menu system, 3-27

- Menu Builder (cont'd)
 - Mark, specific to menu pads and submenu options, 3-30
 - Menu Definition code, 3-16
 - Menu Design window, 3-1
 - Procedures (generated), 3-16
 - Quick Menu, 3-11
 - Result popup, 3-14
 - Setup code, 3-16
- Menu creation, 3-14
- Menu Definition code (generated), 3-16
- Menu Design window, 3-1
- Menu menu, 3-20
- Menu popups, 3-4, 3-13
- Menu programs (.MPR)
 - Calling, 4-4
 - Generating, 3-16
- Menus
 - Activating, 3-8, 4-2
 - Keyboard shortcuts, 4-6
 - Popping, 4-3
 - Pushing, 4-3
- Merging text, 11-2
- MESSAGE(), 16-34
- .MNX file extension, 3-4
- MODAL READ, 2-11, 3-8
- MODIFY MEMO command, 16-10
- .MPR file extension, 3-4, 3-7
- .MPX file extension, 3-4, 3-7
- _MSYSMENU (System menu bar), 3-3, 3-37
- Multi-database query optimization, 14-3
- Multiple arrays, 9-3
- Multiple databases and Rushmore, 14-4

N

- Name expressions
 - Advantages, 14-11
- Narrowing help topics displayed, 12-5
- .NDX file extension, 15-8

O

- Object level clauses (Screens), 2-3, 2-54
- ON KEY LABEL command, 12-7
- ON SELECTION BAR command, 3-5, 3-14
- ON SELECTION MENU command, 3-23

- ON SELECTION PAD command, 3-5
- Optimizing combinations of
 - complex expressions, Rushmore, 14-7
- Optimizing combinations of
 - basic expressions, 14-6
- ORGANIZER application, 2-1
- Overlay file, 16-2

P

- Passing arrays to UDFs, 9-10
- Path
 - FoxDoc, 13-6
 - _PDSETUP, 17-28
- Performance
 - Optimizing, 14-12, 16-17
 - FOR...ENDFOR vs. DO WHILE...ENDDO, 14-12
 - Memory availability, 14-10
 - Name expressions vs. macro substitution, 14-11
 - SCATTER TO ARRAY vs. SCATTER MEMVAR, 14-12
- .PLB file extension, 17-5
- Popping menus, 3-10, 4-3
- Popups (Screens), 2-78
- Postscript fonts, 17-6
- Power tools, 1-1, 5-1
- .PRG file extension, 15-8
- Printer drivers
 - Creating custom drivers, 17-13
 - Files listing for GENPD project, 17-4
 - For Postscript printers, 17-5
 - Procedures, 17-14
 - Tips for creating, 17-20
- Private arrays, 9-8
- Procedures
 - Assigning to menu pad (Menus), 3-14
 - Gathering into a project, 14-11
 - Menus, 3-5
 - Printer driver program, 17-14
 - Screens, 2-40
- Program cache, 16-2
- Program execution errors, 6-5
- Program templates
 - Text merge and, 11-9
- Project components, 5-3
- Project creation, 5-1 - 5-2

- Project Manager, 5-1
 - Set main, 5-7
- PROMPT() function, 3-33
- Public arrays, 9-8
- Push buttons (Screens), 2-65
- Pushing menus, 3-10, 4-3

Q

- Quick Menu (Menus), 3-11

R

- Radio buttons (Screens), 2-71
- RAM disk
 - Usage, 16-10
- READ command, 2-11, 16-20
 - And menus, 3-8, 4-2
 - MODAL, 2-11, 3-8, 4-2
- READ level clauses (Screens), 2-3, 2-44
- READ WITH, 2-12
- Read-only access, 10-5, 16-8
- RECALL, 16-20
- Record locking, 16-8
- Redimensioning arrays, 9-7
 - With DIMENSION, 15-13
- Referencing array elements, 9-5
- REGIONAL variables, 2-5, 2-29
- REPLACE, 16-11
- Report variable hints, 8-1
- Reports
 - Importing from earlier
 - FoxPro versions, 15-13
- Resource file, 15-13
- RETRY, 16-14
- RLOCK(), 16-13
- RUN MODE command, 10-10
- Runtime errors, 6-5
- Rushmore Technology, 14-2
 - And Extended Version
 - FoxPro 2.0, 14-2
 - And FOR clause, 14-4
 - Disabling with NOOPTIMIZE, 14-9

S

- SAA, 15-2
- Saving/restoring environment
 - (Projects), 5-13
- SCATTER MEMVAR command, 2-16, 9-11
- Screen Builder, 2-1
 - Access order of screens, 2-15
 - Advantages, 2-2
 - Check boxes, 2-75
 - Code snippets, 2-7
 - Color, 2-17
 - Coordinating Browse with screens, 2-89
 - GATHER MEMVAR command, 2-16
 - Generator-named procedure, 2-8
 - Hot keys, 2-16
 - Lists, 2-84
 - MODAL READ, 2-11
 - Object level clauses, 2-54
 - Popups, 2-78
 - Push buttons, 2-65
 - Radio buttons, 2-71
 - READ command, 2-11
 - READ level clauses, 2-44
 - READ WITH command, 2-12
 - REGIONAL variables, 2-5
 - SCATTER MEMVAR command, 2-16
 - Screen sets, 2-6
 - Terms defined, 2-3
 - Utility screens, 2-4
 - Window definitions, 2-43
 - Window types, 2-43
- Screen code examples
 - @ ... GET/EDIT ERROR, 2-62
 - @ ... GET/EDIT VALID, 2-60, 2-61
 - @ ... GET/EDIT WHEN, 2-57, 2-58
 - @ ... SAY Refresh, 2-63
 - Check box VALID, 2-76, 2-77
 - Cleanup and procedure code, 2-37
 - List — 1st Element, #Elements, 2-85
 - List, moving between two lists, 2-87
 - Popup, array with VALID, 2-81
 - Popup, defined with SQL SELECT, 2-82
 - Popup, list with VALID, 2-79
 - Push Button VALID, 2-67, 2-68
 - Push Button WHEN, 2-70
 - Radio Button VALID, 2-72, 2-73
 - READ ACTIVATE, 2-45
 - READ SHOW, 2-44, 2-49, 2-50
 - READ WHEN, 2-52, 2-53
 - Setup code, 2-25

- Screen design
 - Design considerations, 2-15
 - Working environment, 2-14
- Screen output, 14-10
- Screen programs (.SPR)
 - Calling, 2-10, 4-5
- Screen sets, 2-3, 2-6
- See Also references in FOXHELP, 12-4
- SELECT command (SQL)
 - DISTINCT clause, 7-7, 7-12
 - Popup defined with, 2-82
- SET
 - EXCLUSIVE, 16-6
 - REPROCESS, 16-14
- SET DEVELOPMENT ON command, 15-10
- SET FORMAT TO command, 15-8
- SET HELP TO command, 12-5
- SET HELPFILTER command, 12-1, 12-7
- Set Main (Project Manager), 5-7
- SET MARK OF MENU command, 3-27
- SET OPTIMIZE command, 14-9
- SET PDSETUP command, 17-22
- SET SKIP command, 3-8
- SET SYSMENU command, 3-9, 4-3
- SET TOPIC TO command, 12-5
- Setup code (Screens), 2-27, 2-90
 - Generator directives, 2-31
- Setup code (Menus), 3-16
- Shared use of files, 16-16
- SHOW GET vs.
 - SHOW GETS commands, 2-48
- Sort files, 16-11
- .SPR file extension, 2-3, 2-18
- SQL SELECT command, 7-1, 14-4
 - And arrays, 9-13
 - And screens, 2-82
- SYS(2013) function, 3-13
- SYS(2015) function, 3-5
- System Application Architecture
 - (SAA), 15-2
- System menu, 3-2

T

- Templates
 - Text merge and, 11-9
- Temporary files, 16-2
- Text editor, 16-3

- Text merge
 - _PRETEXT, 11-6
 - Commands and functions, 11-1
 - Component evaluation conditions, 11-2
 - Delimiters, 11-2
 - Directing output, 11-7
 - Example, 11-3
 - Form letter, 11-1
 - Program templates, 11-9
 - TEXT...ENDTEXT, 11-3
- Tips
 - For debugging, 6-6
 - For improved application performance, 14-10
 - For printer driver programs, 17-20
 - Report variables, 8-1
- Trace window, 3-40
- Transferring data between
 - arrays and databases, 9-11
- Transferring items between two
 - lists (Screens), 2-87
- Trapping for keystrokes/mouse
 - clicks (Help), 12-7

U

- UPDATE, 16-11
- USE EXCLUSIVE, 16-6
- User-named procedures (Screens), 2-3, 2-39
- Utility screens, 2-3
 - Naming variables in, 2-4

W

- Window definitions (Screens), 2-43
- Window types (Screens), 2-43
- Write access, 10-5, 16-8