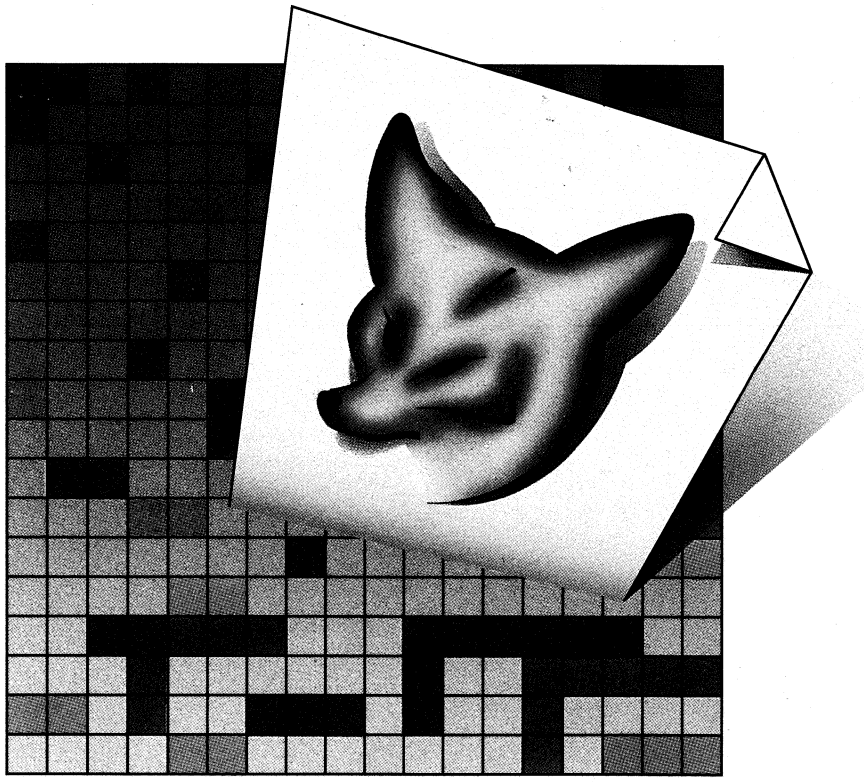
FoxPro



Developer's Guide

Includes Putting It All Together

Fox Software

Overview of the Documentation

Getting Started introduces you to FoxPro and gives you a quick tour of its powerful features. After the quick tour, you can continue to the tutorial and learn how easy it is to use the features previewed in the quick tour.

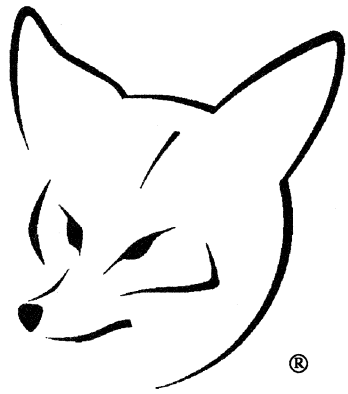
Interface Guide begins with an introduction to the interface followed by an item-by-item tour of the interface. Each menu option is described along with any related dialogs and windows. FoxPro power tools, which allow you to build interface components and access your data without programming, are also described in this manual.

Commands & Functions is a complete reference for the FoxPro programming language. It contains detailed descriptions of all FoxPro commands and functions in alphabetical order, plus an alphabetical listing of system memory variables.

Developer's Guide contains an extended discussion of recommended techniques for designing and implementing FoxPro 2.0 applications. Suggested usage of new FoxPro 2.0 power tools is illustrated by numerous examples. Advanced topics such as customization, compatibility and optimization are also included in this manual.

Help is always a keystroke away when you use FoxPro. To see context-sensitive help for the active window, dialog or menu option, press F1. For a command, function or system memory variable, select it and press F1. Each help topic is linked with related help topics for easy access.

FoxPro™



Developer's Guide

May 1991

Fox Software, Inc.
134 W. South Boundary
Perrysburg, Ohio 43551

Trademarks

FoxBASE®, FoxBASE®+, FoxBASE®+/Mac, FoxPro™ and Rushmore™ are trademarks of Fox Holdings, Inc. Other trademarks may be used throughout this manual.

Contents

Overview: Putting It All Together

Screens

Menus

Coordinating Screens and Menus

Project — The Main Organizing Tool

Debugging Your Application

SQL Quiz

Report Variable Hints

Arrays

Low-Level File I/O

Text Merge

Customizing Help

Documenting Applications with FoxDoc

Documenting Applications with FoxDoc (cont'd)

Customizing FoxPro

Optimizing Your System

Optimizing Your Application

Compatibility

Multi-User FoxPro

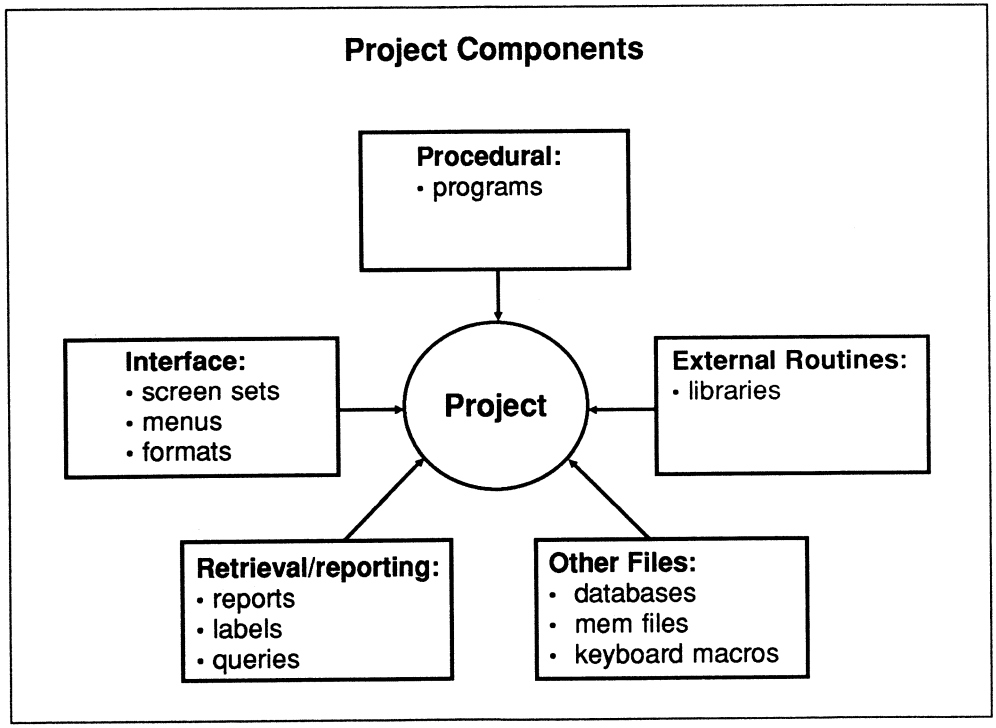
Appendices

Index

Putting It All Together

1 Overview: Putting It All Together

Chapters in Putting It All Together provide insight into using FoxPro's 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 an .EXE, and the updating of applications when components change. A project is the unifying mechanism that gathers the pieces of an application together, as shown below.



Each chapter in Putting It All Together includes examples, explanations and recommended techniques to help you get the most out of FoxPro. Special attention is focused on options that offer significant or unique opportunities for your applications.

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 to operate the power tools, refer to the FoxPro *Interface Guide* or the FoxPro *Getting Started* manual. If you are interested in details about a particular command or function, refer to the FoxPro *Commands & Functions* manual.

Chapters in Putting It All Together contain examples from the ORGANIZER sample application provided with FoxPro 2.0. The ORGANIZER is located in the directory SAMPLE. Feel free to investigate ORGANIZER in depth, use its techniques and develop your own approaches.

To use ORGANIZER, just execute ORGANIZE.MPR. Two options are added to the **System** menu popup: **Conversions** and **Organize...**. **Conversions** allows you to convert from one unit of measurement to another. When you choose **Organize...**, a submenu appears displaying the following options:

- **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 your business transaction information.
- **Restaurants** organizes information about restaurants.
- **Family & Friends** organizes information about family members and friends.

Code used in the examples in this manual may differ slightly from the ORGANIZER code on disk.

The ORGANIZER consists of seven projects: ACCNTS.PJX, TRANS.PJX, CLIENTS.PJX, FAMILY.PJX, CREDIT.PJX, CONVERT.PJX and RESTAURS.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. What a headache!

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 you own library of reusable screens to reflect you 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 *Commands & Functions* manual.

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 may 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 a 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 *Interface 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.

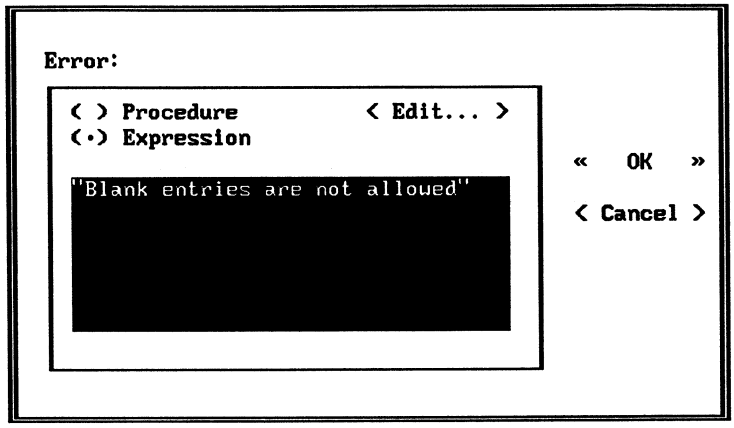
When you define and generate screen sets containing multiple screens, setup code from successive screens is concatenated with the setup code from the first screen. It is possible to write code for one screen that will unintentionally change the desired output of code written for a previous screen. Careful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

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

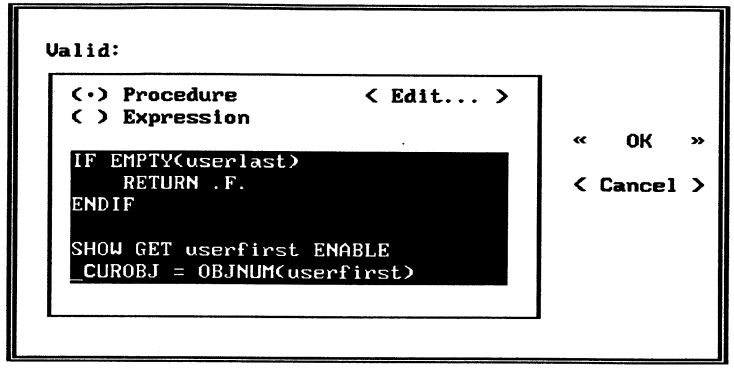
If an expression is defined for a clause, the expression is inserted in the generated program with the associated clause.




```
@ 8, 41 ... GET username ;
                .
                .
                .
        ERROR "Blank entries are not allowed"
```

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.



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

```
@ 8, 41 GET userlast ;
                .
                .
                .
        VALID _puu17rjed( )
```

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.

```
*       ************************************************************
*       * _PUU17RJED          userlast VALID                      *
*       *                                                        *
*       * Function Origin:                                      *
*       *                                                        *
*       * From Screen:         ADDUSERS,     Record Number:   10 *
*       * Variable:            userlast                          *
*       * Called By:           VALID Clause                      *
*       * Object Type:         Field                             *
*       * Snippet Number:      6                                 *
*       ************************************************************
*
FUNCTION _PUU17RJED      && userlast VALID
#REGION 1
IF EMPTY(userlast)
        RETURN .F.
ENDIF

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

Function origin comments

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.

Screen programs have been provided with a different extension so your screens can have the same base names as menus, projects, queries, programs, etc., 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 can 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 with 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 activate. 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(s) you want to include to 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 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 on 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.

Cut, Copy and Paste 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 Control

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 *Interface 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 *Interface Guide*. For information on color schemes, see the Color chapter in this manual.

Following is a list of tips for using color:

- Use color as a complimentary 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
- Setup Code — Section 2
- Define window commands
- 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.

```
*       ************************************************************
*       * 04/27/91              CONVERT.SPR              10:57:12 *
*       ************************************************************
*       * Author's Name                                           *
*       *                                                        *
*       * Copyright (c) 1991 Company Name                        *
*       * Address                                                *
*       * City,      Zip                                         *
*       *                                                        *
*       * 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

```
*       ************************************************************
*       *             CONVERT Setup Code - SECTION 1               *
*       ************************************************************
SET PROCEDURE TO utility
ON ERROR DO errorhandler WITH MESSAGE( ), LINENO( )
CLEAR PROGRAM
CLEAR GETS
PUBLIC area, exact, safety, deci
DO setup
```

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).

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

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

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

```
currarea = SELECT( )

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

```
*       ************************************************************
*       *              CONVERT Setup Code - SECTION 2              *
*       ************************************************************
#REGION 1
PRIVATE hidecomm, unittype, i, size, fromarry, toarry

SET DECIMALS TO 18
                .
                .
                .
size = ALEN(fromarry)
DIMENSION toarry[size]
FOR i = 1 TO size
        toarry[i] = fromarry[i]
ENDFOR

frompop = fromarry[1]
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.

```
DEFINE WINDOW convert ;
        FROM INT((SROW( )-17)/2),INT((SCOL( )-53)/2) ;
        TO INT((SROW( )-17)/2)+16,INT((SCOL( )-53)/2)+52 ;
        FLOAT ;
        NOCLOSE ;
        SHADOW ;
        DOUBLE ;
        COLOR SCHEME 5
```

Define window commands — 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 Screen Layout                 *
*       ************************************************************
#REGION 1
ACTIVATE WINDOW convert NOSHOW
@ 9,28 TO 14,49
@ 9,1 TO 14,22
@ 9,3 SAY " From: "
@ 9,30 SAY " To: "
@ 11,4 GET frompop ;
       PICTURE"@^" ;
       FROM fromarry
       SIZE 3,16  ;
       DEFAULT 1 ;
       VALID _puu17o66o
              .
              .
              .
@ 1,17 GET unittype ;
       PICTURE "@*RVN \<Area;\<Length;\<Mass;\<Speed;Tempe\<rature;\<Time;\<Volume" ;
       SIZE 1,15,0 ;
       DEFAULT 1 ;
       VALID _puy0nhdyn( )
@ 1,8 SAY " Type: "

ACTIVATE WINDOW convert
READ CYCLE ;
       DEACTIVATE _puy0nhet4( )

RELEASE WINDOW convert

#REGION 0
IF USED("units")
       SELECT units
       USE
ENDIF
IF USED("factors")
       SELECT factors
       USE
ENDIF
SELECT (currarea)

IF talkstat = "ON"
       SET TALK ON
ENDIF
IF compstat = "ON"
       SET COMPATIBLE ON
ENDIF
```

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 is generated with generated name or a user-defined name provided in the Screen Layout dialog.

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

RELEASE WINDOW commands — RELEASE WINDOW commands are generated only when Release Windows is checked in the Generate Screen dialog. You can suppress the generation of these commands by unchecking this check box.

Close file commands — These commands are generate 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.

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 NOT EMPTY(oldhelp)
       SET HELP TO LOCFILE (oldhelp, "DBF", "Where is "+oldhelp+" help file?")
ENDIF
SET UDFPARMS TO VALUE

IF hidecomm
       SHOW WINDOW command
ENDIF

RELEASE oldhelp, fromarry, toarry, skipvar
POP MENU _MSYSMENU
DO cleanup
```

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.

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

IF topop = frompop
       new = old
       SHOW GETS
       RETURN
ENDIF          .
               .
               .
new = stripzeros(new)
old = stripzeros(old)
SHOW GETS      .
               .
               .
```

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

```
*       ************************************************************
*       * _PUU170660           frompop VALID                       *
*       *                                                         *
*       * Function Origin:                                        *
*       *                                                         *
*       * From Screen:         CONVERT,     Record Number:    9   *
*       * Variable:            frompop                            *
*       * Called By:           VALID Clause                       *
*       * Object Type:         Popup                              *
*       * Snippet Number:      1                                  *
*       ************************************************************
*
FUNCTION _puy0nhazm     &&  frompop VALID
#REGION 1
IF EMPTY(fromval)
        _CUROBJ = OBJNUM(fromval)
        SHOW GET fromval
        RETURN .F.
ENDIF
= convrt (fromval, toval, "left")
                .
                .
                .
```

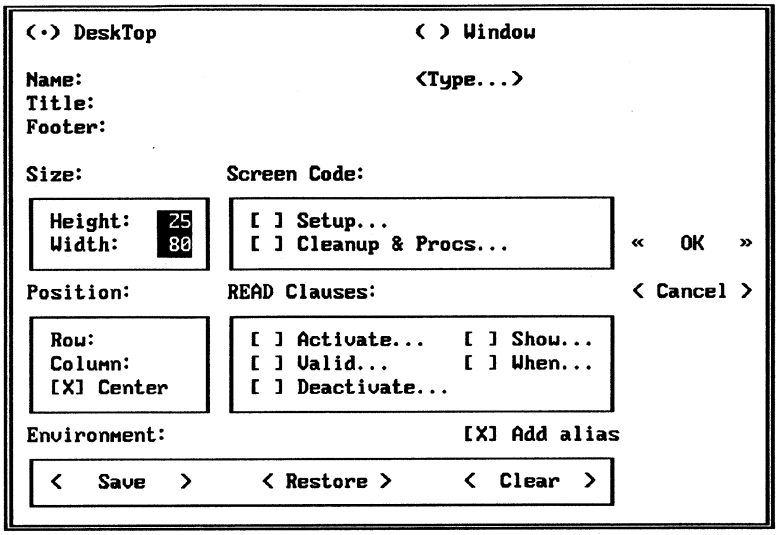
Object level clause

```
*
         ************************************************************
 *       *                                                         *
 *       * _PV60NIAGH           Read Level When                    *
 *       *                                                         *
 *       * Function Origin:                                        *
 *       * From Screen:         CLIENTS                            *
 *       * Called By:           READ Statement                     *
 *       * Snippet Number:      19                                 *
 *       ************************************************************
 *
 FUNCTION _pv60niagh     && Read Level When
 *
 * When Code from screen: CLIENTS
 *
 #REGION 0
 DO mainmenu.mpr
```

READ level clause

Screen Layout

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



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.

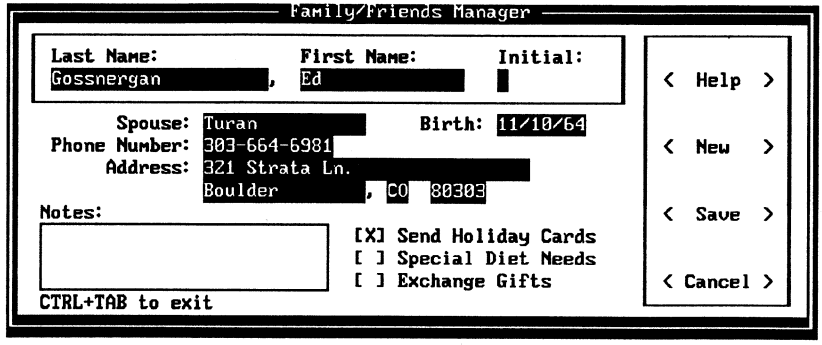
Setup Code

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 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.

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.




```
PRIVATE hidecomm, adding
PUSH MENU _MSYSMENU

adding = .F.
hidecomm = WVISIBLE("command")
IF hidecomm
      HIDE WINDOW command
ENDIF
```

Make variables PRIVATE

Push menu system on stack (new menu system is installed in WHEN clause code snippet.)

Define and initialize memory variable

Hide command window

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 *Commands & Functions* manual.

Setup Code Example 2

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


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

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

Generator directive (described in next section)

Make variables REGIONAL

Initialize variable for "OK" push button

Initialize variables with top, bottom and current RECNO( )

Return record pointer to current record

Generator Directives

Generator directives can be placed in the setup code to divide the setup code into two sections, define code that is concatenated on the READ command, or to specify a character allowing you to control whether picture clauses, window titles or window footers are expressions or string literals. A generator directive is a command that communicates solely with GENSCRN, the screen generator. Generator directives do not appear in the generated code.

Code located in Setup Code — Section 1 is generated and executed at the beginning of the .SPR program. Parameter statements and ON ERROR statements are examples of code you would want to place in Setup Code — Section 1.

The second section of the setup code is generated and executed before the screen layout commands. For information on the sequence of the generated program, see the section titled The Generated Program earlier in this chapter.

You must precede commands to be inserted in section one of the setup code with a #SECTION 1 generator directive. If you would like other setup code placed in Setup Code — Section 2, a #SECTION 2 generator directive should follow the commands in section 1. The generator directives must be placed in the first column. An example follows:

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

If you have #SECTION generator directives in the setup code for multiple screens in a screen set, all the code between the #SECTION 1 and #SECTION 2 generator directives is concatenated in Setup Code — Section 1 segment of the generated program. All code that follows the #SECTION 2 generator directive is concatenated in the Setup Code — Section 2 segment of the generated program.

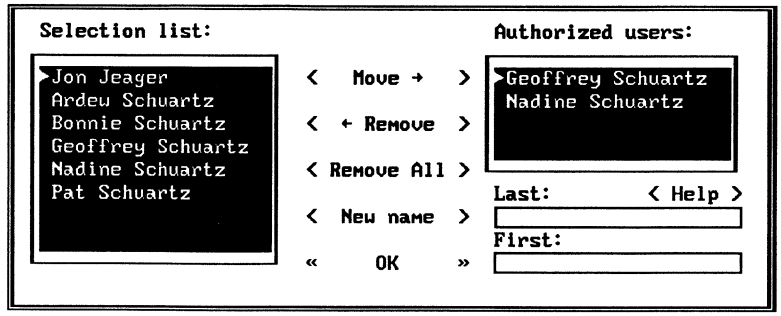
Other generator directives and the actions they perform are:

#READCLAUSES <clauses> — Specified on a single line, <clauses> are placed at the end of the READ command in a screen program and may include clauses such as TIMEOUT, SAVE, etc.

#ITSEXPRESSION <char> — This is a single character that allows you to control whether picture clauses, window titles or window footers are expressions or string literals.

Setup Code Example 3

This example is from ADDUSERS.SCX, a screen is 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.




```
#SECTION 1
PARAMETER cardid
```

Generator directive

PARAMETER statement

```
#SECTION2
```

Generator directive

```
PRIVATE mover, user, allcnt, saverec, usrcnt, limit, ;
      allusers, status, savearea, userlast, userfirst
```

Make variables PRIVATE

```
SET EXACT ON
```

Environment setting

```
user = 1
userlast = ""
status = .T.
savearea = SELECT( )
DIMENSION allusers[1,3]
allusers = ""
```

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

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

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

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

Fill array using SQL SELECT

```
allcnt = ALEN(allusers,1)

IF EMPTY(users)
      usrcnt = 0
ELSE
      usrcnt = 1
      limit = ALEN(users,1)
      DO WHILE usrcnt <= limit
            IF EMPTY(users[usrcnt,1])
                  EXIT
            ENDIF
            usrcnt = usrcnt + 1
      ENDDO
      usrcnt = usrcnt - 1
ENDIF
```

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

Check size of Authorized Users List so 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 commands in the Setup Code — Section 2 code snippet. 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 NOT EMPTY(oldhelp)
      SET HELP TO LOCFILE (oldhelp, "DBF",;
            "Where is "+oldhelp+" help file?")
ENDIF

IF hidecomm
      SHOW WINDOW command
ENDIF

RELEASE oldhelp
POP MENU _MSYSMENU
```

Return to previous Help (if any)

Show Command window

RELEASE variables

Restore previous menu system

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.

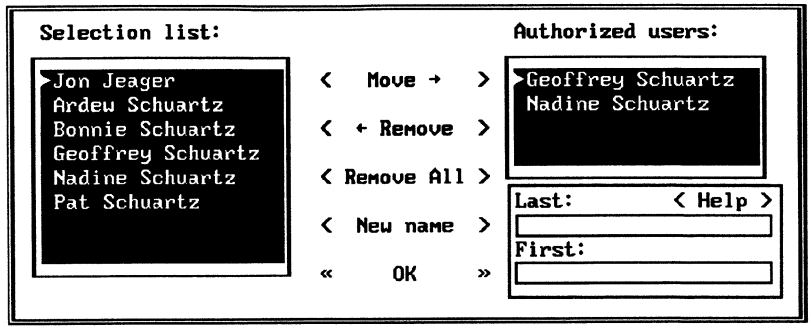
When you define a procedure in the code snippet, the generator assigns the procedure a unique name that changes every time the screen is regenerated. The unique name is inserted in a FUNCTION command and the code in the code snippet follows.

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 that .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 WHEN clause code snippet contains the following code:

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

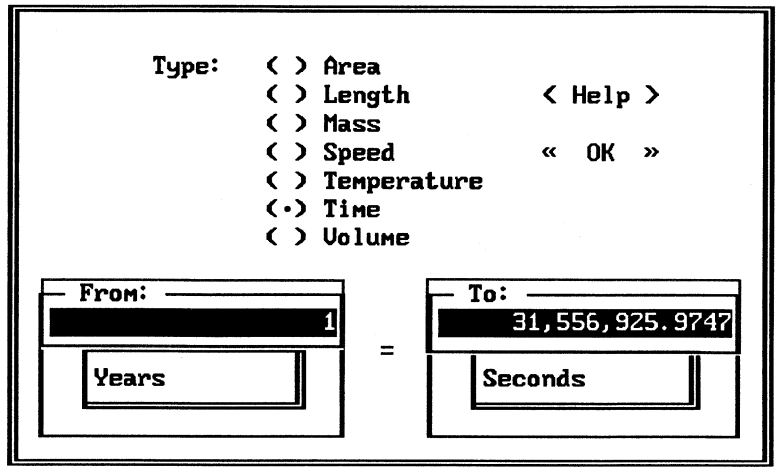
This procedure is defined in the cleanup code for the screen and contains the following code:

```
*
* ESCHANDLER
*
PROCEDURE eschandler
ON KEY LABEL esc
username  = SPACE(22)
userfirst = SPACE(14)
SHOW GET username DISABLE
SHOW GET 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 To and From GET fields use an expression to call a UDF.



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 new, old, direction
PRIVATE toid, fromid, tounit, fromunit, factor

      .
      .
      .
```

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 on incorporating context-sensitive help in you 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.

Following is a list of the order that READ events and clauses are executed when the READ is first issued:

- READ level WHEN clause
- ACTIVATE WINDOW
- READ level ACTIVATE
- READ level SHOW[1]
- 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

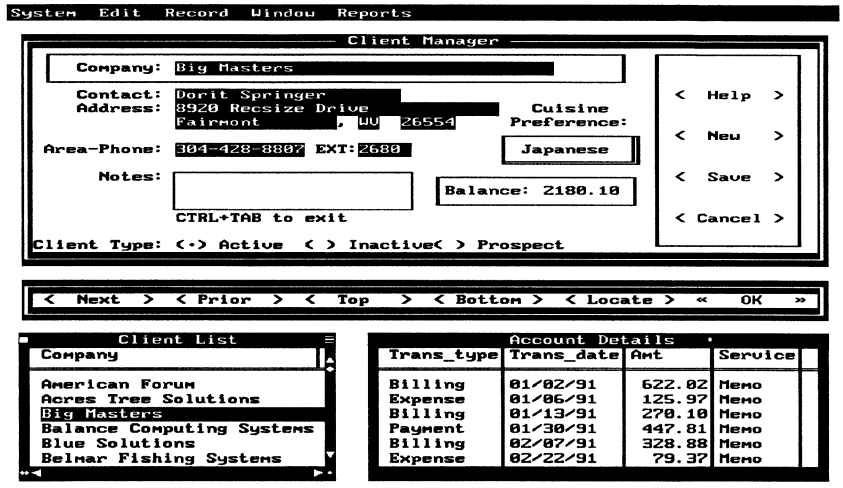
[1] The SHOW clause is also executed whenever the SHOW GETS command is issued. The SHOW GETS command can be issued in any code snippet in the screen.

[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 on coordinating Browse windows with screens, see the section titled Coordinating Screens with Browse later in this chapter.



```
IF UPPER(WLAST()) = 'ACCOUNT DETAILS'
      SELECT clients
ENDIF
```

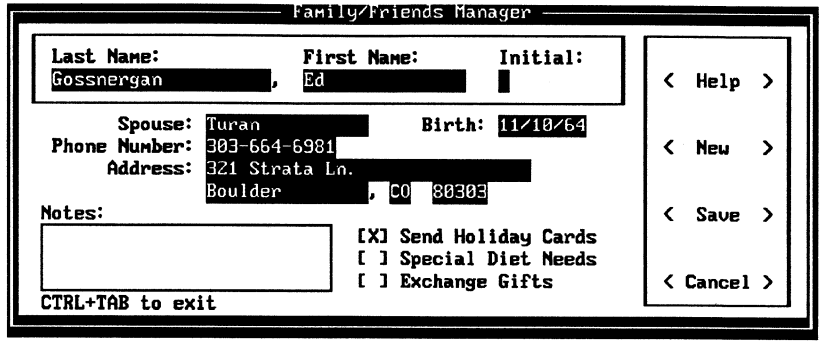
If Browse window was the last window on top, reselect the CLIENTS database because activating the Browse window automatically selects the associated database.

```
IF NOT WEXIST('Account')
      SELECT details
      BROWSE NORMAL NOWAIT LAST
      SELECT clients
ENDIF

IF NOT WEXIST('Client')
      BROWSE NORMAL NOWAIT LAST
ENDIF
```

If a Browse window is closed, reopen it.

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.




```
IF NOT adding
      SCATTER MEMVAR
      m.notes = Notes
ENDIF
```

Check if this is a new record

Create variables for fields and move data from current record into variables. Because SCATTER does not create variables for memo fields, a variable is created for the memo field.

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 any code snippet in the screen.

SHOW GETS redisplays 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.


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

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
      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
            SHOW GET m.choice ENABLE
      ENDIF
ENDIF
```

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

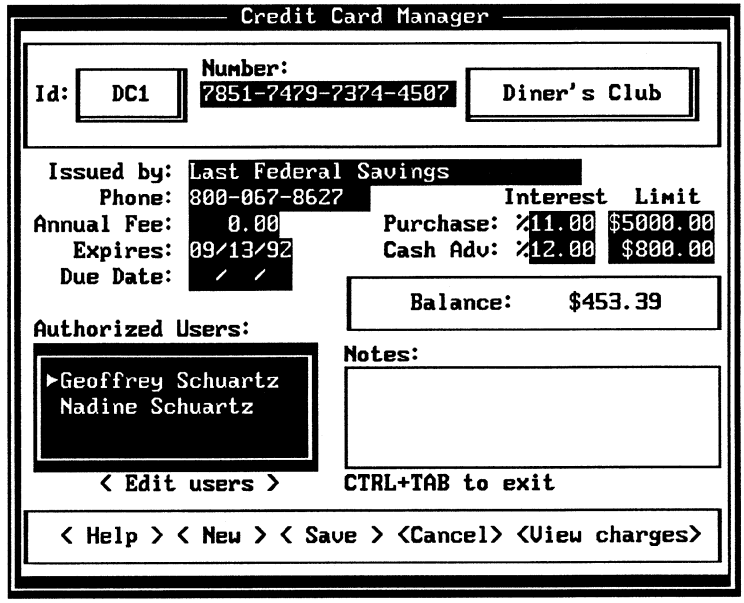
Enable/Disable buttons if record pointer is positioned on bottom record.

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

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.




```
IF NOT adding
      SCATTER MEMVAR
      m.notes = Notes
      cards = Type
```

Check if this is a new record.

SCATTER fields and create variables for memo fields. (See READ Level SHOW Example 1)

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

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

```
            FOR i = 1 TO ALEN(users,1)
                  users[i,3] = ALLTRIM(users[i,2])+;
                        " "+ALLTRIM(users[i,1])
            ENDFOR
            SELECT credcard
      ELSE
            SELECT credcard
            users = ""
            SHOW GET user
      ENDIF
ELSE
      SHOW GET user
ENDIF
```

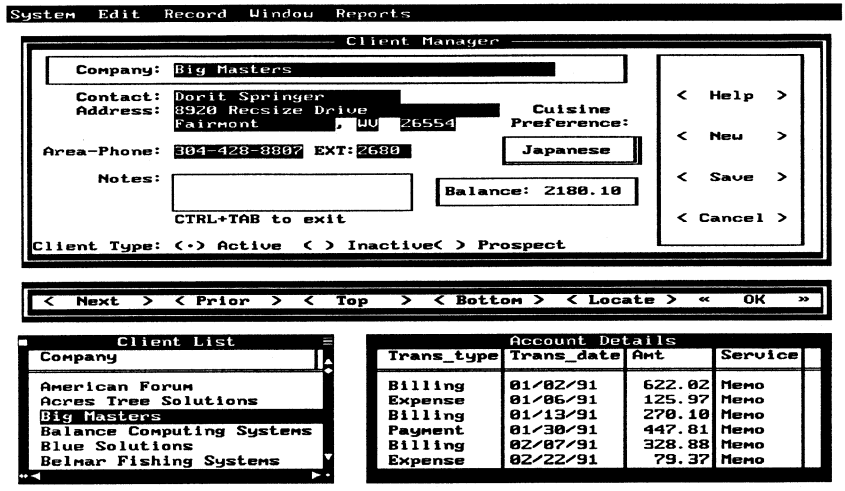
If there are no card users, blank the array (users = "") and refresh the display.

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.

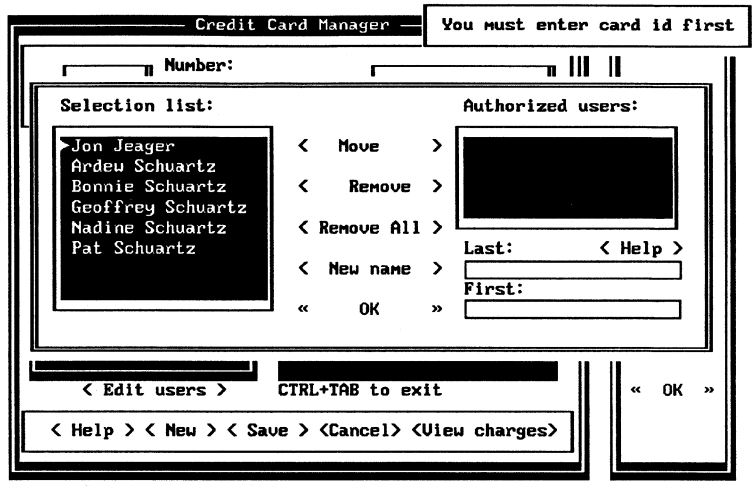
In this example, the WHEN routine is used to install the menu system associated with the Client Manager screen. This allows the menu system to be reactivated whenever the user is in a screen with GET fields.




```
#REGION 0
 DO mainmenu.mpr  <-------------  Install menu system.
```

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.




```
IF empty(cardid)
      WAIT WINDOW "You must enter card id first" NOWAIT
      status = .F.
      RETURN .F.
ENDIF
```

Field Objects and Controls

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

Defining field objects and controls is described in the FoxPro *Interface 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.

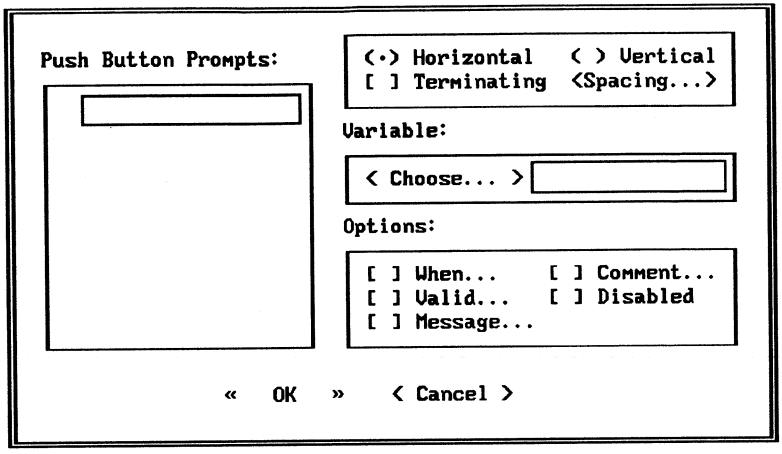
ERROR Clause The ERROR clause is executed when the cursor exits a GET or EDIT field with invalid data (as defined in a VALID clause). The ERROR clause is available for GET and EDIT fields only.

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

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.



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.

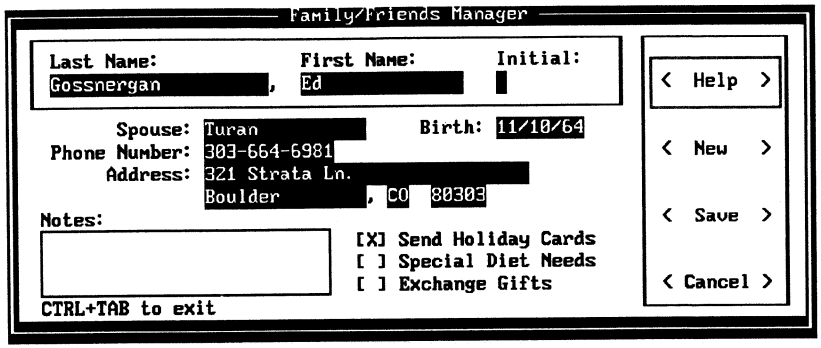
Terminating push buttons execute the VALID (if one is defined), exit the READ and execute the next line in the controlling program.

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 *Interface Guide* or the @ ... GET — Push Buttons command in the FoxPro *Commands & Functions* manual.

Push Button VALID Clause Example 1

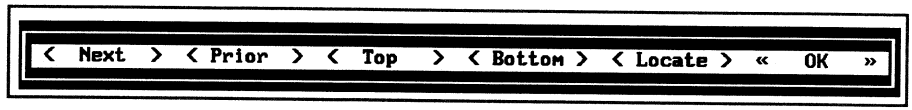
This example of an individual push button is taken from FAMILY.SCX, a screen called in the Family & Friends module of the ORGANIZER application. The VALID clause is defined for the **Help** push button and is used to bring forward a Help window for the Family & Friends module.




```
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.



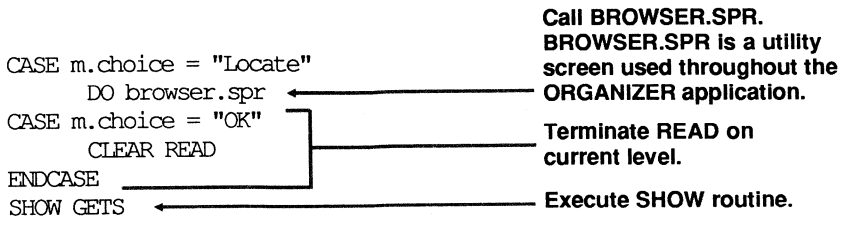

```
DO CASE
CASE m.choice = "Next"
      SKIP 1
      IF RECNO( ) = m.bottomrec
            SHOW GET m.choice, 1 DISABLE
            SHOW GET m.choice, 4 DISABLE
      ELSE
            IF RECNO( ) > m.toprec
                  SHOW GET m.choice, 2 ENABLE
                  SHOW GET m.choice, 3 ENABLE
            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
CASE m.choice = "Bottom"
      GO BOTTOM
      SHOW GET m.choice, 1 DISABLE
      SHOW GET m.choice, 2 ENABLE
      SHOW GET m.choice, 3 ENABLE
      SHOW GET m.choice, 4 DISABLE
```

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

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

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

Executed when user chooses Bottom push button. Moves record pointer to bottom record in database and enables/disables buttons.

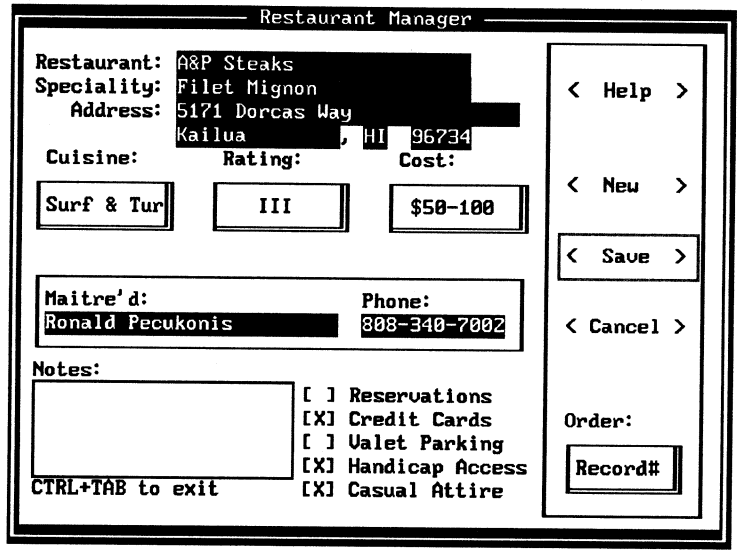


Variables for push buttons can be of character or numeric type. In this example, the variable m.choice is defined as a character variable in the setup code for the screen. This allows you to rearrange the push buttons in the screen without modifying the code snippet.

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 WHEN clause is defined for the **Save** push button and is used to display a message if the user chooses the Save push button without entering a restaurant name.




```
IF EMPTY(m.restaurant)
      ?? CHR(7)
      WAIT WINDOW "Enter restaurant name" NOWAIT
      _CUROBJ = OBJNUM(m.restaurant)
      RETURN .F.
ENDIF
```

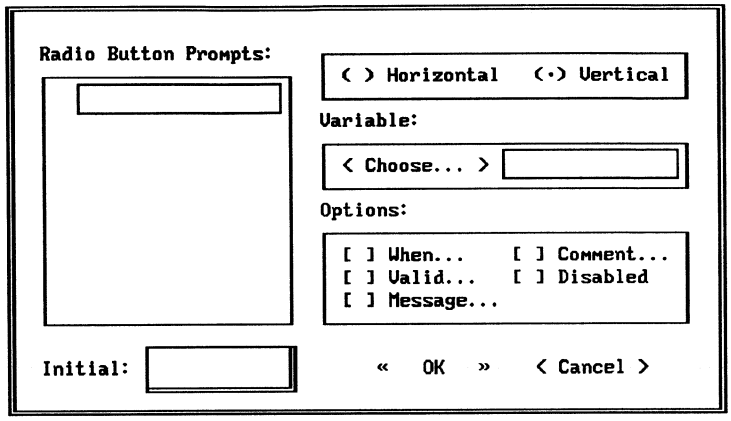
Check to see if Restaurant field is empty.

Display message and ring bell if user chooses Save push button without entering 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.

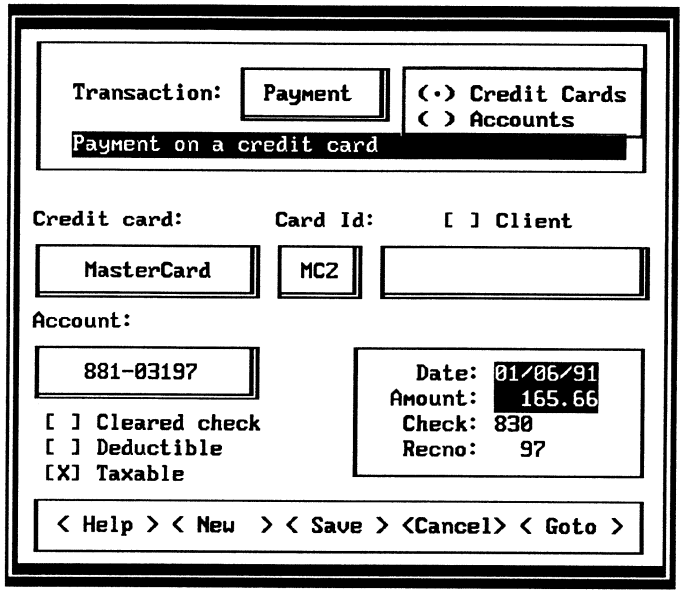


Radio Button Dialog

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.



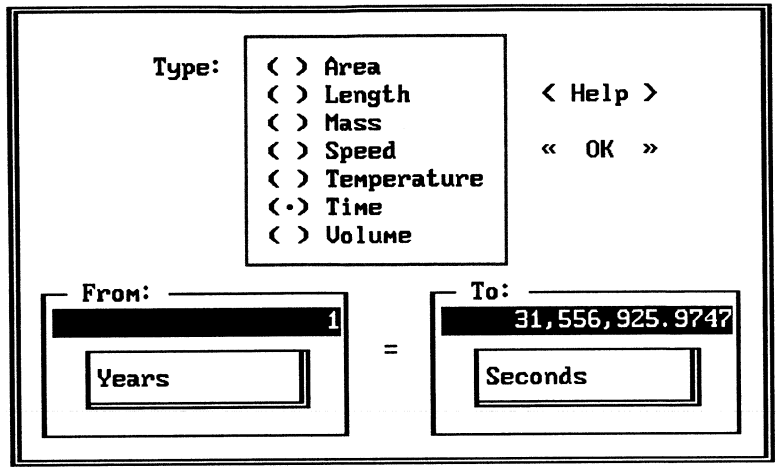

```
DO CASE
CASE decider = 1
      SHOW GET m.cards ENABLE
      SHOW GET m.card_id ENABLE
      SHOW GET m.acct_id DISABLE
      SHOW GET m.cleared DISABLE
      SHOW GET m.check_no DISABLE
CASE decider = 2
      SHOW GET m.cards DISABLE
      SHOW GET m.card_id DISABLE
      SHOW GET m.acct_id ENABLE
      SHOW GET m.cleared ENABLE
      SHOW GET m.check_no ENABLE
ENDCASE
```

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

Enable/disable controls when Accounts radio button is selected.

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.




```
PRIVATE i, size
```
Make variables PRIVATE.

```
SELECT DISTINCT units.unit;
      FROM units;
      WHERE units.type = unittype;
      ORDER BY units.type;
      INTO ARRAY fromarry
```
Create, sort and fill using a SQL SELECT statement.

```
size = ALEN(fromarry)
```
Determine length of array1.

```
DIMENSION toarry[size]
```
Create array2.

```
FOR i = 1 TO size
      toarry[i] = fromarry[i]
ENDFOR
```
Copy elements from array1 to array2

```
frompop = fromarry[1]
topop   = toarry[1]
fromval = SPACE(19)
toval   = SPACE(19)
```
Initialize variables for popups and GET fields.

```
_CUROBJ = OBJNUM(fromval)
```
Make From GET field the current object.

```
SHOW GETS
```
Execute SHOW routine.

Initial Popup

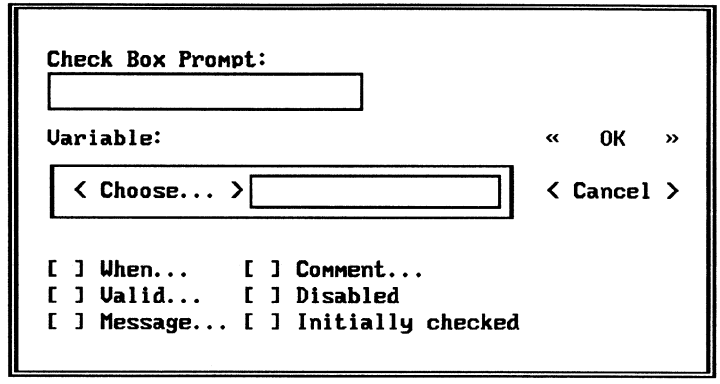
The initial selection is used only for the creation of a variable. By default, the variable is initialized to "1", the number that corresponds to the first prompt.

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.

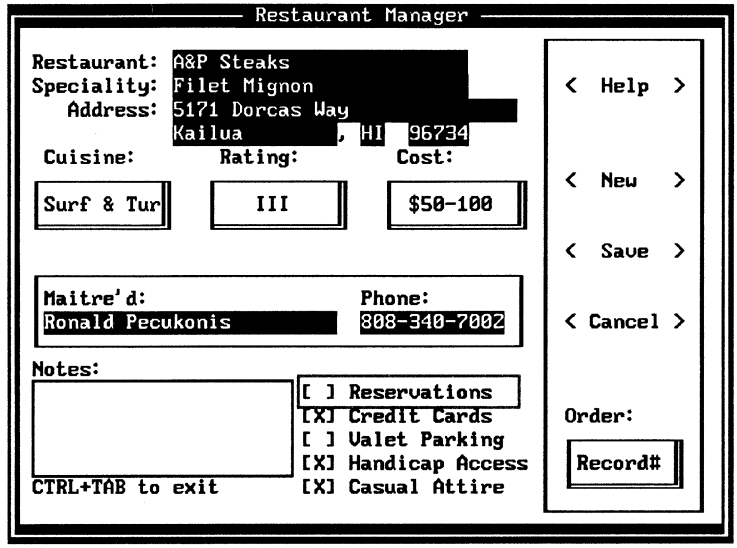
Choose **Check Box...** on the **Screen** menu popup to bring forward the Check Box dialog.



Check Box Dialog

Check Box VALID Clause Example 1

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 **Reservations** 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.

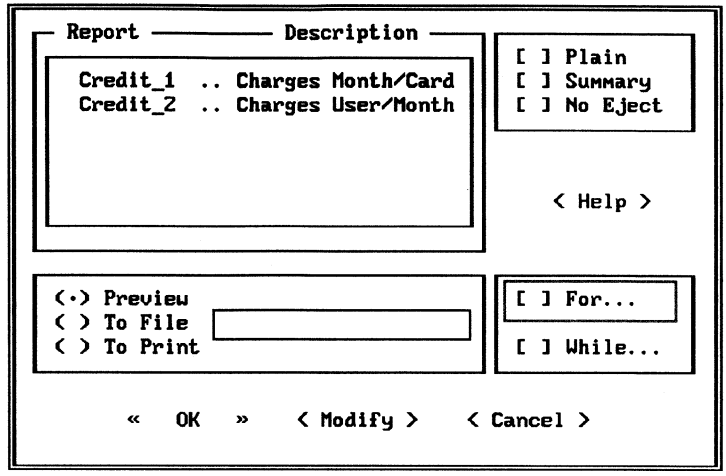



```
IF m.reserve < > restaurs.reserve
      = showsave( )
ENDIF
```

Check to see if value has changed. If so, enable 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.




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

Display Expression Builder allowing user to enter a FOR expression.

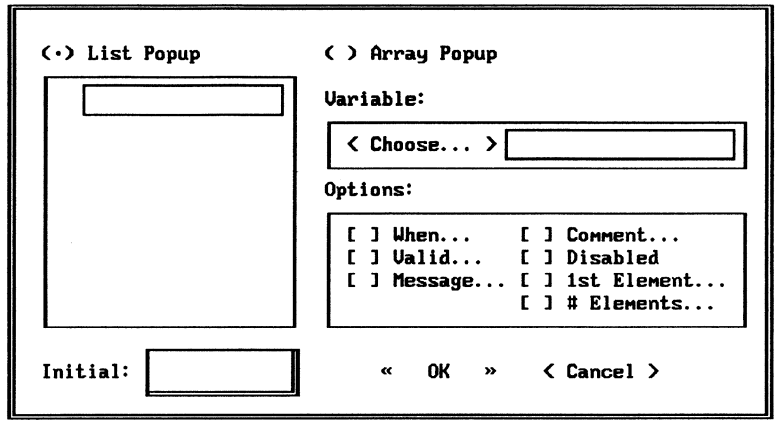
```
for = IIF(EMPTY(forexpr), 0, 1)
SHOW GET for
```

Check or uncheck 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.



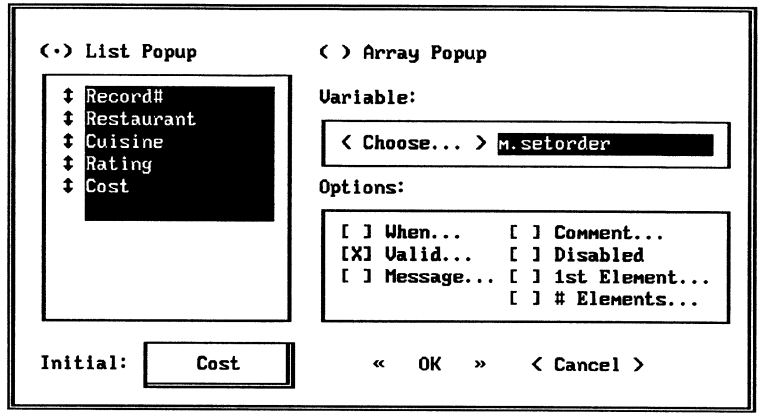
Popup Dialog

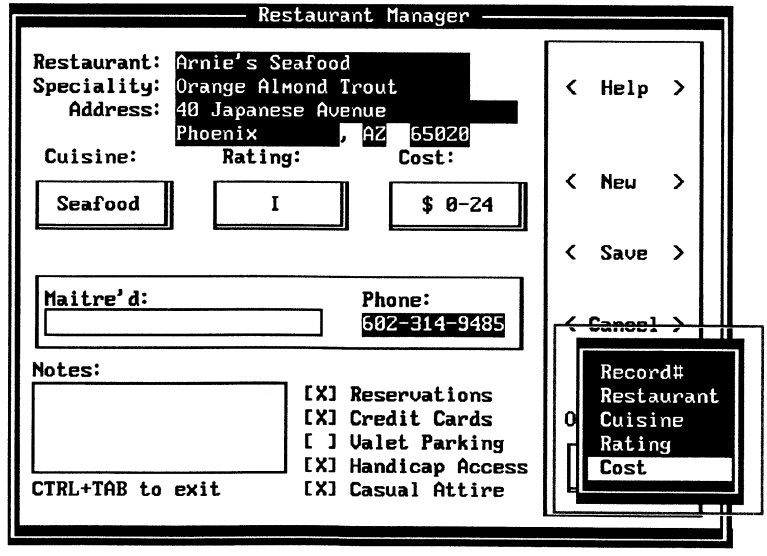
With an array popup, you DIMENSION the array in your setup code (described earlier in this chapter). The elements in the array can be defined in a variety of ways. For information on arrays, see the Arrays chapter in this manual.

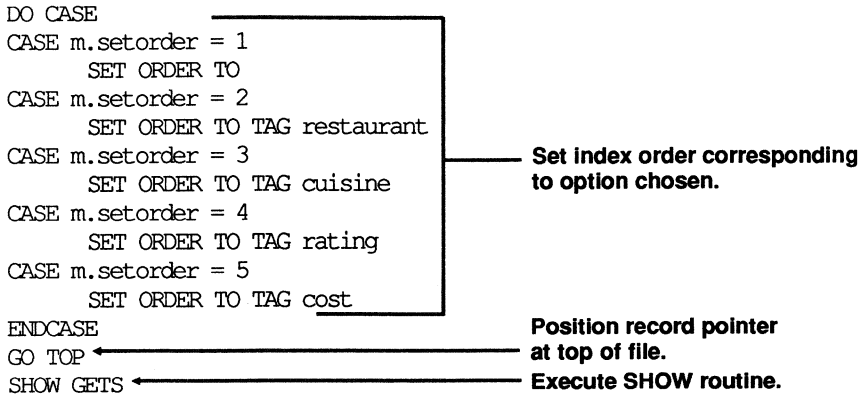
Popup Example 1

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 RESTRAUS.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.

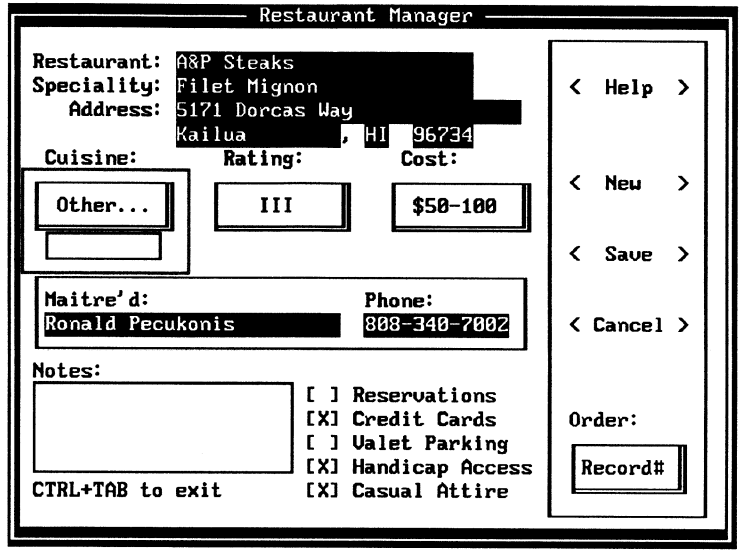





DO CASE
CASE m.setorder = 1
SET ORDER TO
CASE m.setorder = 2
SET ORDER TO TAG restaurant
CASE m.setorder = 3
SET ORDER TO TAG cuisine
CASE m.setorder = 4
SET ORDER TO TAG rating
CASE m.setorder = 5
SET ORDER TO TAG cost
ENDCASE
GO TOP
SHOW GETS
Set index order corresponding to option chosen.
Position record pointer at top of file.
Execute SHOW routine.

Popup Example 2

This example is taken from RESTRAUS.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.



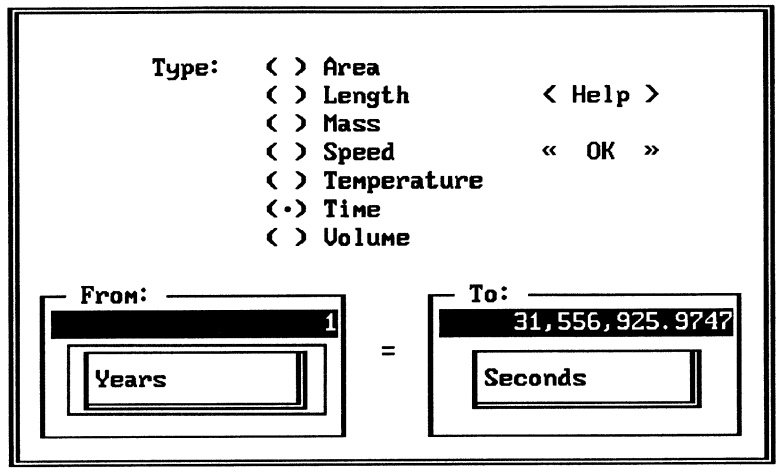

```
IF m.cuisine = "Other..."
      popupedit = .T.
      SHOW GET newcuis ENABLE
      _CUROBJ = OBJNUM(newcuis)
ENDIF
IF m.cuisine < > restaurs.cuisine
      = showsave( )
ENDIF
```

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

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

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.



Setup Code

```
#SECTION 2
PRIVATE hidecomm, unittype, i, size, fromarry, toarry
        .
        .
SELECT DISTINCT units.unit;
      FROM factors, units;
      WHERE units.id = factors.to;
      AND factors.type = "Area";
      INTO ARRAY fromarry

= ASORT(fromarry)

size = ALEN(fromarry)
```

Generator directive.

Make variables PRIVATE.

Create and fill array using a SQL SELECT statement.

Sort array.

Store length of array to variable.

VALID Clause

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

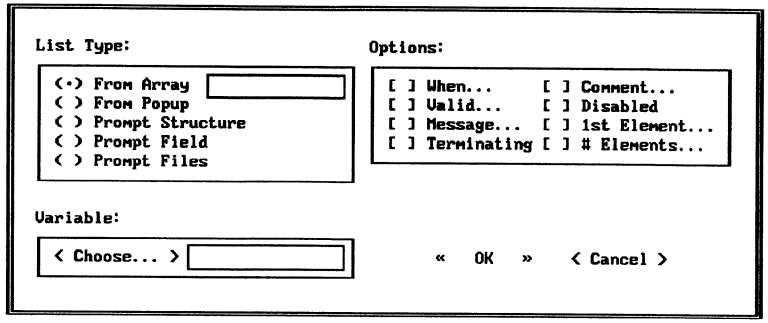
Force user to enter value in From GET field.

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 he user can scroll through a many options.

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

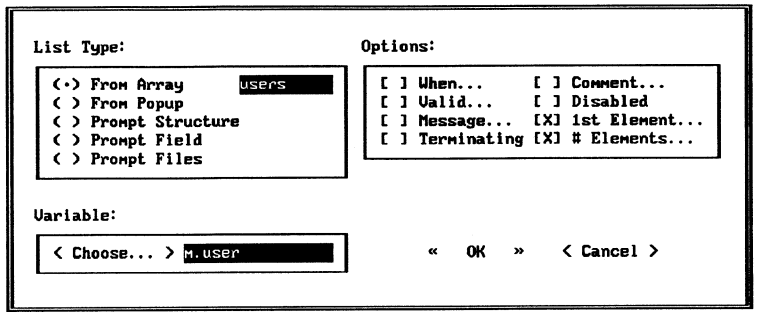


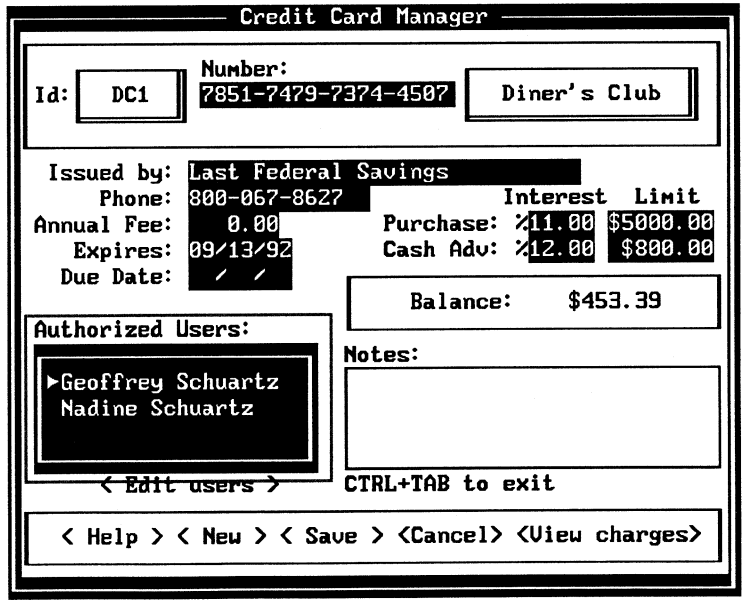
List Dialog

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.





#Elements

```
PRIVATE cnt, limit
cnt = 1
limit = ALEN(users,1)
DO WHILE cnt <= limit
      IF EMPTY(users[cnt,1])
            RETURN cnt-1
      ENDIF
      cnt = cnt + 1
ENDDO
RETURN cnt-1
```

Make variables PRIVATE.

Initialize variable.

Store number of rows to variable.

Prevent display of blank records in list.

1st Element

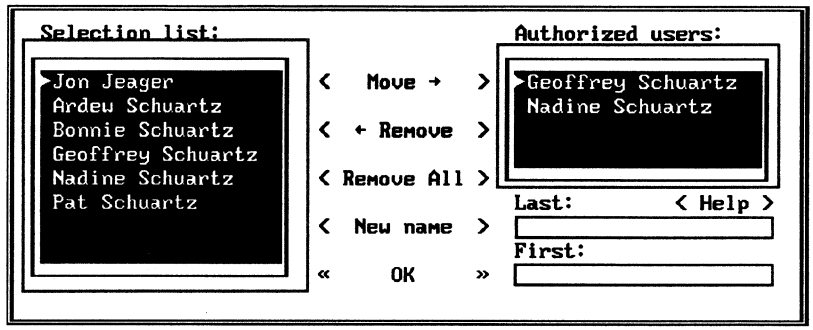
```
3
```

Display the third column in array.

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.



Selection List VALID

```
IF alreadyin(allusers[m.alluser,3])
      WAIT WINDOW "Duplicate entry" NOWAIT
      RETURN .F.
ENDIF
IF usrcnt+1 > ALEN(users,1)
      DIMENSION users[usrcnt+1,3]
ENDIF

users[usrcnt+1,1] = allusers[m.alluser,1]
users[usrcnt+1,2] = allusers[m.alluser,2]
users[usrcnt+1,3] = allusers[m.alluser,3]
usrcnt = usrcnt + 1
user = usrcnt
SHOW GET mover, 2 ENABLE
IF usrcnt > 1
      SHOW GET mover, 3 ENABLE
ENDIF
SHOW GET user
```

Compare Selection list with Authorized users list to prevent duplication.

Add row to array (if necessary).

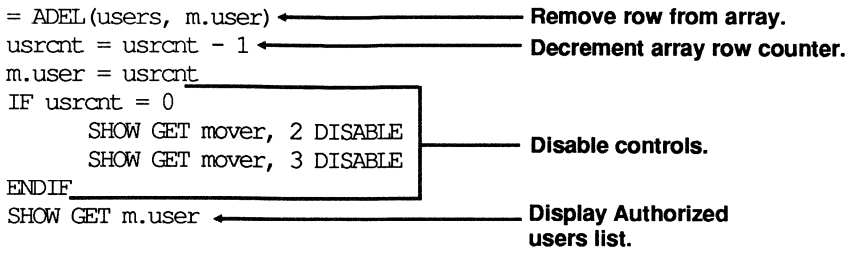
Copy selected name to Authorized users list.

Increment array row counter.

Enable controls.

Display Authorized users list.

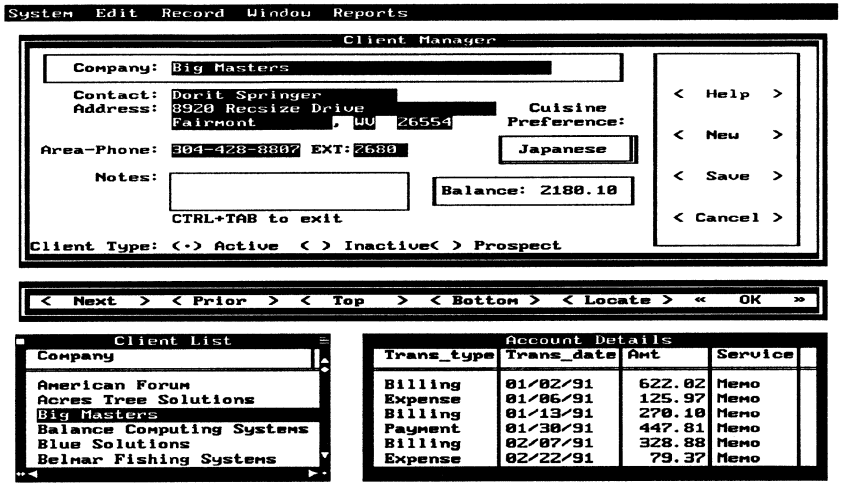
Authorized Users VALID



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.



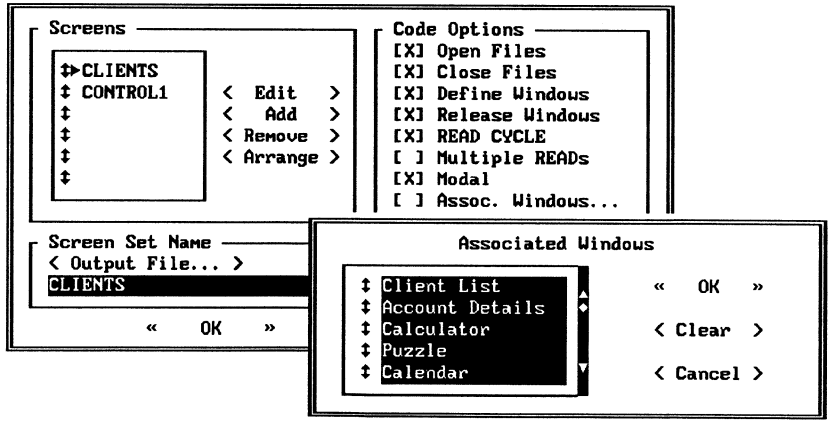
This section describes how these Browse windows are defined and activated, and offers tips on using Browse with screens.

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:

```
BROWSE NORMAL NOWAIT LAST TITLE 'Client List' ;
      NOEDIT NOMENU FIELDS company ;
      WHEN showgets()
      VALID clie_valid( )

SELECT details
BROWSE NORMAL NOWAIT LAST TITLE 'Account Details' ;
      NOAPPEND NOEDIT NOMENU ;
      FIELDS Trans_type:10, Trans_date:10, Amt:7, Service
SELECT clients
```

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 _pv60y7dco( ) ;
      ACTIVATE _pv60y7dcu( ) ;
      SHOW _pv60y7dd0( ) ;
      WITH Client, Account, Calculator, ;
            Puzzle, Calendar, 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:

```
resoset = SET("RESOURCE")
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 save in the resource file.

The following commands are included in the cleanup code for CLIENTS.SCX restore the previous resource file or allow you to choose another resource file:

```
IF NOT EMPTY(oldreso)
      SET RESO TO LOCFILE(oldreso, "DBF", ;
            "Where is "+oldreso+" resource file?")
ENDIF

IF 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 section in the FoxPro *Interface 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.

```
*       ************************************************************
*       * _PUU170660           frompop VALID                       *
*       *                                                         *
*       * Function Origin:                                        *
*       *                                                         *
*       * From Screen:         CONVERT,      Record Number:    9  *
*       * Variable:            frompop                           *
*       * Called By:           VALID Clause                      *
*       * Object Type:         Popup                             *
*       * Snippet Number:      1                                 *
*       ************************************************************
*
FUNCTION _puy0nhazm     &&  frompop VALID
#REGION 1
IF EMPTY(fromval)
       _CUROBJ = OBJNUM(fromval)
       SHOW GET fromval
       RETURN .F.
ENDIF
= convrt (fromval, 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 FoxPro power tool for creating menus. In the 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.

You can also create code snippets for a menu. A code snippet can be executed before your menu is activated, after your menu is activated or when you make a choice from the menu. Because code snippets are stored with the menu, a menu is actually a form of source code and can be included in a project.

The Menu Builder eliminates the most tedious step in creating menus — writing the menu program code. After you've interactively designed your menu, FoxPro generates the corresponding menu program code for you. The Project Manager can integrate menus, screens, programs, queries, reports, labels and more into a single executable FoxPro application.

This chapter assumes that you are familiar with the FoxPro interface and the Menu Builder. If you are not familiar with these, consult the Interface Basics and Menu Builder chapters in the FoxPro *Interface Guide*.

The menus used as examples in this chapter are from the ORGANIZER sample application included with FoxPro. The ORGANIZER menus used in the examples in this chapter are MAINMENU.MNX and CONVERT.MNX. You can open and examine these menu files in the Menu Builder, and you can 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 a menu is created in the Menu Builder, the procedures that are executed when a choice is made from a menu become an integral part of the menu.

Increased Productivity

Designing a menu interactively in the Menu Builder is much faster than writing a program to create a menu. After you've interactively designed your menu, FoxPro can generate the corresponding menu program code for you. At any time while you're creating a menu in the Menu Builder you can choose the **Try It** option to test the appearance and operation of your menu.

FoxPro's System Menu Bar

The Menu Builder automatically utilizes FoxPro's system menu bar. There are numerous advantages in utilizing FoxPro's system menu bar:

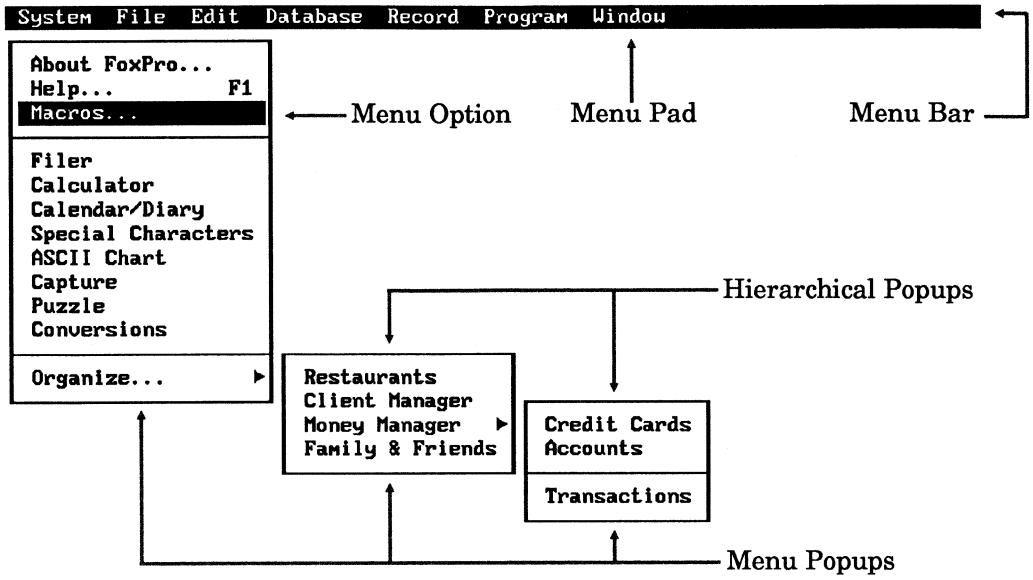
- The system menu bar is automatically activated and deactivated in your application as necessary. Whenever your application is waiting for keyboard input (during a READ, in the interactive mode, etc.), the system menu bar is accessible.

- You can include options from FoxPro's system menu popups in your popups. FoxPro's system options are automatically enabled and disabled as needed.

- It isn't necessary to explicitly ACTIVATE your menu.

Terms Used in this Chapter

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



Menu Components

Menu Bar

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

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

Menu Pads

The names on the menu bar are *menu pads*. When a menu pad is chosen, a menu popup is displayed below the menu pad, or a procedure or command is executed.

In the Menu Builder, menu pads are created in the initial 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.

Menu popups are created for a menu pad by choosing the **Submenu** option from the **Result** popup in the Menu Builder.

Menu Options

A menu popup contains a set of related *options*. When you choose an option from a menu popup, a procedure or command is executed or another menu popup, called a *hierarchical popup*, is displayed.

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

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

Code Snippets

When you design a menu in the Menu Builder, you can create *code snippets* that are assigned to menu pads, popups or options. Code snippets are a set of FoxPro commands or functions that are contained in a *procedure*. A procedure is executed when a menu pad or option is chosen from your menu.

When a code snippet is one line long, the code snippet is included with the command that executes a code snippet. This command is typically ON SELECTION PAD or ON SELECTION BAR. The command with its code snippet is placed in the menu definition code.

If a procedure's code snippets are longer than one line, the snippets are placed in a separate procedure that is assigned a unique name created by the menu code generator with SYS(2015).

The command that executes the procedure (ON SELECTION PAD, ON SELECTION BAR, ON SELECTION POPUP, etc.) calls the procedure by its generated name. This procedure is placed with other procedures at the end of the generated menu program.

> The unique names are created for procedures when the menu program is generated. If you make changes to a menu and regenerate the menu program, the generated procedure names will be different.

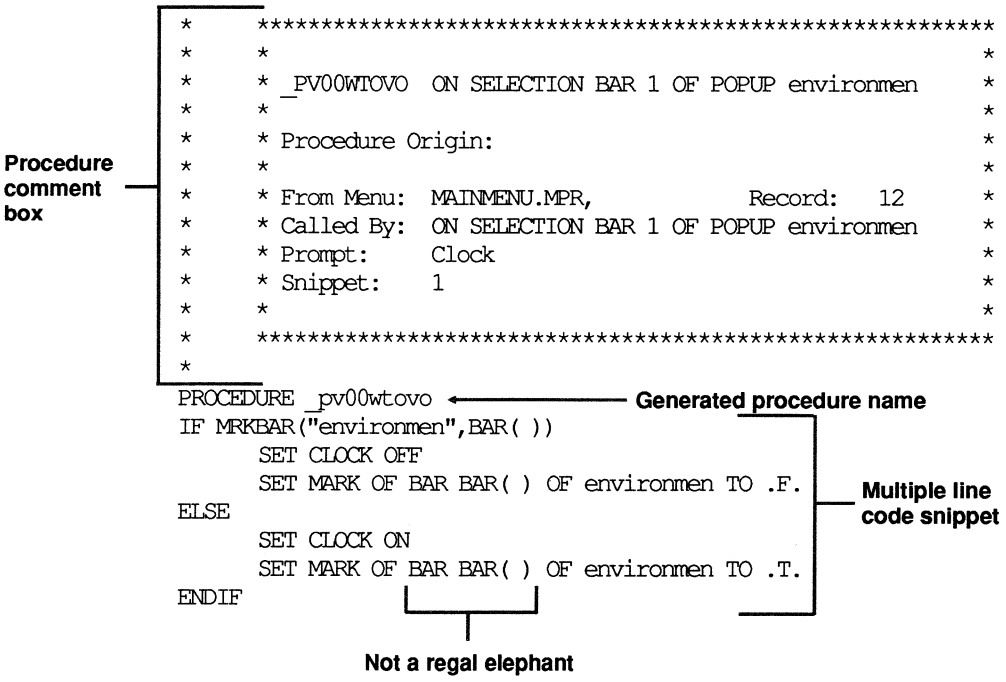
In the generated menu code, procedures are preceded by a comment box that documents the procedure. Comment boxes ease debugging and provide information for FoxDoc, the program documentor included with FoxPro.

The comment box describes:

- The procedure name
- The command that calls the procedure
- The number of the record in the menu's .MNX database that contains the procedure
- The prompt of the menu pad or popup option that calls the procedure
- The snippet number — each procedure in the generated code is numbered

Following is the code snippet for the Clock option on the System popup in MAINMENU. Because the code snippet is longer than one line, a procedure is created. The procedure is assigned the generated name _PV00WTOVO. The procedure comment box provides information about the origin of 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
```



Calling a Menu Program

When you call a menu program, use the following syntax:

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

You must include the .MPR extension. Because different types of executable files (menus, screens, queries, etc.) may have the same name, you must include the .MPR extension to execute a menu.

FoxPro executes the compiled .MPX version of the menu program. If the compiled .MPX version of the menu program cannot be located, the source .MPR menu program is automatically compiled, creating the compiled .MPX version.

Activating the Menu

Menus created in the Menu Builder automatically utilize the FoxPro menu system. If you create a menu that utilizes the FoxPro menu system, it isn't necessary to explicitly activate your menu with ACTIVATE MENU. Your menu can be made available when FoxPro is waiting for keyboard input.

READ and Menus

READ activates controls in FoxPro screens. When a READ is active your menus may or may not be accessible, depending upon the type of READ issued.

When a modal READ is issued, your menu 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 including a READ WHEN clause.

To reactivate a menu that utilizes the FoxPro menu system, issue

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

in the READ WHEN clause.

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.

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

When a READ is issued, access to your menu also depends on the setting of SYSMENU. SYSMENU is discussed in detail in the following section.

SET SYSMENU

SET SYSMENU is a powerful command for manipulating menus that utilize the FoxPro system menu. 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. Here are some of the forms SET SYSMENU can take:

- SET SYSMENU ON – Your menu bar is accessible during program execution when FoxPro is waiting for keyboard input (during BROWSE, a non-modal READ, MODIFY MEMO, etc.) Your menu bar isn't displayed, but may be displayed and made 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 at any time 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 is waiting for keyboard input.
- SET SYSMENU TO DEFAULT – The default FoxPro menu system is restored to its default configuration.

For additional information on SET SYSMENU, see the FoxPro *Commands & Functions* manual.

PUSH MENU and POP MENU

PUSH MENU and POP MENU let you save and restore menus. A menu can be pushed onto a "stack" in memory, and then be restored later by popping it off the stack.

Pushing a menu onto the stack does not remove it from the screen. It just saves its current state. While the menu is saved in memory, additions can be made to the menu on screen or it can be replaced with another menu. After you've modified or replaced the original menu, you can use POP MENU to restore the original menu.

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

The ORGANIZER application demonstrates how menus can be replaced and restored. When you first run the ORGANIZER, ORGANIZER replaces the FoxPro's System popup with its own. When you choose an option from the ORGANIZER's popups, a screen program is run. The screen program pushes the current menu into memory, and runs a menu program that creates a new menu to replace the original menu. When the screen program is exited, the original ORGANIZER menu is restored (popped) from memory.

Here are the commands in the CONVERT.SCX screen program 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 is exited.

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

For additional information on PUSH MENU and POP MENU, refer to the FoxPro *Commands & Functions* manual.

Your Working Environment

Here are some suggestions to make menu design faster and easier.

Use an Extended Video Mode

If your video hardware supports an extended display mode, use it! Designing menus in an extended video mode (more than 25 screen lines) lets you display multiple code snippet windows and the Menu Design window simultaneously. Cutting and pasting between code snippets windows is easier in an extended video mode.

Manipulating Code Snippet Windows

Code snippet windows can be sized, minimized and docked like other FoxPro windows. You can place more code snippet windows on the screen if you minimize them.

When you save your menu, the state of the code snippet windows is saved as well. When you open the menu again, the code snippet windows are opened as they were when you saved your menu.

Use Quick Menu

If your menu utilizes FoxPro's system menus, you may want to create a Quick Menu and then selectively delete or add menu pads, popups or popup options.

A Quick Menu is created by choosing the **Quick Menu** option from the **Menu** menu popup. The **Quick Menu** option is only available when the Design window is empty (it contains no menu pads, popups or options).

Restoring the System Menu and Command Window

When you are testing a menu that replaces the FoxPro system menus, the system menus and the Command window may not be accessible. To restore access to the Command window and the default FoxPro system menus, execute this command before you test your menu:

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

At any time you can press Alt+F9 to restore the default system menus and the Command window. Any valid ON KEY LABEL key or key combination can be used instead of Alt+F9.

Design Considerations

Your menu design and the options you place on your menus are determined by your application. Here are some suggestions for designing your menus that can make your applications "user friendly".

Emulate the FoxPro Interface

If your application's interface emulates the FoxPro's interface, your application is quickly accessible to user's with experience with this type of interface.

The FoxPro interface is also similar to the FoxBASE+/Mac interface. A FoxBASE+/Mac user can quickly learn your PC application when it emulates the FoxPro interface.

Execute Irreversible or Rarely Used Routines from Menus

If a routine makes irreversible changes or is rarely executed, use a menu option instead of a control to execute the routine. Because making a choice from a menu requires an additional conscious effort, routines that make irreversible changes (packing or zapping a database, month or year end closing, etc.) should be executed from a menu.

Do not assign keyboard shortcuts to options that execute a routine that makes irreversible changes. This helps to prevent the option from being accidentally chosen.

Routines that are infrequently executed (printing reports, labels, etc.) are good choices to be menu options. Placing an option for an infrequently executed routine on a menu reduces clutter in your screens.

Using FoxPro System Options

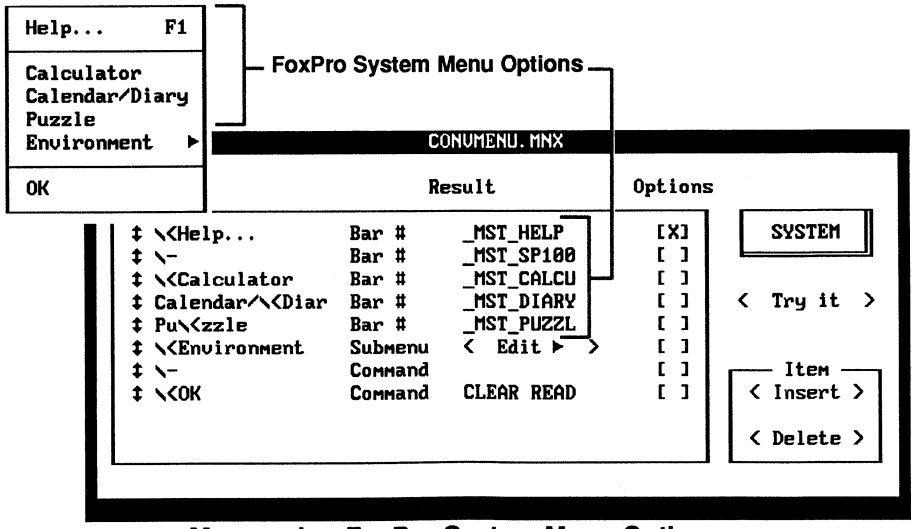
Most of the popup options available in FoxPro's system menus can be placed in your menus. When you include the name of a FoxPro option in your menu, the option is available in your menu.

When you are creating popup options in the Menu Builder, you can place a FoxPro system menu option in your popup by choosing the **Bar #** option in the **Result** popup. Type the name of the FoxPro option in the text box.

When FoxPro menu options are included on your menus, FoxPro automatically manages the enabling and disabling of the options. For example, if you include the **Paste** option in your popup, **Paste** is only enabled when you're in a text editing region and the clipboard isn't empty.

The names of the FoxPro system menu options are available:

- In the topic Menu – Systems Menu Names in the on-line help facility.
- In the section Menu – Systems Menu Names in the FoxPro *Commands & Functions* manual.
- From the SYS(2013) function. SYS(2013) returns a space delimited character string containing the names of the system menu bar, each pad in the system menu bar, system menu popups and each option in the menu popups.



Menu using FoxPro System Menu Options

Option Result Popup

The example on the previous page demonstrates how you can create a menu option by choosing **Bar #** from the **Result** popup. The **Result** popup also lets you specify a command or procedure to execute when an option is chosen, or create a hierarchical popup for an option.

Command

If you choose **Command** from the **Result** popup you can enter a command that is executed when the option is chosen. When you enter a command to execute, the command

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

is added to the menu definition code. Each option in a popup has its own unique <option number>.

Procedure

If you choose **Proc.** from the **Result** popup you create a procedure that is executed when the option is chosen. When you create a procedure to execute, the command

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

is added to the menu definition code. The <procedure name> is a unique name generated by the menu code generator; <menu name> is the name of the generated menu program.

Submenu – Hierarchical Popup

If you choose **Submenu** from the **Result** popup you can create a hierarchical popup that is displayed when the option is chosen. When you create a hierarchical popup, the command

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

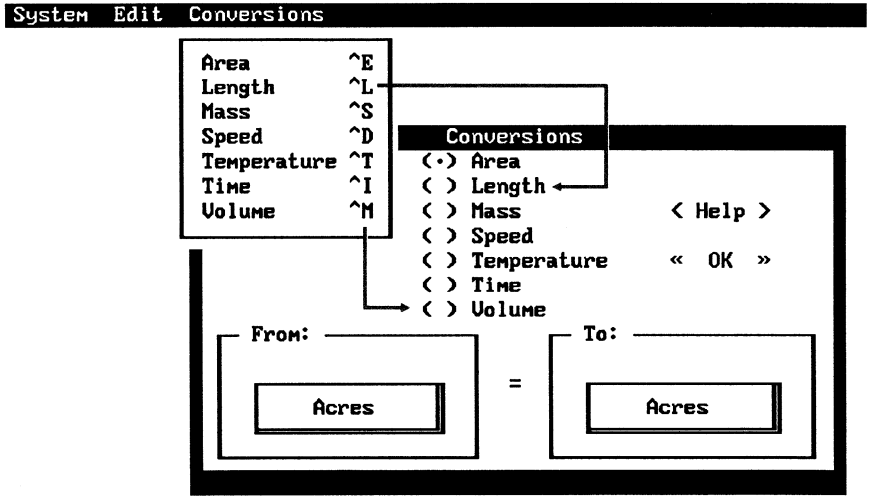
is added to the menu definition code.

Use Keyboard Shortcuts for Screen Controls

Some controls in your screens may be frequently used. In this case, it is helpful to keyboard users if you include a menu option that performs the same action as the screen control.

Be sure to create keyboard shortcuts for these types of options. When you press the keyboard shortcut, the option is chosen.

For example, the CONVERT.SCX screen has a set of radio buttons that lets you select the measurement unit (area, length, mass, etc.). A choice can be made from the radio buttons by pressing a keyboard shortcut created in the **Conversions** menu popup.



Keyboard Shortcuts for Screen Controls

The Generated Program

GENMENU, the FoxPro menu program generator, uses the information saved in a menu's .MNX database to create a menu program. The generated menu program is given an .MPR extension.

Menu code is generated and executed in a specific order. No matter how different your menus are, the generated programs will always have the same organization, and the order of execution does not vary.

Menu program code consists of four parts, and is generated and executed in the following order:

1. Setup Code – Initializes variables used by the menu, saves the current environment, creates an environment for the menu and so on. 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 (ON SELECTION PAD, ON SELECTION BAR, etc.) that specify the procedure or command to execute when a menu pad or option is chosen. The menu generator automatically creates the commands in this code section.
3. Cleanup Code – Typically used to initially turn mark characters on or off or to enable or disable menu pads and options. You create the code snippets placed in the cleanup code.
4. Procedures – Procedures are snippets of FoxPro code that are executed when a menu pad or popup is chosen. You create the code snippets placed in these procedures.

The following example contains sectioms from the generated program code for the CONVMENU menu from the ORGANIZER sample application. This example illustrates the organization of the generated program code, and how code snippets are integrated with the menu.

```
*     ************************************************************
*     *                                                          *
*     * 05/01/91          CONVMENU.MPR              19:29:11 *
*     *                                                          *
*     ************************************************************
*     *                                                          *
*     * Author's Name                                           *
*     *                                                          *
*     * Copyright (c) 1991 Company Name                         *
*     * Address                                                 *
*     * City,     Zip                                           *
*     *                                                          *
*     * Description:                                           *
*     * This program was automatically generated by GENMENU.   *
*     *                                                          *
*     ************************************************************
```

Program Header. This information is always generated.

```
*     ************************************************************
*     *                                                          *
*     *                      Menu Definition                     *
*     *                                                          *
*     ************************************************************
*

SET SYSMENU TO
SET SYSMENU AUTOMATIC

DEFINE PAD _pv215rlc8 OF _MSYSMENU PROMPT "\<System" ;
       COLOR SCHEME 3;
       KEY ALT+S, "ALT+S"
DEFINE PAD _pv215rlcz OF _MSYSMENU PROMPT "\<Edit" ;
       COLOR SCHEME 3;
       KEY ALT+E, "ALT+E"
DEFINE PAD _pv215rldn OF _MSYSMENU PROMPT "\<Conversions";
       COLOR SCHEME 3;
       KEY ALT+C, ""

ON PAD _pv215rlc8 OF _MSYSMENU ACTIVATE POPUP system
ON PAD _pv215rlcz OF _MSYSMENU ACTIVATE POPUP edit
ON PAD _pv215rldn OF _MSYSMENU ACTIVATE POPUP conversion

DEFINE POPUP system MARGIN RELATIVE SHADOW COLOR SCHEME 4

DEFINE BAR _MST_HELP OF system  PROMPT "\<Help..." ;
       KEY F1, "F1";
       SKIP FOR NOT FILE("orghelp.dbf")
DEFINE BAR _MST_SP100 OF system  PROMPT "\-"
DEFINE BAR _MST_CALCU OF system  PROMPT "\<Calculator"
DEFINE BAR _MST_DIARY OF system  PROMPT "Calendar/\<Diary"
DEFINE BAR _MST_PUZZL OF system  PROMPT "Pu\<zzle"
DEFINE BAR 6 OF system  PROMPT "\<Environment"
DEFINE BAR 7 OF system  PROMPT "\-"
DEFINE BAR 8 OF system  PROMPT "\<OK"

ON BAR 6 OF system ACTIVATE POPUP environmen
ON SELECTION BAR 8 OF system CLEAR READ
```

Remove FoxPro system menu.

Always display the system menu bar.

Create menu pads in system menu bar.

Procedures executed when pads are chosen.

System popup, activated when System pad is chosen.

Create options in the System popup.

Procedure and command exectuted when these options are chosen.

```
*     ************************************************************
*     *                                                          *
*     *                 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
```

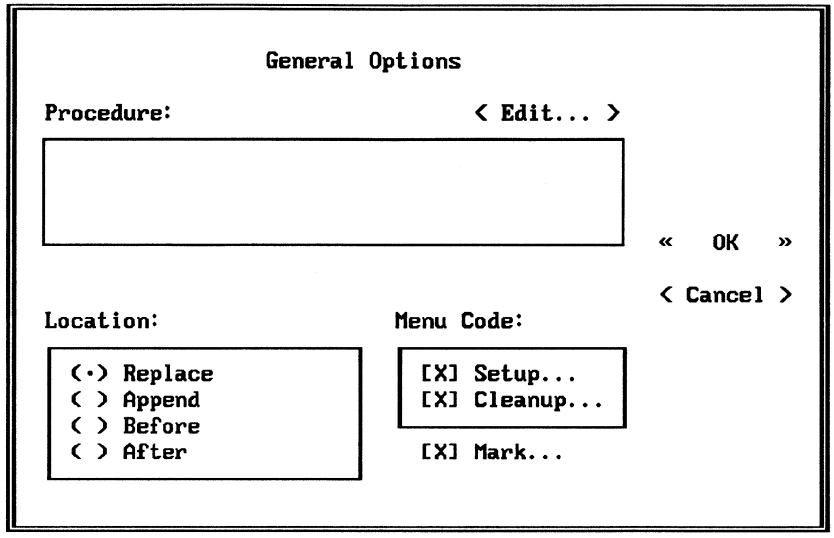
Cleanup Code. This code is created in the General Options Dialog, and is executed after the menu is created and activated.

```
*     ************************************************************
*     *                                                          *
*     * _PV215RLQE  ON SELECTION BAR 1 OF POPUP environmen      *
*     *                                                          *
*     * Procedure Origin:                                       *
*     *                                                          *
*     * From Menu:  CONVMENU.MPR,            Record:   12       *
*     * 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. Turns 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 is displayed.



General Options Dialog

In the General Options dialog you can:

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

Setup Code

When you choose **Setup...** in the General Options dialog, you can create setup code for your menu. This setup code is generated and executed before the menu definition code and any other menu code.

Because the setup code is executed first, you can initialize variables used by the menu, save the current environment, create an environment for the menu and so on. The setup code snippet typically contains code that:

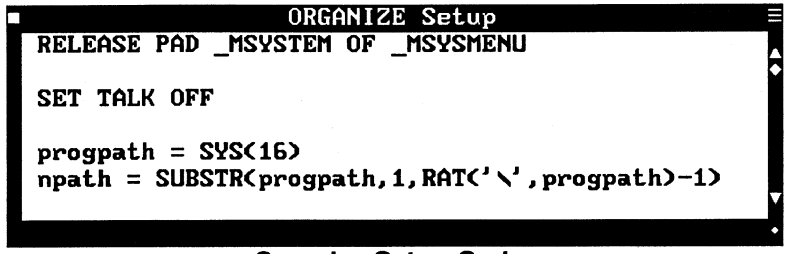
- Creates memory variables and arrays used by the menu
- Saves the current environment to be restored later
- Creates a new environment
- Specifies an error handling routine
- Saves the current menu

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

When you choose **Setup...**, a text editing window opens behind the dialog. When you close the General Options dialog by choosing **OK** you can create the setup procedure in the text editing window.

Setup Code Example

Here is the setup procedure for the ORGANIZE menu as it appears in the Setup text editing window, followed by the generated setup code.



Organize Setup Code

```
*     ************************************************************
*     *                                                          *
*     *                        Setup Code                        *
*     *                                                          *
*     ************************************************************
*
RELEASE PAD _MSYSTEM OF _MSYSMENU <-- Remove System Pad

SET TALK OFF <-- Suppress output to the screen from commands

progpath = SYS(16) <-- Program name with path
npath = SUBSTR(progpath,1,RAT('\',progpath)-1) <-- Remove 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 new path to applications

opath = SET("PATH") <-- Current path
IF AT(npath,opath)=0 <-- If new path isn't contained in old path
      opath = opath + IIF(EMPTY(opath),"",";") + npath + ;
            ";"+npath+"\DBFS" + ;
            ";"+npath+"\REPORTS"
      SET PATH TO &opath <-- Enable new path
ENDIF

Add DBFS and REPORTS directories to old path
```

Procedure

You can create a general procedure that is executed when any *menu pad* is chosen. The only times that a general procedure is *not* executed is when a command, menu popup or another procedure is assigned to a menu pad.

For example, when you are developing an application, you may want the general procedure to be a code snippet program stub. This way if you haven't written code for a menu pad, the code snippet is executed. The general procedure code snippet could be this single command:

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

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

To create a general procedure, type the code in the procedure editing region in the General Options dialog, or choose the **Edit...** push button and enter the code in the text editing window that opens behind the dialog.

Generated Menu Code

When the menu code is generated, an ON SELECTION MENU command is placed after the menu definition code. If the code snippet in the general procedure is one line long, it is included in the ON SELECTION MENU command. If the snippets in your general procedure are 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 creates a unique generated <procedure name>, and your code snippets are placed in this procedure.

Cleanup Code

When you choose **Cleanup...** in the General Options dialog, you can create cleanup code for your menu. Cleanup code is generated after the setup code and the code that creates the menu, and before any procedures assigned to menu pads or popup options.

Cleanup code typically contains code snippets that:

- Initially turn menu pad or popup options mark characters on or off
- Initially enable or disable menu pads, popups or options

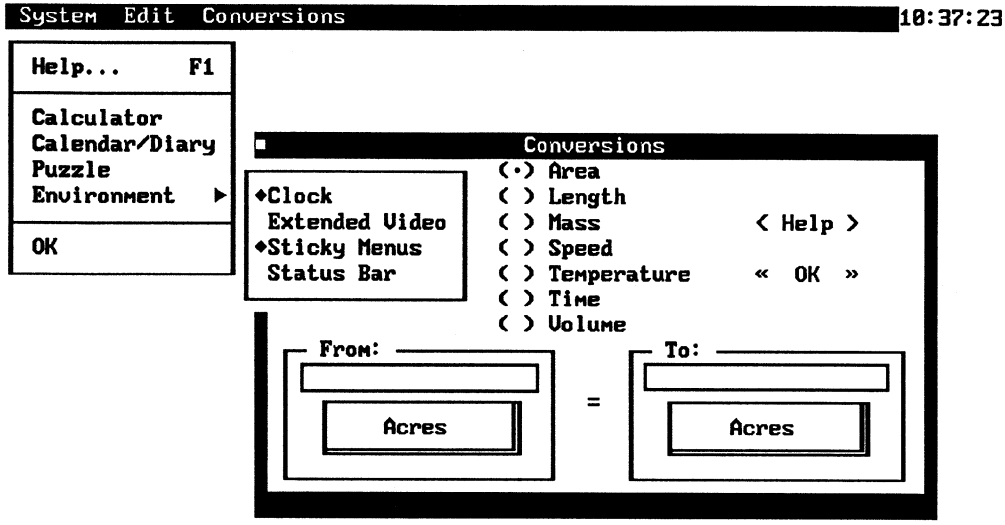
Cleanup code is executed immediately after the menu definition code, and can be used to *initially* turn mark characters on or off, enable or disable menu pads, popups and options and so on.

Code snippets can be included in menu pad or option *procedures* to turn a mark character on or off, enable or disable a menu pad and so on when a menu pad, popup or option is *chosen*.

Cleanup Code Example

Here is the cleanup code snippet for the cleanup procedure from the CONVMENU menu. The cleanup code for CONVMENU turns the mark characters on or off as appropriate 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 is displayed.




```
FOR i = 1 TO cntbar('environmen')  <-- Loop for number of options
  DO CASE
  CASE PRMBAR('environmen',i) = 'Clock'  <-- Clock option
    SET MARK OF BAR i OF environmen TO  SET('CLOCK') = 'ON'
  CASE PRMBAR('environmen',i) = 'Extended Video'  <-- Video option
    SET MARK OF BAR i OF environmen TO  SROW() > 25
  CASE PRMBAR('environmen',i) = 'Sticky'  <-- Menu option
    SET MARK OF BAR i OF environmen TO  SET('STICKY') = 'ON'
  CASE PRMBAR('environmen',i) = 'Status Bar'  <-- Status bar option
    SET MARK OF BAR i OF environmen TO  SET('STATUS') = 'ON'
  ENDCASE
ENDFOR
```

Turn clock on or off as appropriate

Toggle video mode as appropriate

Toggle menu state as appropriate

Turn status bar on or off as appropriate

Location

The **Location** radio buttons determine how your menu will be integrated with the FoxPro system menu bar.

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

```
SET SYSMENU TO
```

This removes all pads from the menu system. Your menu pads are placed on the menu bar.

Append Choosing **Append** automatically adds your menu to the right end of the system menu bar. Additional commands are not generated.

Before Choosing **Before** places your menu in the system menu bar before the menu pad you choose from the **Location** popup. *All* your menu pads are placed before the system menu pad you chose. The following clause is appended to every 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 your menu in the system menu bar after the menu pad you choose from the **Location** popup. *All* your menu pads are placed after the system menu pad you chose. The following clause is appended to every 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 every menu pad by choosing the **Mark...** check box. A mark character is placed before a menu pad when a specific condition exists.

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

When you choose the **Mark...** check box, a scrollable list of mark characters is displayed. You can choose a character from this list as the mark character for menu pads. If you choose a character from this list, it replaces the default ♦ mark character.

While you can use any character as a mark, there are a few characters that make better marks than others. Some good mark characters are:

Character	ASCII Value
◆	4
●	7
*	42
✓	251

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

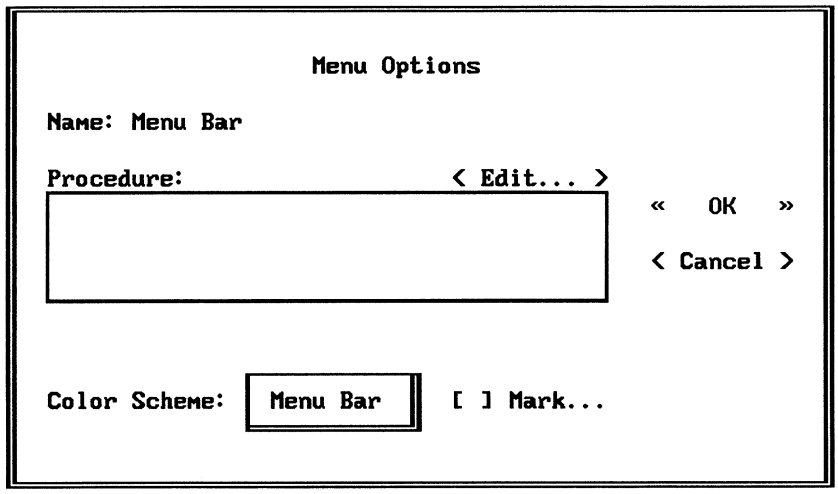
Generated Menu Code

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...

In addition to general menu options, options are available for the menu bar and menu popups. When you create menu pads in the Menu Builder, **Menu Bar Options...** is available on the **Menu** menu popup. Choosing **Menu Bar Options...** brings forward the Menu Bar Options dialog.



Menu Bar Options Dialog

In the Menu Bar Options dialog you can:

- Create code snippets for a global procedure that is executed 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 is executed when *any* option is chosen from any popup. However, if a command, hierarchical popup, menu popup or procedure is assigned to a popup option, this menu bar procedure *is not* executed when the option is chosen.

To create a menu bar procedure, type the code in the procedure editing region in the Menu Bar Options dialog, or choose the **Edit...** push button and enter the code in the text editing window that opens behind the dialog.

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 (color scheme 3) is selected by default.

Generated Menu Code

When the menu code is generated, a COLOR SCHEME <color scheme number> clause is appended 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 is placed before a menu pad when a specific condition exists.

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

When you choose the **Mark...** check box, a scrollable list of mark characters is displayed. You can choose a character from this list as the mark character for menu pads. If you choose a character from this list, 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.

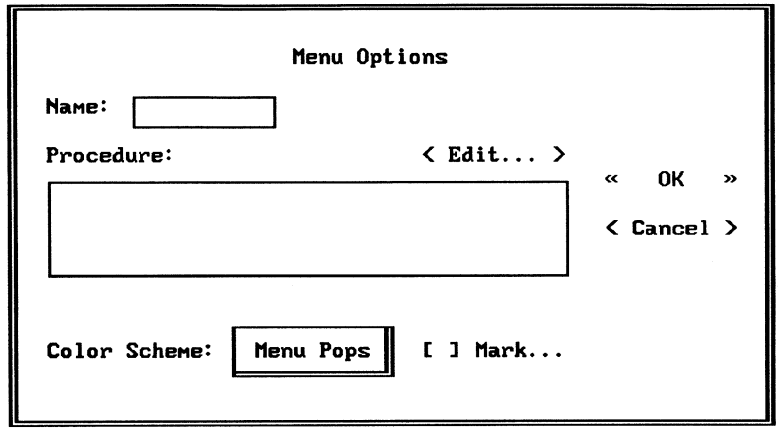
Generated Menu Code

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 in the following example:

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

Menu Popup Options...

When you create a menu popup in the Menu Builder, an option with the same name as the popup you create replaces the **Menu Bar Options...** option on the **Menu** menu popup. If you choose this option from the **Menu** menu popup, the Menu Popup Options dialog comes forward.



Menu Popup Option Dialog

In this dialog you can:

- Create code snippets that are executed 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 all the popup's options

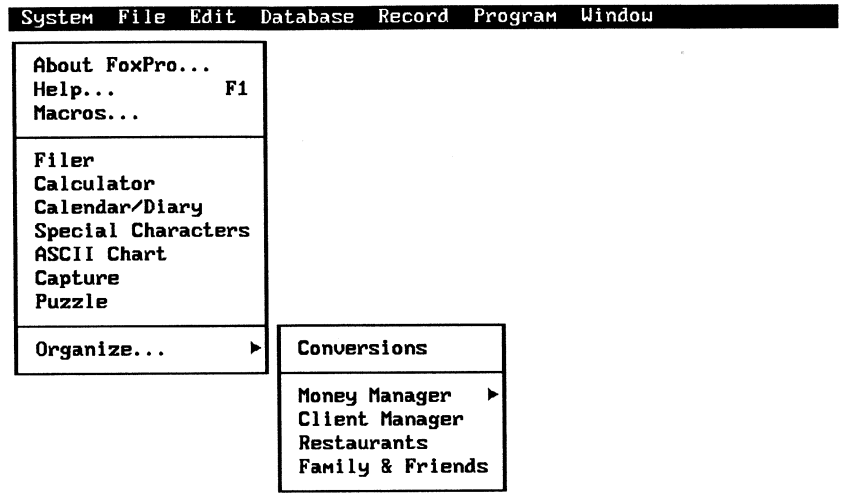
Procedure

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

To create code snippets for the options, type the code snippets in the procedure editing region, or choose the **Edit...** push button and enter the code in the text editing window that opens behind the dialog.

Menu Popup Procedure Example

Here is the menu popup procedure from the ORGANIZE menu. The menu popup procedure is executed whenever an option is chosen from the ORGANIZER popup. PROMPT( ) returns the prompt of the option chosen and a corresponding program name is stored to the memory variable MODULE.




```
ON SELECTION POPUP organizer;
      DO _pv00u3xof IN SAMPLE\MENUS\ORGANIZE.MPR
```

When any option is chosen, execute this procedure.

```
PROCEDURE _pv00u3xof
PRIVATE module, where, newpath
```

Memory variables private to this procedure.

```
DO CASE
CASE PROMPT( )="Conversions"
      module = "convert"
CASE PROMPT( )="Client Manager"
      module = "clients"
CASE PROMPT( )="Restaurants"
      module = "restaurs"
CASE PROMPT( )="Family & Friends"
      module = "family"
ENDCASE
```

Store a value to the memory variable MODULE based on the option chosen from the popup.

Name

A popup is automatically assigned a default name. This name is the prompt of the menu pad the popup is attached to. If the popup is a hierarchical popup, the default popup name is the prompt of the option the popup is attached to. The default name is displayed in the Name text box. You can specify a different name for the popup in this text box.

Generated Menu Code

When the menu code is generated, the following line is placed in the menu definition code:

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

The <popup name> is taken from 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 selected by default.

Generated Menu Code

When the menu code is generated, a COLOR SCHEME <color scheme number> clause is appended 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 is displayed. You can choose a character from this list as the mark character for the menu pad or popup option. If you choose a character from this list, it replaces the default ♦ mark character.

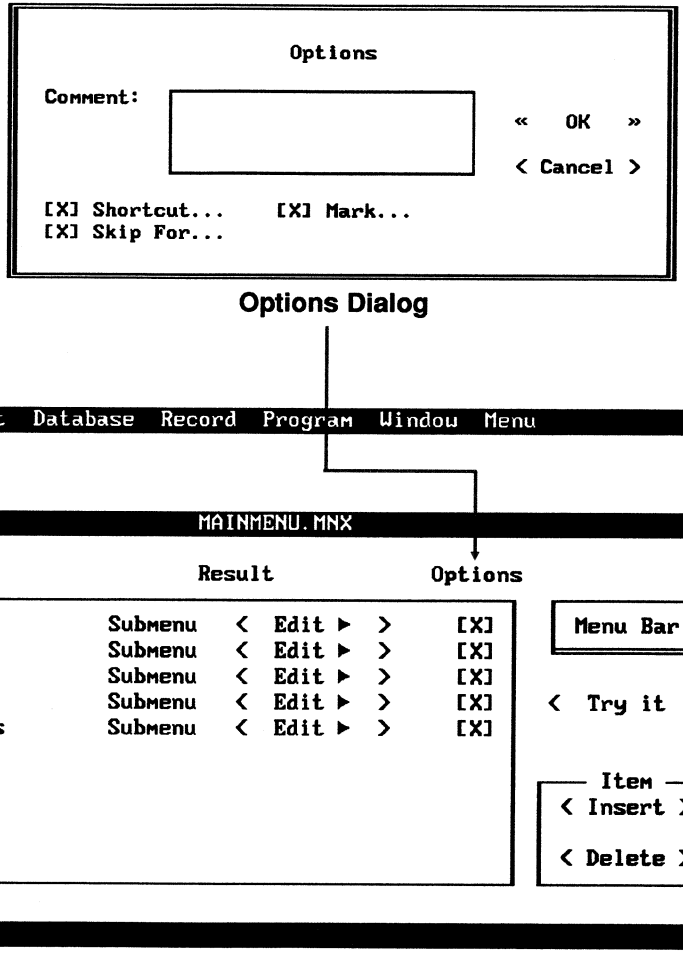
This option only specifies a mark character for the popup's options. It does not turn the mark character on or off.

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 can be disabled when 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. The Options dialog is displayed.



Options Dialog

Shortcut...

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

Unlike a hot key, a keyboard shortcut immediately chooses a menu pad or option. To choose a menu pad with a hot key, the menu bar must first be activated by pressing the Alt key. Then press the hot key to choose the menu pad. To choose an option with a hot key, the popup containing the option must first be displayed. Press the option's hot key to choose it.

Press the key or key combination shortcut for the menu pad or option. The shortcut is placed in the Key Label and Key Text text boxes. The *key text* is placed to the right of a menu pad or popup option as a shortcut reminder. To keep the key text reminder from being displayed next to a menu pad or popup option, delete the key text from the Key Text text box.

If your menu utilizes the FoxPro system menu bar _MSYSMENU, key text reminders are *not* displayed next to menu pads in the menu bar. However, they are displayed next to your popup options.

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

Generated Menu Code

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

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

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

If you delete the reminder, the generator adds the following to DEFINE PAD or DEFINE BAR:

```
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. A mark character is placed before a menu pad or popup option when a specific condition exists. For example, a mark can be placed next to an option to indicate that 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 is displayed. You can choose a character from this list as the mark character for the menu pad or popup option. If you choose a character from this list, 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 any other dialog.

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 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 on a logical condition. If the logical condition you specify evaluates to true (.T.), the pad or option is disabled and cannot be selected or chosen. If the logical condition evaluates to false (.F.), the pad or option is enabled and can be selected or chosen.

Choosing **Skip For...** displays the Expression Builder. Create a logical expression that will determine if the menu pad or popup option will be disabled or enabled. For example, if the expression

```
CDOW(DATE( )) = 'Monday'
```

is entered in the expression builder, the menu pad or popup option is *enabled* on 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 or the popup option's DEFINE BAR command. For example, if you create the logical expression in the example above, the menu generator adds the following to DEFINE PAD or DEFINE BAR:

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

Comment

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

Generated Menu Code

The comment is not included in generated menu code.

Debugging your Menus

After you create your menu you may find that it doesn't behave as you expected. In this case, you can use the menu code to help isolate the problem. FoxPro's powerful debugging tools, the Trace and Debug windows, make it easy to pinpoint the source of a menu problem.

You can also open, examine and modify a menu program in the FoxPro text editor with MODIFY COMMAND <menu program name>. Be sure to include the extension (.MPR) with the menu program name.

Any changes you make to the *menu program* will be lost when you make changes to menus through the Menu Builder and the menu program is generated again.

Trace Window

The Trace window can display menu program code as the menu program executes. The executing program line is highlighted. You can set breakpoints on menu program lines to pause program execution just before each line, and you can single step through a menu program, executing a single program line at a time.

Debug Window

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

FoxDoc

FoxDoc is an automatic program documentor that is included with FoxPro. FoxDoc is a powerful debugging tool that creates documentation for a menu program or an application containing menus.

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

Comment Boxes

The menu generator automatically inserts a comment box before each of the menu program sections. Comment boxes describe the origin of the code that follows the box and are useful when debugging menu programs. Information in comment boxes is also used by FoxDoc, the FoxPro program documentor.

The 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 following are included in the comment box:

- The procedure name
- The command that calls the procedure
- The number of the record in the menu's .MNX database that contains the procedure
- The prompt of the menu pad or popup option that calls the procedure
- The snippet number — each procedure in the generated code is numbered.

For additional information on debugging programs in FoxPro, see the Debugging Your Applications chapter in this manual.

Here's a procedure comment box taken from the CONVMENU menu program.

```
*      ************************************************************
*      *                                                          *
*      * _PV40X706B ON SELECTION BAR 1 OF POPUP environmen        *
*      *                                                          *
*      * Procedure Origin:                                       *
*      *                                                          *
*      * From Menu:  CONVMENU.MPR,            Record:   12       *
*      * Called By:  ON SELECTION BAR 1 OF POPUP environmen       *
*      * Prompt:     Clock                                       *
*      * Snippet:    1                                           *
*      *                                                          *
*      ************************************************************
```

Additional Tips

Hide the Command Window

If your application is run in the Development version of FoxPro (as opposed to a Runtime version), you may want to hide the Command window so only your menu can be accessed. To hide the Command window, create a small window and ACTIVATE the Command window in the small window. Do not ACTIVATE the small window. These two 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

Menu programs can contain commands that execute screen programs and screen programs can contain commands that invoke a menu system. The method you choose is entirely up to you.

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

- Options that allow the user to access screen controls with control key shortcuts.
- Seldom used options that are not available in screen controls.
- Irreversible options (such as PACK).

Activating a Menu System

Menus created in the Menu Builder automatically utilize the FoxPro menu system. If you create a menu that utilizes the FoxPro menu system, it isn't necessary to explicitly activate your menu with ACTIVATE MENU. Your menu can be made available when FoxPro is waiting for keyboard input.

READ and Menus

READ activates controls in FoxPro screens. When a READ is active your menus may or may not be accessible, depending upon the type of READ issued.

When a modal READ is issued, your menu 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 including a READ WHEN clause.

To reactivate a menu that utilizes the FoxPro menu system, issue

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

in the READ WHEN clause.

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.

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

When a READ is issued, access to your menu also depends on the setting of SYSMENU. SYSMENU is discussed in detail in the following section.

SET SYSMENU

SET SYSMENU is a powerful command for manipulating menus that utilize the FoxPro system menu. 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. Here are some of the forms SET SYSMENU can take:

- SET SYSMENU ON – Your menu bar is accessible during program execution when FoxPro is waiting for keyboard input (during BROWSE, a non-modal READ, MODIFY MEMO, etc.) Your menu bar isn't displayed, but may be displayed and made 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 at any time 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 is waiting for keyboard input.
- SET SYSMENU TO DEFAULT – The default FoxPro menu system is restored to its default configuration.

For additional information on SET SYSMENU, see the FoxPro *Commands & Functions* manual.

PUSH MENU and POP MENU

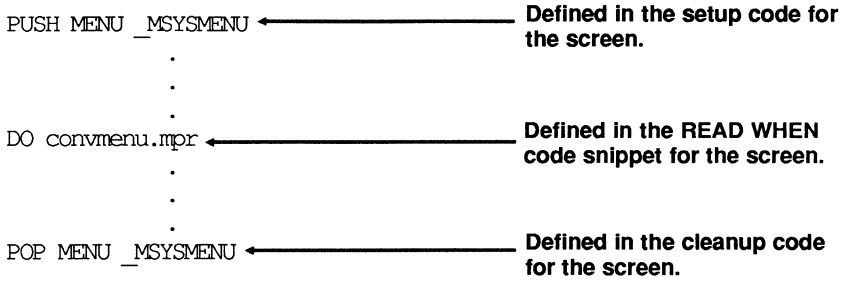
PUSH MENU and POP MENU let you save and restore menus. A menu can be pushed onto a "stack" in memory, and restored later by popping it off the stack.

Pushing a menu onto the stack does not remove it from the screen. It just saves its current state. While the menu is saved in memory, additions can be made to the menu on screen or it can be replaced with another menu. After you've modified or replaced the original menu, you can use POP MENU to restore the original menu.

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

The ORGANIZER application demonstrates how menus can be replaced and restored. When you first run the ORGANIZER, ORGANIZER replaces the FoxPro's System popup with its own. When you choose an option from the ORGANIZER's popups, a screen program is run. The screen program pushes the current menu into memory, and runs a menu program that creates a new menu to replace the original menu. When the screen program is exited, the original ORGANIZER menu is restored (popped) from memory.

Here are the commands in the CONVERT.SCX screen program 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 is exited.



Calling a Menu Program

When you call a menu program, use the following syntax:

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

You must include the .MPR extension. Because different types of executable files (menus, screens, queries, etc.) may have the same name, you must include the .MPR extension to execute a menu.

FoxPro executes the compiled .MPX version of the menu program. If the compiled .MPX version of the menu program cannot be located, the source .MPR menu program is automatically compiled, creating the compiled .MPX version.

For additional information on PUSH MENU and POP MENU, refer to the FoxPro *Commands & Functions* manual.

Calling a Screen Program

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

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

You must include the .SPR extension. Because different types of executable files (menus, screens, queries, etc.) may have the same name, you must include the .SPR extension to execute a Screen.

FoxPro executes the compiled .SPX version of the menu program. If the compiled .SPX version of the menu program cannot be located, the source .SPR menu program is automatically compiled, creating the compiled .SPX version.

Keyboard Shortcuts for Screen Controls

Some controls in your screens may be frequently used. In this case, it is helpful to keyboard users if you include a menu option that performs the same action as the screen control. The same code that defines the behavior of a screen control can be copied and pasted into the code snippet that defines the behavior of a menu option.

Be sure to create keyboard shortcuts for these types of options. When you press the keyboard shortcut, the option is chosen.

For example, the CONVERT.SCX screen has a set of radio buttons that lets you select the measurement unit (area, length, mass, etc.). A choice can be made from the radio buttons by pressing a keyboard shortcut created in the **Units** menu popup.

5 Project — The Main Organizing Tool

The Project Manager is a FoxPro power tool that helps developers organize their applications. The Project Manager unifies and coordinates the elements of a FoxPro application by gathering the necessary components (screens, menus, programs, reports, etc.) into a project. A project created by the Project Manager manages the interrelations between the interface elements and ensures that the pieces are up-to-date when you are ready to build the application.

Creating a project is usually the first step in the process of developing a FoxPro application. You can add in all the pieces for an application, even if they aren't complete, then edit the files through the Project window.

This chapter describes the following topics:

- Advantages of a project
- What a project can contain
- One project versus multiple projects
- Home directory for portable applications
- Selecting a main file
- Including modifiable files in applications
- Unknown references in projects
- Procedural code in projects

Advantages of a Project

Locates and Assembles Referenced Files

When you build a project, FoxPro automatically locates and gathers all components of the application. To create a project from an existing application, just add the startup program for the application and rebuild the project. All referenced programs, screens, menus, reports, queries, labels and libraries are automatically incorporated in the project.

Remembers the Location of Every File it Contains

A project keeps track of the location of all programs, screens, reports, etc., that comprise an application. This makes it very convenient for you to organize programs, screens and other components in directories according to function, subsystem, or other appropriate category. This also makes it easy for you to maintain an application.

Accesses Prewritten Programs and Interface Components Easily

Because projects permit application components to be scattered across many directories, they also make it easy to access libraries of prewritten program elements (like control panels, browsers, etc.) that are stored in a common directory and used by many applications.

Stores Object Code

Object code is stored in the project itself (in memo fields). This reduces clutter on your disk caused by saving individual object files for each compiled version of a program.

Tracks Current Versions of Files

When you build a project, FoxPro checks that the object code stored in the project is up to date. If it isn't, FoxPro recompiles programs, regenerates and recompiles screens and menus, etc. This project feature is similar to "make" utilities with which you may be familiar.

Streamlines Distribution

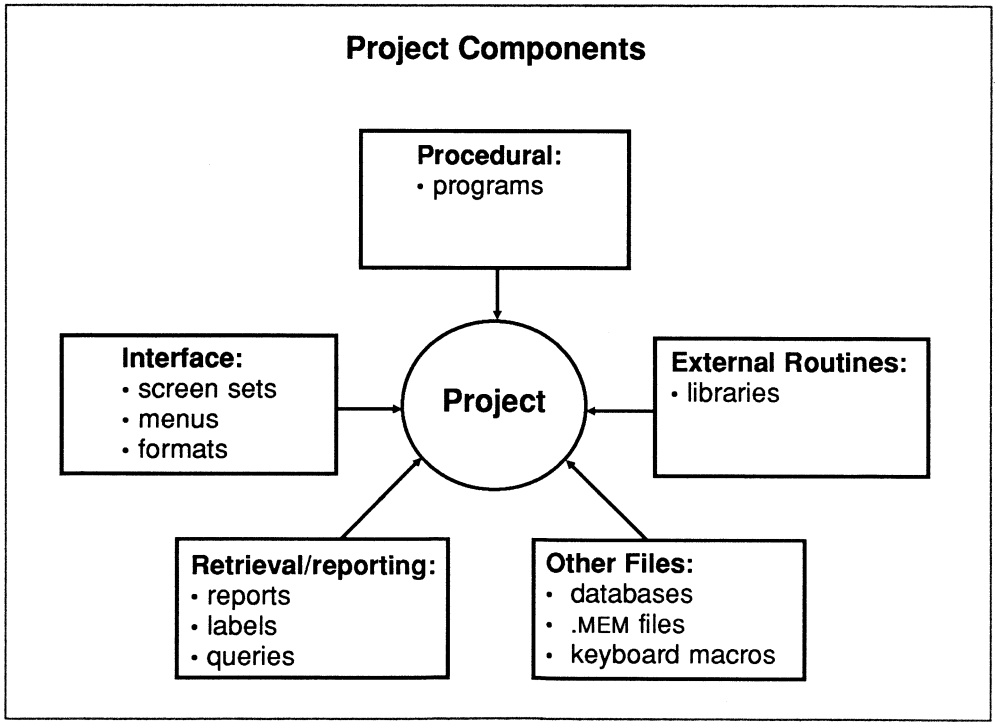
When an application or an .EXE is generated from a project, all the elements of the application are gathered together into a single .APP or .EXE file. This makes distribution of an application particularly convenient.

What Can Projects Contain?

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

For purposes of this discussion, the components will be grouped according to function:

- 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 multiple 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
- RESTAURS.PJX
- TRANS.PJX

The ORGANIZER was designed with multiple projects but could have been designed with just 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 cases where you sell or update individual modules of an entire application. With multiple projects, you must remember that when you make changes to a component that is shared by several projects, such as BROWSER.SCX, you must rebuild all the projects containing the component instead of just one project.

Home Directory for Portable Applications

Each project has a home directory.

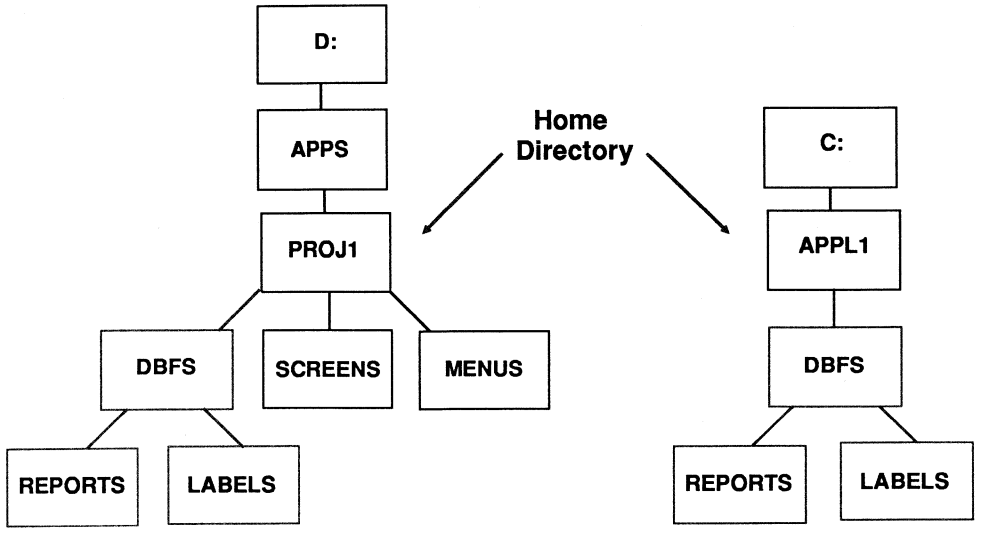
You can specify a home directory in the Home Directory area of the Project Options dialog. For details, refer to the chapter titled Project Manager in the FoxPro *Interface Guide*.

Files needed by the project should be stored in subdirectories of the home directory. It is handy to create a subdirectory for each type of file needed by the project so you can organize files by file type.

When you distribute an application, you can use any directory on the destination machine as the new home directory. The new home directory can be *any* directory with *any* name at *any* level of the directory structure.

If any files are listed in your project for reference purposes but marked as excluded (refer to Including Modifiable Files in Applications), you must distribute these files in addition to the application. Off of the new home directory, you must create the subdirectories where these files are located, duplicating the directory names and structure, and you must copy the referenced files into the proper directories.

Just copy the application to its new home directory, create the needed subdirectories off of the new home directory and copy the referenced files to these subdirectories. (You may want to create an installation routine to create directories and copy files.)



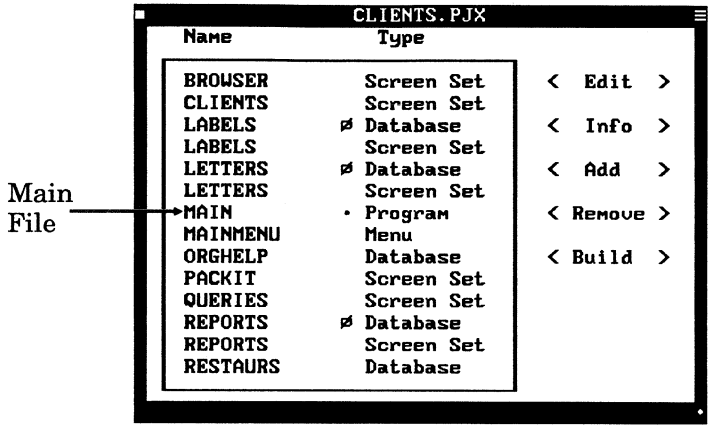
Directory Structure for Developer **Directory Structure for User**

In the example above, the home directory for development is D:\APPS\PROJ1. In the left directory structure, subdirectories have been created for databases, reports, labels, screens and menus.

The new home directory is C:\APPL1. Databases, reports and labels were listed in the project for reference only because the user needs to be able to modify them, so the DBFS, REPORTS and LABELS subdirectories must be created on the user's machine off of APPL1 just as they existed off of PROJ1. The referenced files must be copied to the appropriate subdirectory.

Selecting a Main File

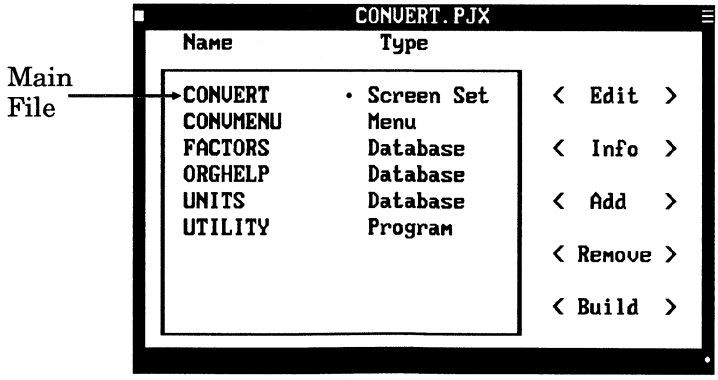
The first file added to a project that FoxPro can execute (menu, screen or program) appears in the Project window with a bullet next to its type, indicating that it is the main file. When you run an application, the main file in the project is executed first. To specify a different main file, use **Set Main** on the **Project** menu.



CLIENTS Project

The main file is the file that starts your application (or module, in the case of an application consisting of several projects). In the case of CLIENTS.PJX, the file named MAIN is used to start the CLIENTS module when **Client Manager** is chosen on the **Organize...** submenu.

Below, the CONVERT screen set is the main file and is executed first in CONVERT.APP



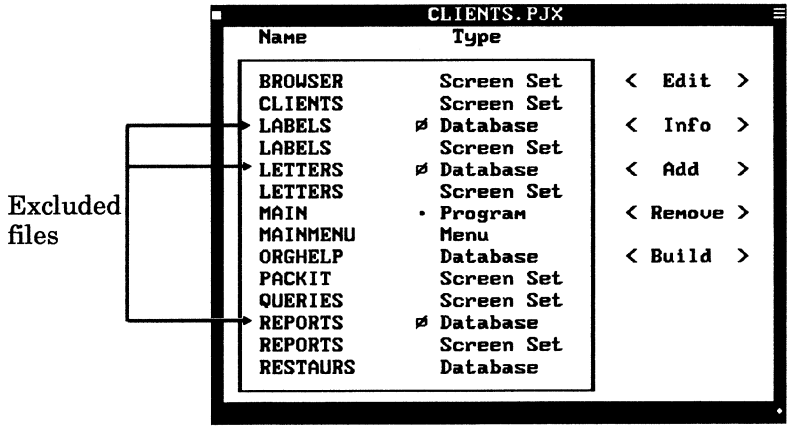
Convert Project

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, all files in the project that can be executed (programs, screens, menus) are combined as part of the application code. Files that can't be executed, such as reports, labels and databases, can also be included in the application in read-only form.

If you want your users to be able to modify any non-executable files in the project, you can select the file in the Project window and choose **Exclude** from the **Project** menu popup. A ø appears in the Project window next to the selected file name.

Excluded files are listed in the Project window for your reference but must be *distributed in addition to the application* if needed by the application.



CLIENTS Project

In CLIENTS.PJX, three databases are excluded so that users can modify them when running the ORGANIZER. You should *exclude* any files such as databases, indexes, reports and labels that you want the end user to be able to modify. You'll probably want to *include* help databases and databases that contain look up information to prevent users from modifying them.

If you change your mind and want to include a file you excluded, select the file in the Project window and choose **Include** from the **Project** menu popup.

Unknown References in Projects

Occasionally, when you build a project, the Project Manager alerts you that it can't find a file or an array that is referenced in the project. In this case, you can do one of the following:

- Temporarily ignore the message and build the project without resolving the reference
- In the case of a file, manually add the file to the project then build the project
- Add an EXTERNAL command to the appropriate location so that FoxPro automatically includes the file or finds the array, then build the project

You may need to include an EXTERNAL PROCEDURE command as shown below to identify an external procedure or 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, you need an EXTERNAL REPORT command.

```
EXTERNAL REPORT dataentr        && REPORT dataentr must exist
STORE 'dataentr' TO reportfile
MODIFY REPORT (reportfile)
```

When an array is created in a program then used in a lower level program as in the following example, you may need to include an EXTERNAL ARRAY command to tell the Project Manager where to look outside of DISPINVO for the INVOICE array.

```
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 an array is passed to a UDF or procedure, you may need to identify the array in the UDF or procedure for the Project Manager. To do this, you can 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 *Commands & Functions* manual.

Procedural Code in Projects

The Project Manager combines files into a single application. In most applications, you'll want to 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 may 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. This section uses examples from the ORGANIZER's procedural code, MAIN.PRG, to illustrate the use of procedural code in projects.

Error Handling

One of the first procedure calls in MAIN.PRG is to an error handling routine. An error handling routine traps for errors in your application so you can gracefully recover from the error.

When you include an error handling routine at the beginning of your startup program, any error that occurs in the application (including the startup program) can be caught. When an error occurs in the ORGANIZER, the ERRORHANDLER procedure that follows is executed.

```
*
* 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 error window

Display error window

Display error message in window

Clean up environment

Saving the Current Environment

If your application returns control to the Command window or to another application when it is finished, you should save the current environment in your startup routine so you can restore it later. If your application returns to DOS when it finishes, you don't need to restore the environment.

The environment includes:

- Files (database, index, etc.) open in all 25 work areas
- Relations between database files
- Filters in effect
- DEFAULT and PATH settings
- Current procedure, help and resource files
- SET command status (ON, OFF, etc.)
- 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 is run and your application sets TALK OFF, the application should SET TALK ON before terminating.

TALK is a setting that is ON by default but usually needs to be OFF when a user runs an application. If you want TALK to be OFF, you should make the first line in your application SET TALK OFF. In the ORGANIZER, the following code from ADDUSERS.PRG checks to see if TALK is on and, if necessary, sets TALK OFF and records the fact that TALK was on.

```
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 EXACT, SAFETY and DECIMALS, so MAIN.PRG saves the current settings of these commands in PUBLIC memory variables. These memory variables are released from memory when the application is finished. The following command creates these variables in MAIN.PRG:

```
PUBLIC area, exact, safety, deci
```

The selected work area is saved with the following command:

```
area = SELECT()
```

These lines from MAIN.PRG save the setting of EXACT, SAFETY and DECIMALS.

```
exact = SET("EXACT")
safety = SET("SAFETY")
deci = SET("DECIMALS")
```

Additional Commands to Save Environment Settings

You can also use the following commands to save FoxPro's current environment, keyboard macros, memory variables and arrays, the screen and windows to a disk file, a memo field or memory for later restoration.

- CREATE VIEW – Saves the current FoxPro environment. The environment can be restored with RESTORE VIEW.
- SAVE MACROS – Saves the current keyboard macros to a file or memo field. RESTORE MACROS restores the macros from the file or memo field.
- SAVE TO – Saves the current memory variables and arrays to a file or memo field. RESTORE FROM restores the variables and arrays.
- SAVE SCREEN – Saves the current screen or window image to memory. RESTORE SCREEN restores the screen or window from memory.
- SAVE WINDOW – Saves the current window definitions to a file or memo field. RESTORE WINDOW restores the windows.

Creating the New Environment

Once you have saved the current environment, you can create a new environment for the application. Some aspects of the environmental that you can set are:

- Global memory variables and arrays
- SET commands
- Colors
- Procedure, help and resource files

MAIN.PRG uses the following SET commands to establish the environment for the ORGANIZER application:

```
SET TEXTMERGE DELIMITERS
SET MEMOWIDTH TO 256
SET UDFPARMS TO VALUE
SET EXACT ON
SET SAFETY OFF
SET DECIMALS TO 18
```

Preserving and Restoring the System Menu Bar

If your application modifies FoxPro's system menu bar, you should PUSH the system menu bar into memory. If the system menu bar is pushed, you can restore it from memory later. The following command preserves FoxPro's system menu bar:

```
PUSH MENU _MSYSMENU
```

After you've modified the system menu bar, you can restore the original system menu bar you preserved in your startup code with the following command:

```
POP MENU _MSYSMENU
```

Testing For Resources

Your application may need certain resources to run — specific files, a specific amount of memory or disk space and so on. Several FoxPro functions can test for resources that your application may need:

- FILE( ) – Tests for the existence of a specified file on disk. Use FILE( ) if your application requires certain files to be available.
- SYS(2010) – Returns the FILES setting in your CONFIG.SYS system configuration file. If your application opens many files, use SYS(2010) to make sure you can open all the files.
- MEMORY( ), SYS(12), SYS(1001) and SYS(1016) – Test the amount of available memory. If your application requires a minimum amount of memory, these function can determine if your application will successfully run.
- 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 aren't available, you can display a warning before executing the rest of the application.

Utility Procedures

Procedural code in projects often contains utility procedures that don't pertain to a specific screen or menu, but may be used by several screens or menus. In the ORGANIZER, MAIN.PRG contains the following utility procedures:

- LOCATEDB – Attempts to locate and open a database, asking for your assistance if the database cannot be found.
- CHECKFPT – Checks to see if a memo file exists for a database.
- STRIPEXT – Removes the extension from a file name.
- STRIPPATH – Removes the path from a file name.

Other Development Tools

6 Debugging Your Application

FoxPro provides a comprehensive set of program debugging tools. These debugging tools — Trace window, Debug window, text editor and on-line help — can help you locate and eliminate errors in programs.

The following topics are included in this chapter:

- Compilation errors
- Runtime errors
- Debugging suggestions

Program Errors

Errors that occur in FoxPro programs can be divided into two categories:

- Compilation errors that occur when a program is compiled
- Runtime errors that occur when a program is executing

For example, suppose you want to open the CUSTOMER database with the USE command, but you type it incorrectly:

```
MUSE customer
```

When this command is compiled, an "Unrecognized command verb" error message is generated. This is an example of a compilation error.

Suppose the CUSTOMER database has been erased from disk so it cannot be located. Although the command

```
USE customer
```

is syntactically correct and will compile without error, when the command is executed the error "File 'customer' does not exist" is generated. This is an example of a runtime error.

Compilation Errors

Every FoxPro program must be compiled before you can run it. You can compile the program or let FoxPro automatically compile the program.

Interactive Compilation

Programs can be interactively compiled from the Compile dialog that appears when you choose **Compile...** from the **Program** menu popup. A log file containing compilation errors is created when the **To .ERRs** or **To File** radio button is chosen.

- If **To .ERRs** is chosen, a compilation error log file is created with the same name as the program and an .ERR file extension.
- If **To File** is chosen, you can direct compilation errors to a log file with a different name and extension.

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 FoxPro's editor to open and examine the error log file.

COMPILE Command

From the Command window or within a program, programs can be compiled with COMPILE. The SET LOGERRORS command determines if a compilation error log file is created when programs are compiled with COMPILE.

- If LOGERRORS is SET ON before you issue COMPILE, a compilation error log file is created with the same name as the program and is assigned an .ERR file extension. If an error message log file with the same name already exists, it is overwritten.
- If a program compiles without error or if LOGERRORS is SET OFF before you compile, a log file is not created. If a program compiles without error and an error file exists with the same name as the compiled program, it is deleted.

Save and Compile

When a program is created or edited in the FoxPro editor, a **Compile when saved** check box is available in the Preferences dialog. The Preferences dialog appears when you choose **Preferences...** from the **Edit** menu popup. When **Compile when saved** is checked, programs are automatically compiled each time they are saved.

- If LOGERRORS is SET ON when you save a program with **Compile when saved** checked, a compilation error log file is created with the same name as the program and is assigned an .ERR file extension. If an error message log file with the same name already exists, it is overwritten.
- If a program compiles without error or if LOGERRORS is SET OFF, the log file is not created. If a program compiles without error and an error file exists with the same name as the compiled program, it is deleted.

Causes of Compilation 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 keywords in FoxPro's 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.
- A program line is too long. A command or function cannot exceed the maximum program line length of 2,048 characters.

If a program generates compilation errors, fix the program lines that generate the errors and compile the program again. Continue this process until the program compiles without error.

Runtime Errors

Runtime errors are errors that occur during program execution. Although a program may compile without errors, it may still generate runtime errors. Runtime errors can be more difficult to pinpoint than compilation errors, and may require the use of FoxPro's Trace and Debug windows.

The Trace window displays source code for a program as the program executes. The executing program line is highlighted. 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.

The Debug window lets you monitor the values of memory variables, array elements, functions, database fields and expressions as a program executes. You can set breakpoints in the Debug window to halt program execution when the values of these items change.

The Trace and Debug windows can be open at the same time, and you can set program breakpoints in both windows. The number of breakpoints you can set in both windows is limited only by the amount of available memory.

For more information about the Trace and Debug windows, refer to the Window Menu chapter in the FoxPro *Interface Guide*.

Debugging Suggestions

The following debugging suggestions may be helpful.

Adjust Your Video Display

If your video hardware supports extended display modes, you may want to switch to an extended video mode before you start debugging. In extended mode the Trace and Debug windows can be opened below any output windows the program places on the screen.

If your video hardware does not support extended modes and a program window covers the Trace or Debug windows, pause program execution and interactively move the Trace or Debug windows.

Document with FoxDoc

Use FoxDoc to create documentation for a single program or an entire application. FoxDoc creates tree structures for an application, application summaries, variable cross-reference reports and much, much more. If you've "inherited" an application, FoxDoc will make quick sense of it.

SET ESCAPE ON

The command SET ESCAPE OFF prevents program execution from being paused by pressing Escape. If the program you're debugging contains SET ESCAPE OFF, you may want to temporarily change it to a comment. To make it a comment, place an asterisk (*) at the beginning of the program 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.

When program execution is paused, commands can be issued in the Command window to allow you to examine and change the current FoxPro environment.

DISPLAY MEMORY and DISPLAY STATUS

When program execution is paused, the DISPLAY MEMORY and DISPLAY STATUS commands provide valuable information about the current FoxPro environment:

- 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, menu popups and windows.
- DISPLAY STATUS displays the current status of the FoxPro environment. Items displayed include currently active databases and indexes, relations, low-level file status, SET command settings and information about record and file locking if you are using FoxPro/LAN, the network version of FoxPro.

Screen and Menu Code

Screens and menus can be included in a project created by the Project Manager. To make the generated screen and menu source code available for debugging, choose the **Save Generated Code** check box in the Project Options dialog. The Project Options dialog is displayed when you choose **Options...** from the **Project** menu popup.

Generated menu and screen programs 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.

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 or menu that generated the error and make your changes in the appropriate code snippet.

SET DOHISTORY

SET DOHISTORY can be used to isolate particularly stubborn bugs. Setting DOHISTORY ON places commands from programs into the Command window as they are executed. These commands can then be edited and re-executed as if they were entered directly from the Command window

SET DOHISTORY is suggested as a temporary debugging aid. DOHISTORY constructs a disk-based document as the program executes, and this can fill up even a very large disk *very quickly*. After debugging is completed be sure to remove any SET DOHISTORY ON commands from your programs before you execute or distribute them.

7 SQL Quiz

The SQL quiz is provided for your amusement and also to illustrate a few features of the SQL SELECT facility. It just scratches the surface. SQL provides FoxPro with amazing new capabilities:

- SQL allows you to specify complex relational operations non-procedurally. This means that you don't have to figure out how to do a particular operation; SQL figures it out for you.
- Using SQL can often eliminate pages of code.
- Eliminating pages of code means there are pages of code you don't have to debug.
- We've found that the SQL optimizer can usually perform operations faster than FoxPro programs hand-coded by our senior developers. This is because SQL has access to detailed internal information about the status of databases, indexes, etc., that is simply not available externally.

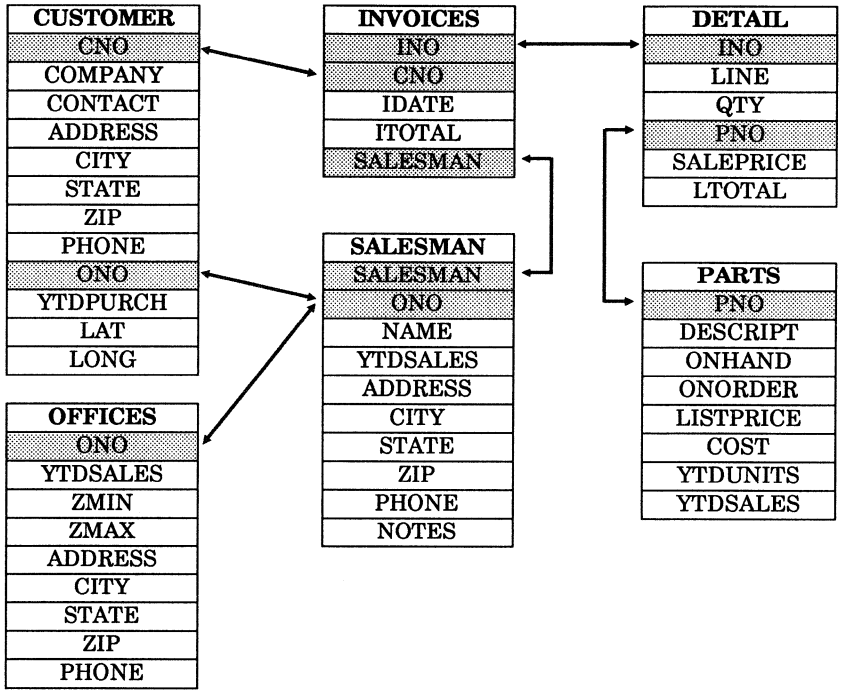
Most of these exercises have several perfectly good solutions. This is a natural consequence of SQL's richness. Therefore, don't be concerned if your solution doesn't match ours exactly because it still may be correct. If your solution generates the same result table as ours, you're in good shape.

In many cases we've provided several alternate solutions. Pick the one that seems most natural to you. (However, in some cases there are performance variations between solutions.)

Have fun!

Quiz Databases

The queries in this chapter utilize the tutorial databases: CUSTOMER, INVOICES, DETAIL, SALESMAN, OFFICES, and PARTS. These databases are located in the TUTORIAL directory.



Questions

Q1 List the company names in CUSTOMER.DBF that contain the word "Computer".

Q2 Determine how many states have at least one customer residing in them.

Q3 List the offices (i.e., OFFICE.CITY) and invoiced total for each office, largest total to smallest.

Hint: This requires a three-table join.

Q4 List part number, description, and 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 companies that purchased more than one "Woodyard lizard", total number purchased, and the total amount paid.

Hint: A 4-table join is required.

Q6 List companies that have the letter "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: A 4-table join is required.

Q9 For each salesman, list his/her sales together with the average sales for those salesmen who sold more than he/she did.

Hint: The solution involves joining the salesman table with itself. Joining a table with itself is sometimes called a "self-join".

Q10 List all pairs of part numbers and descriptions where both parts have been 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 having at least one customer located above 45 latitude.

Q12 List the states where all customers are located between 40 and 45 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 it was sold, the invoice number, and the invoice date.

Hint: A subquery might be useful.

Q15 List states with no invoices.

Q16 List all states wherein 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 in order of ascending commission.

10%	1st 5000
9%	2nd 5000
8%	3rd 5000
6%	above $15,000

Q18 List 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 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.

Q21 Show the invoice number, part number and description for all parts that occur on exactly 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
```

Notes: 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 2.0 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
```

Notes: This demonstrates the use and usefulness of the DISTINCT qualifier within COUNT.

Q3 List the offices (i.e., OFFICE.CITY) and invoiced total for each office, largest total to smallest.

Hint: This requires a three-table join.

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 part number, description, and 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 companies that purchased more than one "Woodyard lizard", total number purchased, and the total amount paid.

Hint: A 4-table join is required.

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 the letter "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"
```

Notes: 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: A 4-table join is required.

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 his/her sales together with the average sales for those salesmen who sold more than he/she did.

Hint: The solution involves joining the salesman table with itself. Joining a table 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
```

Notes: Don't try this on a large table. You'll generate *lots* of output.

Q10 List all pairs of part numbers and descriptions where both parts have been 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
```

Notes: 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 having at least one customer located above 45 latitude.

Solution

```
SELECT DISTINCT state ;
      FROM customer ;
      WHERE lat > 45
```

Q12 List the states where all customers are located between 40 and 45 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
```

Notes: Two different solutions are shown above. However, Solution B executes more than 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)
```

Notes: This query illustrates a point that will arise several more times: queries involving EXISTS can often be reformulated into equivalent queries using IN or other clauses.

Whether you should use EXISTS or IN is largely a matter of taste and style; use whichever seems natural. However, there are sometimes differences in performance, especially if it's possible to reformulate a query to eliminate subqueries.

Q14 Display the largest invoice amount together with the salesman's name, the company to which it 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)
```

Notes: This example again illustrates that it's usually possible to formulate a query using either IN or EXISTS, whichever seems more natural.

About Solution A — Contrary to what you might think, the DISTINCT clause is not required in the subquery above. If you're testing whether or not 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. In this case, the optimizer actually does the same thing whether or not you specify DISTINCT in the subquery.

About Solution B — Why, you may ask, was "*" specified in the subquery select list for this query? The answer is simple. Because we're just interested in whether or not any rows exist, it doesn't matter which fields are in the rows. So we were lazy and coded "*" because it's short and easy to type.

In this instance, there is little difference in efficiency between Solution A and Solution B.

Q16 List all states wherein 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)
```

Notes: The first of the two queries selects all states where there is at least one customer without an invoice.

The second query selects states appearing in the customer file that are not among the states selected by 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 in order of ascending commission.

10% 1st 5000
9% 2nd 5000
8% 3rd 5000
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)
```

Notes: This query illustrates the usefulness of permitting arbitrary expressions and UDFs in queries.

Q18 List 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 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
```

Notes: This query is just plain fun. Lots of similar geographic queries are possible and may even be useful.

Of course, the calculations required are complex and, if you do lots of them, may provide a good excuse to buy a math chip for your PC. In fancy trigonometric calculations, the math chip makes a significant difference.

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
```

Notes: There are at least two ways to do this query. Solution A uses EXISTS and a subquery, whereas Solution B uses just one level of query. Note again that most EXISTS queries can be reformulated using other SELECT clauses.

In our view, Solution B is simpler, hence superior.

Q21 Show the invoice number, part number and description for all parts that occur on exactly 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
```

Notes: Here again there are at least two ways to do the query. The first approach uses a subquery. The second does not and, consequently, is simpler and executes faster.

In Solution B, the DISTINCT modifier in COUNT is necessary to handle the case of a part being on one invoice multiple 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}
```

Notes: 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)
```

Notes: 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)
```

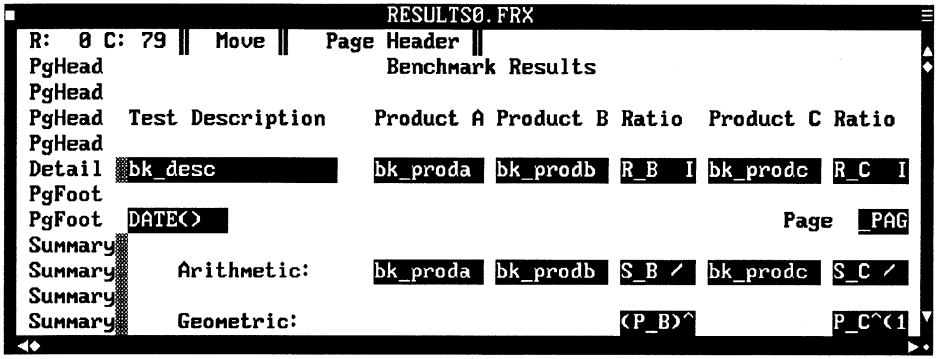
Notes: This exercise illustrates use of the ALL quantifier.

8 Report Variable Hints

With Report Writer variables, most UDFs are no longer necessary. For example, you can use variables to calculate percentages and geometric and arithmetic means without writing any code.

Report Variables

The report form below, RESULTS0.FRX, uses variables to determine the ratio, sum, count, and product (a result obtained by multiplication) for a comparison of benchmarks between Product A and Product B, and Product A and Product C. Examples in this chapter illustrate how to calculate an arithmetic and geometric mean. For information about creating report variables, see the Report Writer chapter in the FoxPro *Interface Guide*.



Database Structure				
Field Name	**Type**	**Width**	**Decimals**	**Description**
BK_DESC	Character	70	N/A	Description of Benchmark
BK_PRODA	Numeric	9	2	Benchmark for Product A
BK_PRODB	Numeric	9	2	Benchmark for Product B
BK_PRODC	Numeric	9	2	Benchmark for Product C

Arithmetic Mean

To calculate an arithmetic mean, you must divide the sum of a set of quantities by the number of quantities in the set.

In the example above, a value of zero in the fields BK_PRODB (benchmark results of Product B) and BK_PRODC (benchmark results of Product C) indicates that a product does not support that feature, and that a comparison should not be made. The values of these ratios should not be added to the sum of the ratios because they do not fall into the set of quantities. These ratios should not be included in the count of ratios since they do not fall into the set.

In the past, to avoid adding these values into the sum and incrementing the count of the ratios, you would have needed to write a UDF similar to the one below and place it in the summary band.

```
*** aritmean()
R_C = 0       && variable for ratio
S_C = 0       && variable for sum of ratios
C_C = 0       && variable for count of ratios
GO TOP
DO WHILE ! EOF()
      R_C = BK_PRODC/BK_PRODA
      IF R_C > 0
            * Increment the count only if there is a non-zero
            * value for the ratio
            C_C = C_C + 1
            * Sum the ratios
            S_C = S_C + R_C
      ENDIF
      SKIP
ENDDO
RETURN S_C/C_C
```

This calculation process is replaced by the report variables in the following table:

Variable	Expression	Initial Value	Reset	Calculate
R_C	RESULTS.BK_PRODC/ RESULTS.PRODA	0	End of Report	Nothing
S_C	S_C +IIF(R_C>0, R_C, 0)	0	End of Report	Nothing
C_C	C_C + IIF(R_C>0, 1, 0)	0	End of Report	Nothing

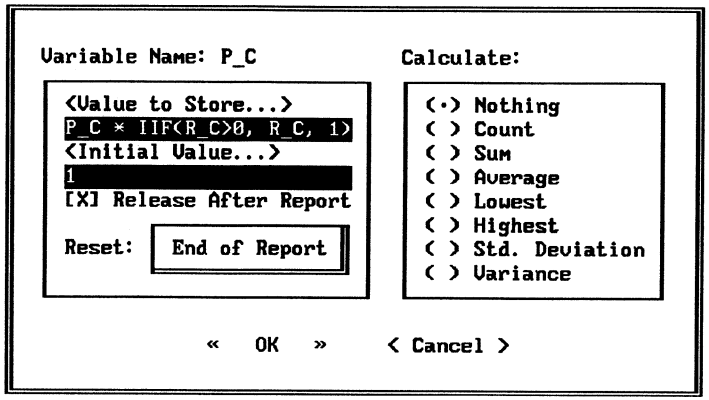
Since R_C is used to define both S_C and C_C, it is important that R_C be initialized first.

The order that variables are displayed in the **Variables** list can affect your output. Variables are evaluated in the order they appear in the list.

You must then create a field in the summary band of the report with the expression S_C/C_C.

Geometric Mean

A geometric mean is the *n*th root of a product of *n* factors. To calculate a geometric mean, another variable, P_C, will need to be created and added to the list of variables in the previous table.



Report Variable P_C

Notice that the initial value of the variable P_C is 1 and not the default 0. If a zero is used as the initial value, the result of the multiplication would be always be zero. Like S_P and C_P, this variable must be initialized after R_P.

You must then create a field in the Summary band of the report with the expression P_P^(1/C_P).

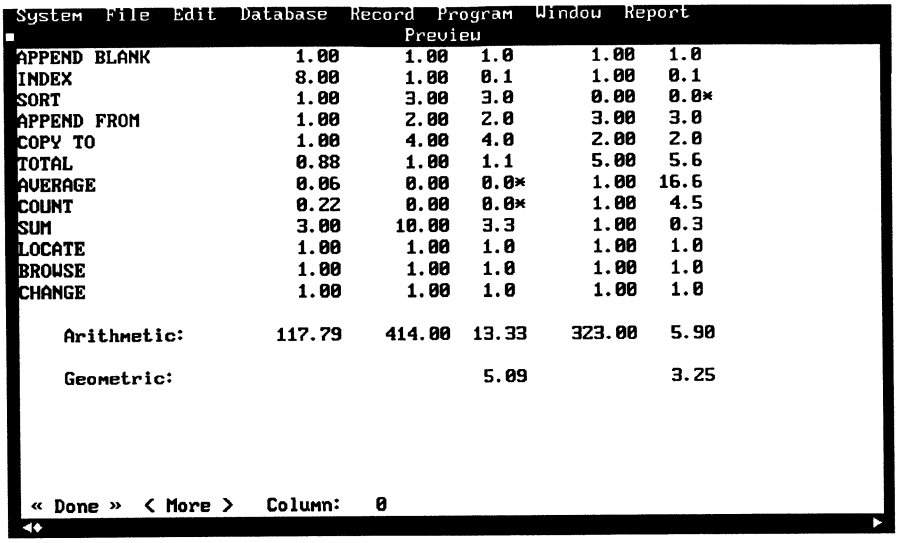
Report Variable Do's and Don't's

When setting up report variables, remember the following:

- The order the variables are initialized is important. If var1 is used to define the value of var2, var1 must be placed in a position prior to var2 so that var1 will be evaluated first.
- The initial value of the variable is important. A default value of 0 will be defined if no other value is specified.
- 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.

Comparison Report Print Out

The picture below shows the report output.



RESULTS0.FRX Output

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, often called an *element*, is referenced by its row and column *subscripts*. Because arrays are stored in memory, they can be accessed and manipulated with exceptional speed.

Array elements can contain any type of data (character, numeric, date or logical). Array elements are initialized to a logical false (.F.) when the array is created.

This chapter includes the following topics:

- 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 SELECT – SQL
- Arrays and FoxPro controls

Creating Arrays

Two commands, DIMENSION and DECLARE, are used to create arrays. These commands are identical in syntax and function and can be used interchangeably. You must specify the array name and size in DIMENSION and DECLARE.

Several FoxPro commands and functions store results to an array. If the array you specify doesn't exist, these commands and functions will automatically create the array:

AVERAGE	ACOPY( )
APPEND FROM ARRAY	ADIR( )
CALCULATE	AFIELDS( )
COPY TO ARRAY	
SCATTER	
SELECT – SQL	
SUM	

Array names can be up to ten characters long and can include alphabetic characters, underscores and numbers. An array name cannot begin with a number or contain embedded spaces.

> Because FoxPro's 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 a single 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 1.

The following examples create a one-dimensional array named DEPTNUMBER with ten rows, and a two-dimensional array named TAXRATES with ten rows and five columns.

```
DIMENSION deptnumber(10)
DIMENSION taxrates(10,5)
```

Multiple arrays can be created with a single DIMENSION or DECLARE command. One command creates the arrays in the example above:

```
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 use.

Function	Description
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 on these functions, refer to the FoxPro *Commands & Functions* manual.

Manipulating Arrays

This section describes how to initialize every element in an array with a single command, how to initialize individual elements and how to change the size or dimensions of an array.

Initializing Entire Arrays

Every element in an array can be initialized to the same value with a single STORE command (or = operator). When COMPATIBLE is set to OFF or FOXPLUS (the default setting) and you include in STORE or = the name of an array without subscripts, every element in the array is assigned 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, the array is released from memory, a single memory variable with the same name as the array is created and the memory 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 with its row and column subscripts, or by a single element number. Array subscripts are numbers or numeric expressions that specify the location of an element in the array. The first subscript specifies the row location of an element, the second subscript 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 identical to its row subscript. An element's number in a two-dimensional array 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 elements numbers for d, e and f are 4, 5, and 6, and so on.

Two useful 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

You can store different values to each array element, or a single value to every element in an array. The STORE command and the = operator are used to assign values to array elements.

To assign a value to an individual array element, include the element's subscript (one-dimensional array) or subscripts (two-dimensional array). In the following examples, the letters A through F are assigned to the elements in two 2-by-3 arrays named ALPHA and BETA. Both STORE and = are used to assign the values A through F to the array elements.

```
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
```

Values can also be assigned to elements by using their element numbers:

```
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 again. The size of an array can be increased or decreased, one-dimensional arrays can be converted to two dimensions and two-dimensional arrays reduced to one dimension.

If you increase the number of elements in an array, the contents of all the elements in the original array are copied in element order to the newly redimensioned array. The additional array elements are initialized to a logical false (.F.).

If you decrease the number of elements, the array is truncated in element number order. Specific rows or columns can be removed from an array with ADEL( ).

Public and Private Arrays

Arrays, like memory variables, can be declared public or private.

Public Arrays

PUBLIC is used to create an array that will be available to any program during the FoxPro session. A public array is also accessible from the Command window.

Arrays created in the Command window are automatically declared public.

If you try to declare an array PUBLIC *after* the array is created, a syntax error message is generated.

Private Arrays

PRIVATE is used to hide an array in a higher level program from the currently executing program and any programs called. PRIVATE does not create an array, it simply makes it possible for you create an array with the same name as an array in a higher level program.

PRIVATE hides the array in the higher level program from the current program or routine. Once the program or routine containing the private array declaration is finished executing, the array in the higher level program with the same name is accessible again.

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. DISPLAY MEMORY displays the contents of the array. The procedure THIS is then executed, and it creates a private array with the same name as the array in the higher level calling program. Note that manipulating the private array does not affect the array created in the calling program — it is hidden from the called procedure.

```
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 array contents
DISPLAY MEMORY LIKE myarray  <-- Show the contents of both arrays
?
RETURN                       <-- Return to 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 standard FoxPro, each array can have a maximum of 3,600 elements.

These are the upper limits for arrays. The number of arrays and elements you can create may be limited by the available memory in your computer.

Passing Entire Arrays to User-Defined Functions

Entire arrays can be passed to a procedure or UDF. When an entire array is passed, it is passed by reference, meaning that the array in the calling program can be changed by the routine.

To pass an entire array to a user-defined function, you must SET UDFPARMS to REFERENCE or preface the array name must be prefaced with an AT symbol (@) to force the array to be passed by reference.

If UDFPARMS is set to VALUE or the array name is placed in parentheses, only the first array element is passed to the routine, and it is passed by value. VALUE is the default setting.

The following program example creates a three element array named MULTIPLY. The entire array is passed to a routine called PRODUCT that multiples the first array element by the second element and places the result in the third array element. An ampersand is placed before the array name to pass the entire array by reference.

After the PRODUCT routine is executed the MULTIPLY array is displayed. Because the array was passed by reference, changes made to the array in the PRODUCT routine 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 product routine with entire array
DISPLAY MEMORY LIKE multiply <-------  Display the contents of array

PROCEDURE product  <-------  Routine called by program
PARAMETER localarray <-------  Local array, 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

Several FoxPro commands facilitate the transfer of data from a database to an array and back again. SCATTER transfers data from a single database record to an array; COPY TO ARRAY transfers data from a series of records to an array. SELECT – SQL can transfer the results of a query to an array. SELECT is discussed in the next section.

Data from an array can be transferred to a single database record with GATHER. APPEND FROM ARRAY adds new records to a database and fills the records with data from an array. INSERT – SQL appends a single new record to a database and fills the record with data from an array.

SCATTER and COPY TO ARRAY transfer data from a database to an array. SCATTER and COPY TO ARRAY differ in these respects:

- SCATTER only transfers data from the current record in the currently selected database. COPY TO ARRAY can transfer data from multiple records in the currently selected database.
- SCATTER has an option (BLANK) to automatically create an array with elements the same size and type as the fields in the database. The array elements are left empty.
- SCATTER has an option (MEMVAR) to automatically create a set of memory variables with the same size, type and name as the fields in the database.

GATHER, APPEND FROM ARRAY and INSERT transfer data from an array to a database. GATHER, APPEND FROM ARRAY and INSERT differ in these respects:

- GATHER transfers data from an array to the current record in currently selected database. APPEND FROM ARRAY appends new records to the end of the currently selected database and then transfers data from the array to the newly appended records.

  INSERT appends a new record and then transfers data from the array to the newly append record. Unlike GATHER and APPEND FROM ARRAY, INSERT can append a record in an unselected database (a database open in a work area other than the currently selected work area).

- GATHER has an option (MEMVAR) to transfer data from a set of memory variables to the current database record.

APPEND FROM ARRAY or INSERT perform faster than APPEND BLANK followed by REPLACE, especially on a network.

For additional information on transferring data between arrays and databases, see the sections for these commands in the FoxPro *Commands & Functions* manual.

Arrays and SELECT – SQL

SELECT is a powerful and versatile command for querying databases. SELECT can output query results to an array.

To send SELECT query results to an array, include the INTO ARRAY clause with an array name. The array is automatically created if it doesn't exist. If the array does exist, it is automatically redimensioned to accommodate the query results.

In this example SELECT directs its query results to an array named RESULTS. SELECT finds records with unique data in the specified fields. Part of the results stored to the array is shown.

```
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
  (   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 further information on SELECT, see the SQL Quiz in this manual and the FoxPro *Commands & Functions* manual.

Arrays and FoxPro Controls

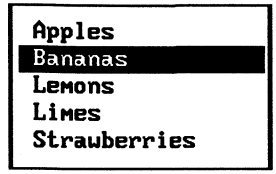
Popups and lists create their options from an array. Popups options can be created from arrays with @ ... GET – Popups. The items in lists created with @ ... GET – Lists are taken from an array.

The FoxPro array functions described below let you dynamically change the size and contents of arrays. Because popups and lists use arrays, lists and popups can also be dynamically changed. Options in a popup or in a list can be inserted and removed by modifying the array. When SHOW GET or SHOW GETS is issued, the popup or list is redisplayed with the new set of options.

The following program example demonstrates how a list can be created from an array. An array is created with 5 elements and then sorted in ascending order with the ASORT( ) function. A list is then created and the elements of the array are used to create the options in the list. When an option is selected in the list a routine is executed that displays the selected option.

```
CLEAR
SET TALK OFF
STORE 1 TO mchoice <-------------------- The first option is selected

DIMENSION scrollarray(5) <-------------- 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, the scrollproc routine is executed.
READ <---------------------------------- Activate the list

PROCEDURE scrollproc <------------------ Done when an option is selected
@12,18 CLEAR
@12,2  SAY 'Your selection: '
@12,18 SAY scrollarray(mchoice) <------- Display the selected option
RETURN .T.
```

Here is the list shown after an option is chosen from the list:




```
Your selection: Bananas
```

10 Low-Level File I/O

FoxPro provides a set of powerful low-level file functions that let you manipulate any type of file. You can create, open, read from and write to any file — the file does not have to be in a FoxPro format. Low-level functions also provide access to your computer's communication ports. Low-level functions utilize FoxPro's highly optimized I/O routines, and thus are extremely fast.

Be careful when using low-level file functions, especially when 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 additional information on each of these functions, refer to the FoxPro *Commands & Functions* manual.

Function	Use	Returns
FCHSIZE( )	Changes the size of a file	Final file size if successful, -1 otherwise
FCLOSE( )	Closes a file opened with FCREATE( ) or FOPEN( )	.T. if successful, .F. otherwise
FCREATE( )	Creates and opens a file	File handle if successful, -1 otherwise
FEOF( )	Determines if the file pointer is positioned at the end of a file	.T. or .F.
FERROR( )	Determines the success of the last low-level file function	0 if no error occurred or the error number
FFLUSH( )	Flushes a buffered file to disk	.T. if successful, .F. otherwise

Function	Use	Returns
FGETS( )	Returns a series of bytes from a file or a communications port	Data from the file
FOPEN( )	Opens a file or communication port for low-level use	File handle if successful, -1 otherwise
FPUTS( )	Writes a character string, carriage return and line feed to a file or a communication port	Number of bytes written if successful, 0 otherwise
FREAD( )	Returns a specified number of bytes from a file or a communications port	Data from the file
FSEEK( )	Moves the file pointer in a file	File pointer position relative to the beginning of the file
FWRITE( )	Writes a character string to a file or a communication port	Number of bytes written to the file if successful, 0 otherwise

Creating Files

FCREATE( ) creates a new file and opens the file for use.

> If a file with the same name already exists, the file is overwritten without a warning. To prevent overwriting an existing file, use FILE( ) to test if the file exists. If the file does exist you can open it with FOPEN( ).

If the file is successfully created, FCREATE( ) returns a unique numeric handle to identify the file. You should store the handle to a memory variable to identify the file in other low-level functions. FCREATE( ) returns -1 if the file cannot be created for any reason.

FCREATE( ) supports an optional numeric argument to specify the DOS attributes of the file you create. The following table lists the numeric arguments and the corresponding DOS attributes.

File Attribute Numbers

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

When a file is opened with a Read Only attribute, you can retrieve data from the file but the file cannot be modified. When a file is opened with a Read/Write attribute, you can retrieve data from the file and the file can be modified (written to). For additional information on file attributes, consult your DOS manual.

To create a file named SAMPLE.TXT with Read Only/Hidden attributes, and save the file handle to a memory variable named SAMPLEHAND, you'd use the following:

```
samplehand = FCREATE('sample.txt', 3)
```

Communication ports cannot be opened with FCREATE. 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 file or port you specify in FOPEN( ) is successfully opened, its handle is returned; if the file or port cannot be opened, -1 is returned.

You can specify the read/write privileges and buffering scheme for the file or port by including an optional numeric argument in FOPEN( ). The following table lists the numeric arguments, privileges and buffering schemes.

Privileges and Buffering		
Numeric Argument	**Read/Write Privileges**	**Buffered/Unbuffered**
0	Read Only (default)	Buffered
1	Write Only	Buffered
2	Read and Write	Buffered
10	Read Only	Unbuffered
11	Write Only	Unbuffered
12	Read and Write	Unbuffered

When a file is opened with buffering, all or part of the file is stored in memory, so it can be accessed many times faster.

Since a buffered file resides in memory, the version on disk is not always current. To assure that the most current version of a file resides on disk, open the file unbuffered. Whenever unbuffered files are modified, they are written to disk. Communication ports should always be opened unbuffered.

The following command opens SAMPLE.TXT with Read and Write privileges, buffered, and stores the file handle to the memory variable SAMPLEHAND:

```
samplehand = FOPEN('sample.txt', 2)
```

Sharing Files on a Network

Files opened with read only privileges by FCREATE( ) and FOPEN( ) can be shared on a network. Files with write or read and write privileges are opened for exclusive use and cannot be shared on a network.

File Pointer

When a file is open, a *file pointer* designates the current byte in the file. The 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 positioned on the first byte when a file is opened with FCREATE( ) or FOPEN( ). The file pointer is moved by FGETS( ), FPUTS( ), FREAD( ) and FWRITE( ). FSEEK( ) lets you move the file pointer to a specific position in the file.

A communications 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 with FREAD( ) and FGETS( ). Include the handle of the file or port in FREAD( ) or FGETS( ).

FREAD( ) FREAD( ) returns a specified number of bytes from a file or port. Data is returned from a file starting at the current file pointer position. The specified number of bytes are returned from a port.

FGETS( ) 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 a carriage return is not encountered first. You can specify a number of bytes other than 254. Line feeds are ignored by FGETS( ).

FREAD( ) is used to sequentially return data from a file or port, while FGETS( ) is used to return a series of lines from a file. Many file formats use a carriage return to specify the end of a line in the file. 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. When FPUTS( ) is used with FGETS( ) you can write to and read from a file line by line.

The following example demonstrates how FGETS( ) can be used to return individual lines from a file.

```
CLOSE ALL
SELECT 0
USE customer                                    Open customer database
STORE RECSIZE( ) TO recordlen                   Record size may be > than 254
COPY TO custtemp.txt DELIMITED WITH BLANK       Delimited file

STORE FOPEN('custtemp.txt') TO custhandle       Open delimited file

IF custhandle < 0                               Can't open the file
      WAIT 'Cannot open file, press a key to exit' WINDOW
      CANCEL                                    Exit this program
ENDIF

CLEAR
DO WHILE NOT FEOF(custhandle)                   Loop through file until end
      @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

Two functions, FWRITE( ) and FPUTS( ), let you write to a file or port opened with write privileges. Include the handle of the file or port in FWRITE( ) or FPUTS( ).

FWRITE( ) FWRITE( ) writes a specified number of bytes to a file or port. Writing begins at the current file pointer position when writing to a file. The specified number of bytes are sent to a communications port.

FPUTS( ) FPUTS( ) is similar to FWRITE( ), except each line written to the file or port is automatically terminated with a carriage return and line feed. When used in conjunction with FGETS( ), a series of lines can be written to and read from a file or port.

Closing Files and Ports

To insure the integrity of data written to and read from a file or port, the file or port must be properly closed. To close a file opened with FCREATE( ) or FOPEN( ), include the file's handle in FCLOSE( ). If the file is properly closed, .T. is returned and the file handle is released.

You can also use CLOSE ALL to close *all* files opened with FCREATE( ) or FOPEN( ). However, CLOSE ALL also 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( ) – Lets you change the size of a file opened with write privileges. Can be used to enlarge or truncate a file's size. FCHSIZE( ) is often used to truncate a file to length 0.
- FEOF( ) – Returns a true value (.T.) if the file pointer is positioned at the end of a file. FEOF( ) always returns true when a port is specified.
- FERROR( ) – Determines the success of the last low-level file function executed. 0 is returned if the previous low-level function executed successfully. A non-zero number is returned if an error occurred. This 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 with this function assures 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. Although HEADER( ) cannot be used with a database file opened with FCREATE( ) or FOPEN( ), the value returned by HEADER( ) can be used 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:

- Drive, directory and file name for each open file
- Handle number for each open file and port
- File pointer position in each open file
- Read and write attributes of each file and port

Low-Level Access to Communications Ports

Communications ports (COM1, COM2, etc.) can be accessed with FoxPro's low-level file functions. Accessing ports lets you read from or write to modems and external devices.

Before you access a port, it must be initialized with the 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 issue RUN MODE in the Command window or in a program.

The following program example demonstrates how a phone number can be dialed by a modem from within FoxPro. This example uses standard Hayes modem commands to initialize the modem, dial the number and hang the modem up. Because of the differences between modems and phone systems, this program may require modification to work for you.

```
SET ESCAPE ON
STORE '5551212' TO phonenumber <-------- Number to dial
RUN MODE COM2:1200,N,8,1,P <-------- Initialize COM2 port

modemhandl = FOPEN('COM2',12) <-------- Open COM2 port

IF FERROR( ) # 0 <-------- Can't open COM2
      WAIT 'Cannot open Com port, press any key to exit' WINDOW
      RETURN
ENDIF

= FPUTS(modemhandl, 'ATDT' + phonenumber) <-- Send phone number

WAIT 'Press any key to disconnect the modem' WINDOW
= FPUTS(modemhandl, 'ATZH') <-------- Send disconnect string
= FCLOSE(modemhandl) <-------- Close COM2
```

The COM2 port is first initialized with the 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, FERROR( ) returns a non-zero value, an error message is displayed in a WAIT window, and the program is exited.

If the port is successfully opened, FPUTS( ) is used to send a dialing command to the port followed by the phone number. The phone number is stored in a memory variable, but could also be in a character database field.

11 Text Merge

FoxPro 2.0 provides several new commands and functions for merging text. Text merge lets you create form letters, documents and programs. You can combine the following with text:

- Contents of database fields
- Contents of memory variables
- Contents of array elements
- Results of functions
- Expressions and results of calculations

These items are referred to as *text merge components*.

For example, a form letter can use the DATE( ) function to place today's date at the top of the letter. Fields from a database can be used to place a customer's name, company and address below the date. The customer's name can be used again in the salutation.

The following table lists text merge commands and functions and their usage.

Command or Function	Use
\ | \\	Output lines of text
TEXT ... ENDTEXT	Output lines of text
SET TEXTMERGE	Enable or disable evaluation of database fields, variables, elements, functions and calculations
SET TEXTMERGE DELIMITERS	Specify delimiters surrounding database fields, variables, elements, functions and calculations
_TEXT	Direct output from \ | \\ and TEXT ... ENDTEXT to a file opened with a low-level function
_PRETEXT	Specify a character expression to preface text merge lines

Merging Text with Text Merge Components

To merge text and text merge components (fields, variables, functions ...) together, the components must be evaluated. For example, if you use DATE( ) to place today's date at the top of a letter, DATE( ) must be evaluated and output.

For a text merge component to be evaluated, three conditions are necessary:

- TEXTMERGE must be ON
- The component to evaluate must be surrounded with the current TEXTMERGE DELIMITERS.
- The component must be placed within TEXT ... ENDTEXT or on a line beginning with \ or \\.

SET TEXTMERGE

The start up default for TEXTMERGE is OFF. To enable the evaluation of text merge components, SET TEXTMERGE ON.

Suppose you open the CUSTOMER database to use its fields in a form letter. When TEXTMERGE is SET ON and a field is surrounded by the text merge delimiters, the *contents* of the fields are merged into the letter.

If TEXTMERGE is SET OFF, text merge components are not evaluated and are literally output (along with the surrounding delimiters). In our example above, the *names* of the fields, not their contents, and the delimiters are merged into the letter.

Text Merge Delimiters

Text merge components must be surrounded by *text merge delimiters* to be evaluated, and to differentiate the components from any text around them.

When you start FoxPro, the default delimiters are sets of double angle brackets << and >>. Suppose you open the CUSTOMER database to use its fields in a form letter. When the fields are surrounded by the text merge delimiters (and TEXTMERGE is SET ON), the contents of the fields are output and merged into the letter. If the fields are not surrounded by the delimiters, the names of the fields, not their contents, are merged into the letter.

SET TEXTMERGE DELIMITERS lets you specify different characters for the text merge delimiters, and restore the default delimiters.

TEXT ... ENDTEXT and \ | \\

In addition to the above requirements, text merge components must be placed within TEXT ... ENDTEXT or on a line beginning with \ or \\ to be evaluated.

The following example demonstrates how SET TEXTMERGE, the text merge delimiters and TEXT ... ENDTEXT work together to perform text merge. The CUSTOMER database is opened. TEXTMERGE is SET ON, enabling the evaluation of the functions and fields used in this example.

SET TEXTMERGE DELIMITERS TO restores the text merge delimiters to the default set of double angle brackets << and >>. TEXT begins the evaluation of the text merge components.

All output from this example is directed to the screen. DATE( ) is evaluated and output, followed the CONTACT, COMPANY and address fields from the database. ALLTRIM( ), PROPER( ) and UPPER( ) remove leading and trailing blanks and capitalize the field contents. The salutation uses the CONTACT field again. The body of the letter then follows as text.

A SCAN ... ENDSCAN loop is used to move through the database. WAIT WINDOW is used to pause program execution before moving to the next record.

```
CLEAR <-------------------------- Clear the screen
SET TEXTMERGE ON <--------------- Enable evaluation of fields, functions, etc.
SET TEXTMERGE DELIMITERS TO <--- Restore default delimiters << >>
SELECT 0
USE customer
SCAN <--------------------------- Traverse 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 you for your interest in our product.
The literature you requested is on its way!

Sincerely,

Fox Software

ENDTEXT
WAIT WINDOW <-------- Pause before next record
CLEAR <-------- Clear the screen
ENDSCAN <-------- Move to the next record
USE <-------- Close the database
```

Here is the output from the first record:

```
04/11/91

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,

Fox Software
```

If this example is modified so that TEXTMERGE is SET OFF or TEXT ... ENDTEXT is removed, output looks like this:

```
<<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,

Fox Software
```

\ | \\

TEXT ... ENDTEXT is used in the above example to specify where text merge starts and end. Single (\) and double backslashes (\\) can also be used to specify where text merge occurs. Single and double backslashes facilitate text merge without use of TEXT ... ENDTEXT.

If the first character of a program or Command window line is a single backslash \, the content of the line is evaluated as if the line has been placed between TEXT and ENDTEXT. A line feed and a carriage return is sent as output before the line is evaluated. If a double backslash \\ is used at the beginning of the line a line feed and a carriage return are not sent before the output.

Here is the example from before with \ and \\ replacing TEXT ... ENDTEXT. The output is identical. Note how \\ is used to place the contents of the STATE and ZIP fields on the same line as the ADDRESS2 field.

```
CLEAR                                  <- Clear the screen
SET TEXTMERGE ON                       <- Enable evaluation of fields, functions, etc.
SET TEXTMERGE DELIMITERS TO            <- Restore default delimiters < >
SELECT 0
USE customer
SCAN                                   <- Traverse 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,
\
\Fox Software
\
WAIT WINDOW                            <- Pause before next record
CLEAR                                  <- Clear the screen
ENDSCAN                                <- Move to the next record
USE                                    <- Close the database
```

_PRETEXT

\ and \\ allow use of _PRETEXT. _PRETEXT lets you indent and preface lines beginning with \ and \\. When a character expression is stored to _PRETEXT, it is output before the text or text merge components. Tabs cans be stored to _PRETEXT to create indentation in program templates.

Directing Output to the Screen, Windows and Files

By default, output from text merge is directed to the screen. By including options with SET TEXTMERGE, output to the screen can be suppressed, and you can direct output to windows and files. The _TEXT system memory variable lets you direct output to files opened with FCREATE( ) or FOPEN( ).

Screen Output

If text merge output is being directed to the screen, you can suppress the screen output by including the NOSHOW option with SET TEXTMERGE. To enable output to the screen again, include the SHOW option with SET TEXTMERGE.

If output is being directed to the screen, this command suppresses it:

```
SET TEXTMERGE NOSHOW
```

Screen output is suppressed until you issue the command:

```
SET TEXTMERGE SHOW
```

Window Output

Output from text merge can be directed to a window. Including the WINDOW clause in SET TEXTMERGE directs text merge output to the specified window. Text merge output can be directed to a window that is not active or visible. Once output is directed to a window, the NOSHOW and SHOW options can be used in SET TEXTMERGE to suppress or enable output to the window.

This short program example 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 or active. ACTIVATE WINDOW DATETIME then 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
```

File Output

Text merge output can also be directed to a file. Text merge can be sent to a file in two ways:

- Include the name of a file where output will be directed in the TO <file name> clause of SET TEXTMERGE.
- Store the file 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. The file can be manipulated with the low-level file functions. See the Low-Level File I/O chapter in this manual for more information about FoxPro's low-level file functions.

To close the file opened with SET TEXTMERGE, you can issue the command SET TEXTMERGE TO without a file name, or you can include the file's handle in FCLOSE( ).

If a file is opened with FCREATE( ) or FOPEN( ) and you store its handle to _TEXT, text merge is directed to the file when TEXTMERGE is SET ON.

Program Templates and Programs

A powerful capability of text merge is program generation. Because FoxPro programs are text files, text merge can create programs. Commands and functions can be text merge components.

The following example demonstrates how a program template is created using FoxPro's text merge capabilities and commands and functions. Two procedures are shown from the GENSCRN program that generates screen program code from information stored in a screen database. The first procedure creates the program code for push buttons, the second creates program code for radio buttons.

After a file is created to store the generated screen code, procedures like the following output screen program code to the file. These procedures use fields from the screen database to supply information for the commands GENSCRN creates.

In the two procedures that follow, fields from the screen database are surrounded by the default textmerge delimiters (<< and >>). For example, <<VPOS>>, <<HPOS>>, <<NAME>>, etc. are all fields from the screen database. The field 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 screen program code that 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 unittype ;
PICTURE "@*RVN \<Area;\<Length;\<Mass;\<Speed;Tempe\<rature;\<Time;\<Volume" ;
        SIZE 1,15,0 ;
        DEFAULT 1 ;
```

12 Customizing Help

Help text appropriate to the current program environment is said to be *context-sensitive.*

As you know, FoxPro provides context-sensitive help for the user. This chapter explains how you can create context-sensitive help for your own applications.

Topics covered include:

- Help database requirements
- FOXHELP default file
- Design of FOXHELP
- Specifying another help database
- Narrowing available topics
- More ideas
- Help file codes

Context-Sensitive Help in FoxPro

In FoxPro, pressing F1 or Alt+click on a menu, window or dialog brings up context-sensitive help. Selecting text and pressing F1 brings up context-senstive help for the selected text. You can also type the following in the Command window to access a specific help topic:

```
HELP <topicname>
```

You can use a variety of methods to create context-sensitive help for custom applications. Help text can be displayed based on:

- A specific topic name
- The screen field being READ at the moment
- The program being executed
- Conditions specified in SET HELPFILTER
- Any combination of the above

FOXHELP – Default Help File

This section explains how FoxPro's default help file, FOXHELP, is designed. Your help database can be patterned after FOXHELP, or developed in a different way. The description here is intended as an illustration.

Because a help file is a database, you can create custom help by adding information to and removing information from FOXHELP, or by creating a new help database.

To view or edit the FOXHELP.DBF database, you first SET HELP OFF. Then you can open FOXHELP and browse it as you would any other FoxPro database.

The structure of FOXHELP is shown 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 are used to categorize the help topics.

A table of the codes that appear in the CLASS field is located at the end of this chapter. The categories are used by the program HELPTREE.MPR, which is located in the GOODIES directory.

Help Database Requirements

Help databases can have a maximum of 32,776 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.

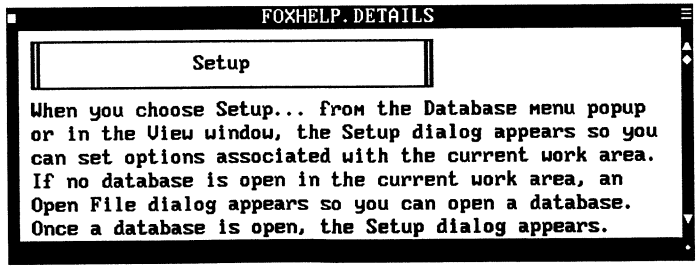
Beyond these requirements, you can add as many additional fields as you wish. Adding fields to your help database provides greater latitude for determining which help text to display for any given context. It doesn't matter what you name these additional fields, but it is a good idea to name them meaningfully.

FOXHELP Topics

FOXHELP includes general topics, interface topics, and command/function/system memory variable topics. In the Topics panel of the Help window, general topics are preceded by a triangular marker (►), formed by pressing Alt+16, and interface topics by a square bullet (■), formed by pressing Alt+254. The other types have no symbol preceding the topic name.

FOXHELP Details

Help information is entered for a topic as text in the FOXHELP memo field named DETAILS. The illustration below shows the memo window from an actual FOXHELP memo field, once you've SET HELP OFF.



Memo Window in FOXHELP Database

Once the information is entered for the topic, the memo window is closed and the information is added to the database.

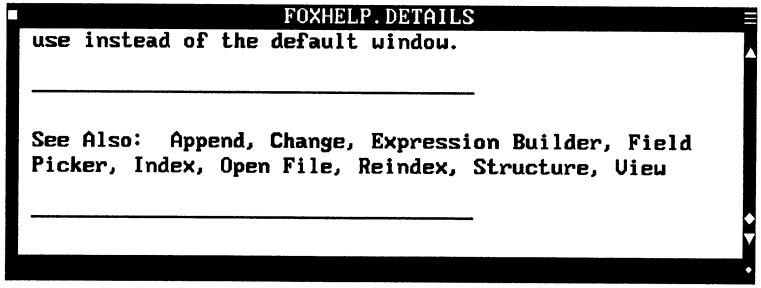
FOXHELP Cross References

"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 adhere to the following rules.

At the very end of the memo field, include:

- The phrase See Also, followed by a colon, and optionally space(s).
- A comma-delimited list of the topics that you want to appear on the **See Also** popup.
- A carriage return and a blank line to signal the end of the list.

Case does not matter in the See Also list. Leading and trailing spaces are trimmed from each referenced topic. Cross references for the Setup topic are shown below.



See Also References in the Setup Topic

Interface topics in the See Also list are *not* preceded 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.

In FoxPro's default architecture, interface help topics only point to other interface topics. Thus, you will not find cross references to a command or function name in the See Also list of help for a window or menu name in the FoxPro interface.

Tailoring the Help Display

When the user chooses **Help...** from the **System** menu popup or types HELP in the Command window, the Help window appears with the Topics panel frontmost. The user can scroll through the list to find the desired topic, or type a letter to move to the first topic starting with that letter. Choosing a topic brings up information about the selected topic.

Specifying a Help Database

By default, FoxPro uses the FOXHELP database. To indicate that another database will be used to provide help, execute the following command in a program or type 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 the FOXHELP database as the help database, issue this command:

```
SET HELP TO FOXHELP
```

Issuing the command SET HELP TO without a <filename> also causes FoxPro to use the help database named FOXHELP.DBF.

Narrowing Displayed Help Topics

Once the help database has been specified with the SET HELP TO command, topic selection can be further narrowed for the user. Two commands are used to control how help topics are selected for display to the user:

- For topic selection by topic name
  `SET TOPIC TO <expC>`
- For topic selection based on a logical expression
  `SET TOPIC TO <expL>`
- To display 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 is to be displayed. The topic name <expC> is evaluated when help is requested, and the help database is searched to find the record whose first field, the TOPIC field, matches <expC>. The search is case insensitive.

When a match is found, the contents of the second field in the help database — the memo field — is displayed as the help text.

For example, you could use the following commands to specify that help information stored with the Customers topic should be displayed when the **Help...** push button is chosen in the CUSTOMERS screen:

```
SET TOPIC TO 'CUSTOMERS'
HELP
```

If you want to select a topic using the form of HELP <topic name>, such as

```
HELP Customers
```

you must include the square bullet (and a space), if ■ appears at the beginning of the topic name in the help database, such as

```
HELP ■ Customers
```

Topic Selection Based on a Logical Expression

When HELP is issued after you've SET TOPIC TO <expL>, <expL> is evaluated and help information is displayed from the first record in the help database for which the expression is true.

Suppose you want to provide help on a field-by-field basis when users are entering data into the CUSTOMER database. For instance, if they are entering data into the field named COMPANY, the help text would assist them in entering the proper data in the COMPANY field.

To do this, you would first add the names of each field in COMPANY into a new field named GETFIELD in the HELPINFO database, and add the associated help text for each to the memo field HELPTEXT.

You would activate the help facility with:

```
SET HELP TO helpinfo
SET TOPIC TO UPPER(GETFIELD) = UPPER(VARREAD())
```

where UPPER(GETFIELD) = UPPER(VARREAD( )) is the logical expression.

UPPER( ) is used to ensure that a case insensitive search is performed.

At the beginning of the input routine, an ON KEY LABEL statement would be used to assign a context-sensitive help routine to the F1 key (or to another key):

```
ON KEY LABEL F1 HELP
```

ON KEY LABEL traps the specified key immediately, then executes the command. An ON KEY LABEL trap could also be followed with SET TOPIC TO <expL> to locate a specific field value. SET TOPIC TO is evaluated when help is requested.

You can also use ON KEY LABEL to trap for a mouse click that will activate help.

```
ON KEY LABEL LEFTMOUSE HELP
```

This traps for a click of the left mouse button. See ON KEY LABEL in FoxPro *Commands & Functions* for further information on the use of this command, including a table of the Key Label Assignments you can use with ON KEY LABEL.

You can also use the function PROGRAM( ) to return the name of the program being executed to your help routine. See FoxPro *Commands & Functions* for information on PROGRAM( ).

Note that specifying SET HELP TO <filename> is enough to ensure that when a user presses F1, your help database <filename> will be brought up. And pressing F1 in a given window will then bring up 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 for the user, rather than to zero in on a specific topic name. Its syntax is:

```
SET HELPFILTER [AUTOMATIC] TO <expL>
```

The help topics are filtered based on the logical expression <expL>. Only those records in the help database that meet the condition specified by the filter expression <expL> appear in the Help window.

Issuing SET HELPFILTER TO with no argument removes the help filter condition. If you include AUTOMATIC directly before TO in the command, the help filter condition is removed immediately after the Help window is closed.

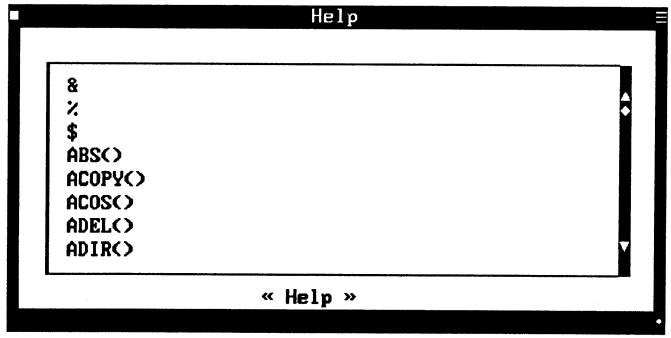
The filter expression <expL> typically includes a field from the help database. Any field from the help database may be included in the filter expression.

HELPFILTER Examples

Assume you are using the FOXHELP database. Issuing the following commands

```
SET HELPFILTER TO 'Function' $ class
HELP
```

results in a Help topics display as shown below:



Restricted Help Topics List

This view of the FoxPro Help window has been limited to topics whose CLASS field includes the term "Function". In the above example, you would need to issue the following command to remove the filter condition:

```
SET HELPFILTER TO
```

The same effect could also be achieved by including AUTOMATIC in the command line:

```
SET HELPFILTER AUTOMATIC TO 'Function' $ class
```

You need not limit the HELPFILTER expression to one restriction. For example, the command:

```
SET HELPFILTER TO 'Function' $ class AND 'nu' $ class
```

would further restrict the topical display to Function names that are numeric in type. See the end of this chapter for a listing of all the abbreviations used in the CLASS field of the FOXHELP.DBF file.

Context-Sensitive Help in Windows

When a user presses F1 in a window in your application, FoxPro displays context-sensitive help for the window title as specified with the TITLE clause. (This assumes you have not redefined the F1 key.) In other words, if the window title is "CUSTOMER", FoxPro displays help information for the topic "CUSTOMER".

If a window has no title, FoxPro displays context-sensitive help for the window name (with 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 last viewed. You cannot explicitly specify the location of FoxPro system windows (such as Help).

To specify the coordinates where your custom help will be displayed, you need to use DEFINE WINDOW, as explained in FoxPro *Commands & Functions*. You can then place your help in this defined window and specify the coordinates of the defined window so that it appears in a given place on screen.

The following commands define a window named RED and activate the help database HELPER.DBF 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 the help database, and because any logical expression can be used to select help topics, you're limited only by your imagination in setting up a help system for your applications.

For instance, you could:

- Create one or many variables in your programs to control the behavior of your help system and assign values to these variables based on the current operating mode of your program
- Provide different amounts of detail in your help files for novice users than you do for experienced users
- Permit users to access help only if they entered an appropriate password

You could develop much more elaborate schemes of help from this base, using the customizing features available through FoxPro's Menu Builder and Screen Builder. For instructions on the use of these power tools, see the appropriate chapters in the FoxPro *Interface Guide.*

Help File Codes

The following table contains the general categories and two-letter abbreviations used in the CLASS field of the FOXHELP.DBF help file. These abbreviations can be used 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 be displayed. If one does not meet the filter condition, choosing it from the popup will display a "No help found for..." message.

Category	Abbreviation
General	General
What's New	wn
Compatibility	cm
Configuration	cf
Error Messages	em
Commands	Command
Database (Fields, Indexes, Records, Relations)	db
Environment (SET Commands, Screen, Keys, ...)	en
Errors and Debugging	er
Event Handlers (ON ERROR, ON KEY, ...)	eh
File Management	fm
Keyboard and Mouse	km
Memory Variables and Arrays	mv
Menus and Popups (System Names)	mp
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
Date and Time	dt
Logical	lo
Database (Fields, Indexes, Records, Relations)	db

Category	Abbreviation
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	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 may 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.

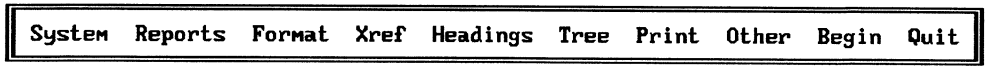
F6-Read Defaults

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 filename 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.



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 may 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

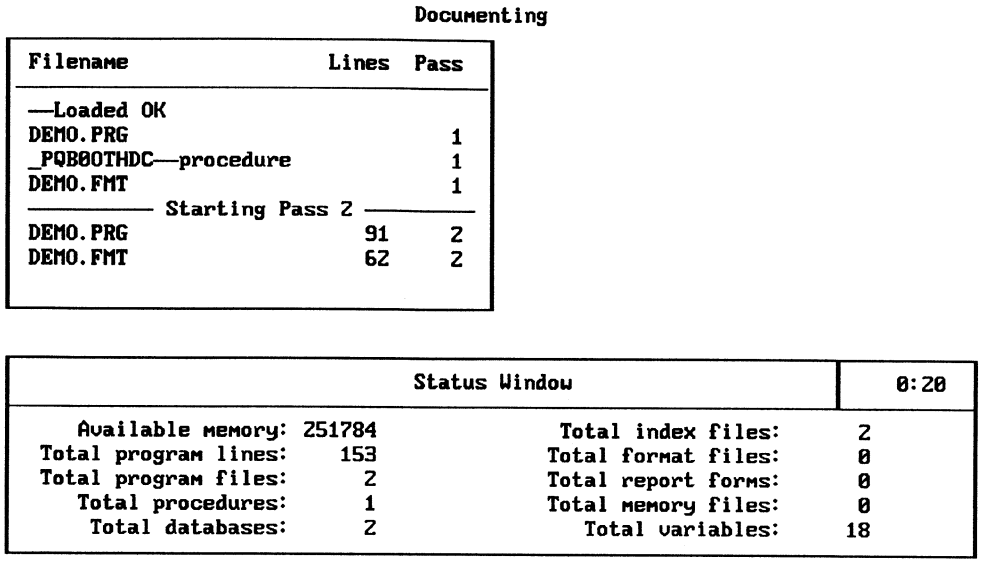
Enter the name of the main program file in your application or the name of your project. If the file you specify cannot be found, or if the FoxDoc files are not in the directories that you specified, FoxDoc returns an error message and allows you to correct the information or exit from FoxDoc.

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 may 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.



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.

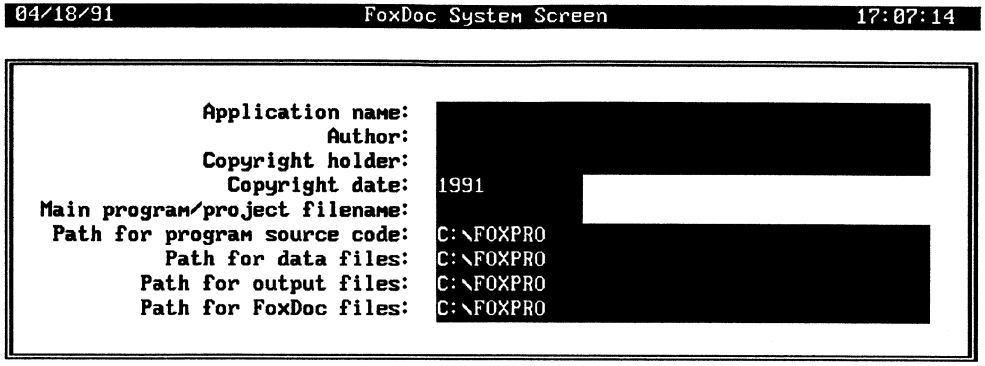
If you choose any option that modifies the source code or if you choose to display the cross-reference report, FoxDoc displays information in the lower portion of the screen indicating the following:

- 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 let's you specify system-level options.



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 may 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 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 you 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 may 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.)

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.

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.

Path For FoxDoc Files

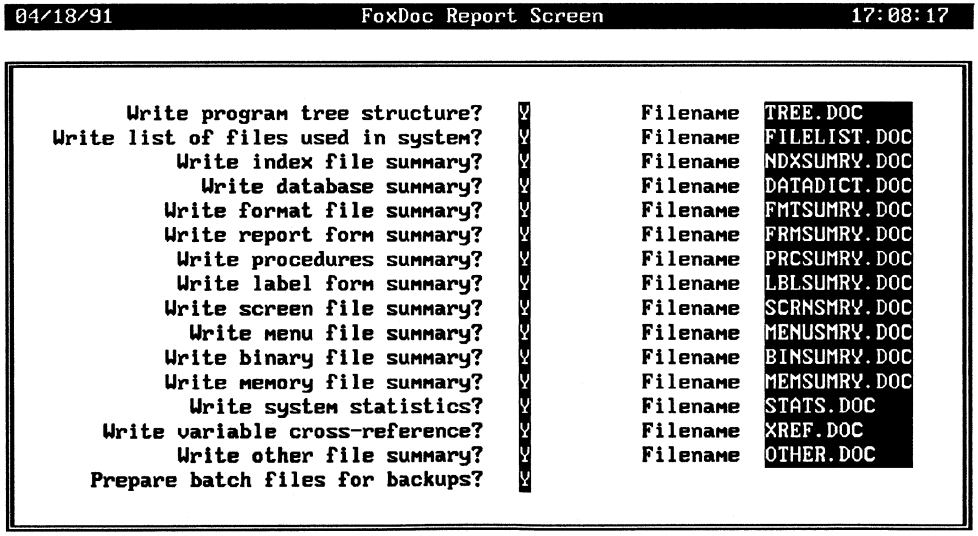
Enter the complete path name where the FoxDoc files are located. The path should be a valid MS-DOS paths, with or without drive designations. The final backslash may be omitted, for example, C:\FOXDOC, C:\FOXDOC\ and \FOXDOC are all valid path names.

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.



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 in 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

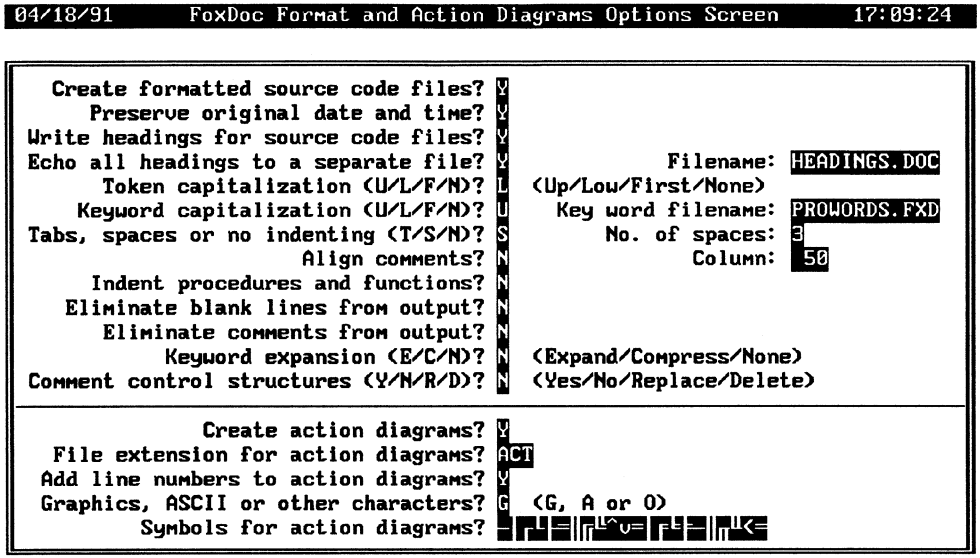
This option allows you to display and echo to file a summary of files other than those covered by separate documentation reports used in the application. FoxDoc watches for references to these files in your programs and prepares a report showing the file name and the program files that use it.

Preparing Batch Files

While FoxDoc is documenting your system, it can produce several 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.



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> WHIL   <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 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

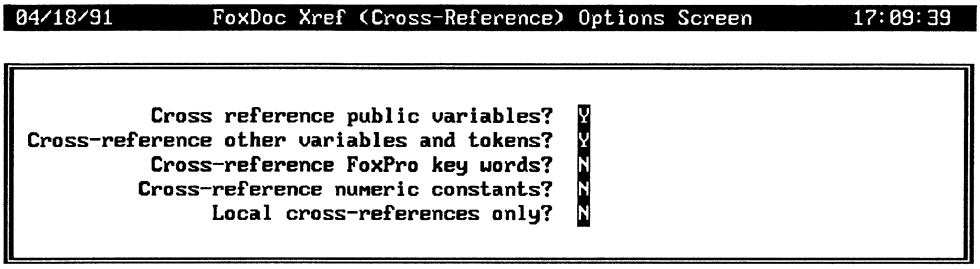
Different extended ASCII characters are used to mark the different control structures. Loops (DO WHILE, FOR) use the double-line vertical bar, while conditional statements (IF, DO CASE) use a single vertical bar. The structures are distinguished further by the horizontal symbol used to connect the control 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.



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

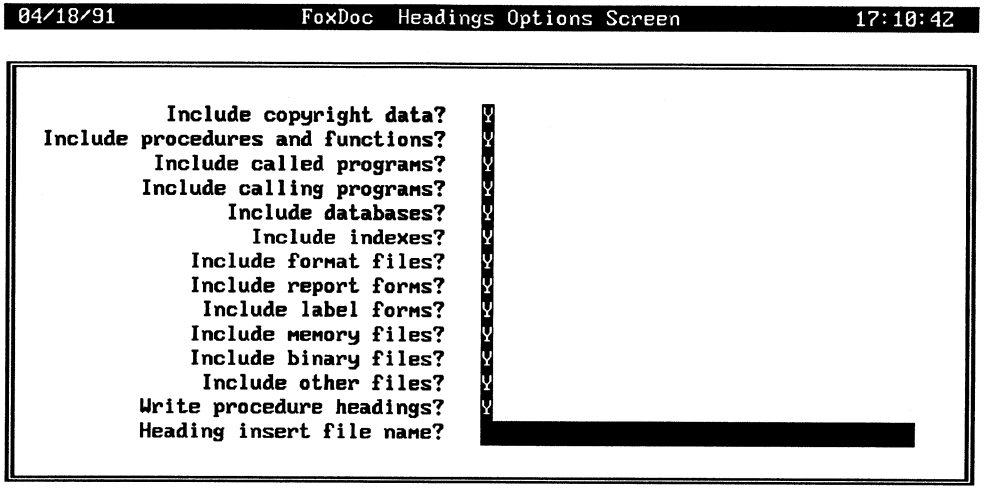
This option determines whether FoxDoc will prepare a local cross-reference report or the standard global system-wide cross-reference report. The local report shows cross reference information for a single module at a time. In other words, the report will show where the variable "APPLE" was used in a 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.



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 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

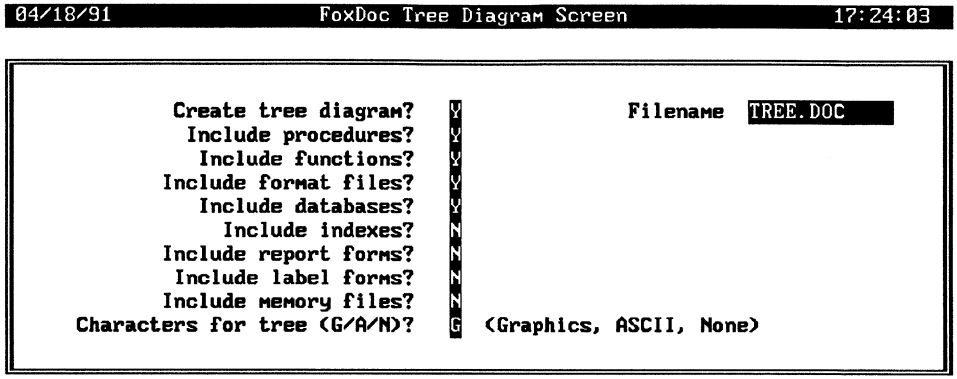
Heading Insert File Name

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.



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 are 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

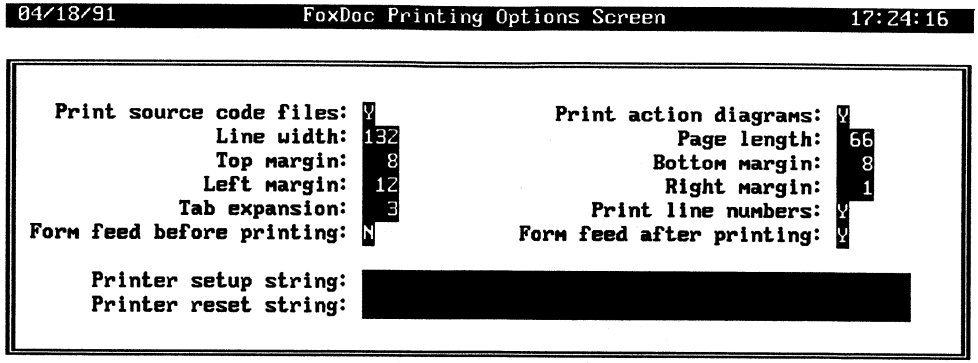
There are two new function key options associated with this screen: **F2-List** and **F9-Print now**. Pressing F2 or choosing **F2-List** with your mouse will cause a list of printer types to appear. Pressing F9 or choosing **F9-Print** now will allow you to print files without going through the full 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 create your own file containing filenames to print also. You can also use this feature to print documentation files in a nicely-formatted way.



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

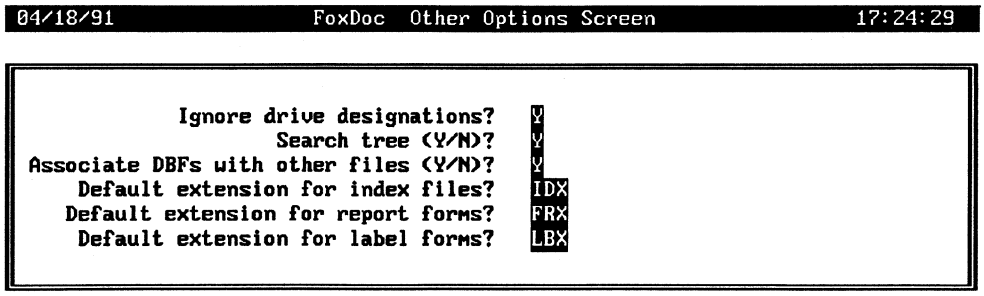
This option allows you to specify setup codes to be sent to your printer before printing begins. The default codes shift an Epson and many other printers into compressed print mode. The format for setup codes is three-digit decimal numbers separated by backslashes (\). For example, the default code is \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.



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, 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 report 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 CAPITAL 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 wish 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 /fsystem1.fxd
FOXDOC /x /fsystem2.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 *nnnn* 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:\FOXPRO\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 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 DOS PRINT utility for background printing. Thus, PRINT must be available somewhere on the path.

Keyword 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_HANDLR         (procedure in TODOPRC.PRG)
*:               : HELPEDIT          (procedure in TODOPRC.PRG)
*:               : TDDEFLT.PRG
*:               : ERRORMSG          (procedure in TODOPRC.PRG)
*:               : SETDATE.PRG
*:               : TDLOGO.PRG
*:               : TDSETUP.PRG
*:               : TDINPUT.PRG
*:
*:   Memory Files: ACCESSES.MEM
*:               : DEFAULT.MEM
*:
*:     Documented: 03/02/88 at 18:39
*:**********************************************************************
PARAMETERS c_line                && command line
*# DOCMACRO s_tdfile    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


                 Index           Report          Label           Memory
Databases        Files           Forms           Forms           Files
-------------  --------------  --------------  --------------  --------------

HELP.DBF         HELPKEY.IDX     SUBREPT.FRM                     ACCESSES.MEM
TODO.SKL         SUBJECT.IDX     TDSUMIN.FRM                     DEFAULT.MEM
TODO.DBF         TODODD.IDX      TDDETIN.FRM                     LASTFILE.MEM
SUBJECT.DBF      &NTXNAME                                        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. Kennamer
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_HANDLR (procedure in TODOPRC.PRG)
    ADDRLIST.PRG
        SETCOLOR (procedure in TODOPRC.PRG)
    NUMLIST.PRG
         SETCOLOR (procedure in TODOPRC.PRG)
HELPEDIT (procedure in TODOPRC.PRG)
TDDEFLT.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_HANDLR (procedure in TODOPRC.PRG)
        ADDRLIST.PRG
            SETCOLOR (procedure in TODOPRC.PRG)
        NUMLIST.PRG
             SETCOLOR (procedure in TODOPRC.PRG)
    F3_HANDLR (procedure in TODOPRC.PRG)

and so forth
```

Procedure and Function Summary

```
System: ToDo -- To Do Management System
Author: Walter J. Kennamer
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_HANDLR

      Called by: TODO.PRG
      Called by: TDDEFLT.PRG
          Calls: ADDRLIST.PRG
          Calls: NUMLIST.PRG
   Contains: F3_HANDLR
      Called by: TODO.PRG
      Called by: TDDEFLT.PRG
          Calls: SETCOLOR           (procedure in TODOPRC.PRG)
   Contains: F4_HANDLR
      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. Kennamer
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   HSCRNNUM     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+hscrnnum+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
```

```
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
HELPMSG	M	10	0	HELP.DBF
HINPUTVAR	C	12	0	HELP.DBF
HSCRNNUM	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+hscrnnum+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
      : TDREINDX.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
       : TDREINDX.PRG
       : TDREPAIR        (procedure in TDFIX.PRG)
 ---------------------------------------------------------------------------

TODODD.IDX -- Indexed on: DTOS(date_due)+priority+caltime
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

```
System: ToDo -- To Do Management System
Author: Walter J. Kennamer
03/02/88    18:55:50
Report Form File Summary

--------------------------------------------------------------------------------

2 report forms in the system
   SUBREPT.FRM
   TDSUMIN.FRM
 ------------------------------------------------------------------------------

SUBREPT.FRM                                   Last updated:  08/24/87 at
10:14    Summary report?  No
   Eject page before printing?  No          Eject page after printing?  Yes
   Double space report?  No                 Plain page?  No
   Left margin:    8                        Right Margin:    0

   ---------------------------------------------------------------------------
                                Report Contents
   ---------------------------------------------------------------------------
 No.   Field                              Length    Decimals    Totaled?
 ---   ---------------------------------  ------    --------    --------

  1    Subject                              40          0           No
  2    Subcode                              25          0           No
                                           -----
                                            65
                                           =====

   ---------------------------------------------------------------------------
                                Report Layout
   ---------------------------------------------------------------------------
-
Page No.     1
00/00/00



                                        Subject Codes

Subject                                 Charge Code
1                                       2


   ---------------------------------------------------------------------------
            Database and Program References
   ---------------------------------------------------------------------------

   FoxDoc could not find an associated database

   Used by: SUBREPT          (procedure in SUBJECT.PRG)
```

TDSUMIN.FRM Last updated: 09/01/87 at 15:17

Summary report? No
Eject page before printing? Yes
Eject page after printing? Yes
Double space report? Yes
Plain page? No
Left margin: 1
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
  TDDETIN.FRM
  TDSUMIN.FRM

Memory files:
  ACCESSES.MEM
  CLIP.MEM
  DEFAULT.MEM
  LASTFILE.MEM
  PRINTER.MEM
```

Advanced Topics

14 Customizing FoxPro

When you install FoxPro, it is configured to work the way most users will prefer. Many of FoxPro's settings can be adjusted to fit your needs or improve FoxPro's performance.

This chapter describes the simple but flexible procedures you can use to modify FoxPro for your particular hardware configuration or your own personal tastes. Topics covered include:

- Configuring startup files
- SET commands
- Startup switches
- Resource files
- Special configuration items
- Function keys
- Colors

Startup Files

One easy way to establish FoxPro's working environment automatically is through the startup configuration files:

- CONFIG.SYS
- CONFIG.FP

CONFIG.SYS

The system configuration file, CONFIG.SYS, contains certain commands which are checked and executed when you start up your computer. These commands change your computer's default configuration. For example, you can increase or decrease the amount of memory that's set aside to hold data when reading or writing from disk, or you can add a device driver that allows extended memory to be used as expanded memory (EMS).

CONFIG.SYS is not a FoxPro file. It's a file that your computer's operating system uses to establish the working environment. Because FoxPro interacts with this environment, you must be sure that certain settings are properly established. Two of the CONFIG.SYS statements are of immediate importance to FoxPro:

- BUFFERS =
- FILES =

Buffers

The BUFFERS statement contains the number of disk buffers that the operating system sets aside in memory when your computer is started. A disk buffer is a block of memory (typically 512 bytes) that the operating system uses to hold data when reading and writing from disk. For best performance with FoxPro, the CONFIG.SYS file should contain a BUFFERS statement with a number between 20 and 40.

Files

The FILES statement sets the number of files that the operating system can open and access at one time. This number is directly related to the number of files that FoxPro will be able to open. The FILES statement in CONFIG.SYS should *always* be *at least* 10 more than the greatest number of files that will be open at one time. We recommend a *minimum* setting of 40.

Every file that you open in FoxPro uses a file handle. When you close the file it "gives" the file handle back. If you open 10 database files and five of the database files have memo files associated with them, 15 file handles will be in use. If seven of the files have two indexes each, an additional 14 file handles will be used for a total of 29. If your resource (FOXUSER) file and help file are in use an additional five file handles will be used (FOXUSER.DBF, FOXUSER.FPT, FOXUSER.CDX, FOXHELP.DBF and FOXHELP.FPT). .CDX files use one file handle each regardless of the number of tags contained in each.

Two editing sessions will take two file handles each. Editing sessions take a file handle for the file itself and one for the temporary file that is opened for each editing session. The temporary file is closed at the end of the editing session when the file being edited is closed.

See the reference material that accompanied your computer's operating system for complete details on CONFIG.SYS and the various statements it may contain.

Two DOS environment variables can also be set in CONFIG. These are FOXPROSWX and FOXPROCFG, described in the section Startup Switches later in this chapter.

CONFIG.FP

FoxPro allows you to modify file allocation space and initial SET command defaults with a configuration file named CONFIG.FP.

When FoxPro is started, it automatically searches the current working directory for a CONFIG.FP file. If one isn't found, FoxPro searches the DOS path for CONFIG.FP. (Your DOS manual provides a description of how to set a DOS PATH.) If no CONFIG.FP file is found, FoxPro's built-in settings are used.

When you start FoxPro, you can include an optional switch to specify the name of the configuration file to be used:

FOXPRO -C<pathname>\<file>

Alternately, you can include the following line in the AUTOEXEC.BAT file to specify a configuration file:

SET FOXPROCFG=<pathname>\<file>

Examples

To use MYCONFIG.JJJ in the FOXFILES directory on the C: drive:

```
FOXPRO -CC:\FOXFILES\MYCONFIG.JJJ
```

To use the CONFIG.FP file in the TESTFILE directory on the D: drive:

```
FOXPRO -CD:\TESTFILE\CONFIG.FP
```

If you do not include the <file>, you must include a "\" after the <pathname> for FoxPro to recognize an existing CONFIG.FP. To use the CONFIG.FP file in the FOXFILES directory on the C: drive:

```
FOXPRO -CC:\FOXFILES\
```

If you use SET to specify a parameter in your configuration file, then start FoxPro using a conflicting command line switch, the command line specification will take precedence.

Changing Configuration Settings

It is simple to change settings in the configuration file using any text editor, especially the built-in FoxPro editor. The CONFIG.FP text file contains one or more lines with the following format:

```
<item> = <value>
```

Examples:

```
TALK = OFF
BELL = OFF
LABEL = LBL
BLOCKSIZE = 40
```

In the example configuration above, the interactive talk feature is turned off, the bell does not sound when fields are filled, label definition files have an extension of .LBL (the default is .LBX), and new memo files are written in block sizes of 40 bytes.

Numeric configuration options are checked to ensure that they fall within the range of permissible values for that item. Incorrect values are automatically changed to the nearest permissible value. FoxPro will ignore any CONFIG.FP statements that are not supported, rather than report an error.

After your change your CONFIG.FP file, you must QUIT FoxPro and start again before any of these changes take effect.

SET Commands

FoxPro allows you to customize your working environment in a number of ways. You can make changes automatically when FoxPro is started, or you can make temporary modifications in the middle of a FoxPro session using SET commands. Some of the SET commands act as on/off switches that activate or deactivate various features (e.g., SET TALK OFF), while others establish values that are used by another command (e.g., SET PATH TO ...).

SET commands to include in CONFIG.FP for compatibility with FoxBASE+ are discussed in the Compatibility chapter of this manual. You change the parameters controlled by the SET commands through various panels of the View window or by typing the SET commands directly in the Command window. Any changes that you make through the interface in this fashion are temporary — they remain in effect until they are changed again or until you exit from FoxPro.

With the SET commands you can modify environment settings such as the default drive, color settings, a search path, etc. If you type the keyword SET without any parameters, the View window comes forward. Each panel of the View window contains SET parameters that you can change to meet your present needs. To display a panel, just choose its text button (**View**, **On/Off**, **Files**, **Misc**).

See the FoxPro *Commands & Functions* manual for complete information on all the SET commands. For details on the options that can be set through the View window, see the appropriate description in the Window Menu chapter of the FoxPro *Interface Guide*.

SET Commands in CONFIG

If you find that you frequently adjust various SET options in a certain manner, you may want to designate automatically the default SET values at startup.

In addition to the special CONFIG.FP statements (covered in the next section), the CONFIG file can contain startup defaults for nearly all the SET commands. For details on the meaning of the following items, refer to the FoxPro *Commands & Functions* manual.

SET Command Values

<item>	<value>	default
ALTERNATE	<filename>	
ALTERNATE	OFF | ON	OFF
ANSI	OFF | ON	OFF
AUTOSAVE	OFF | ON	OFF
BELL	ON | OFF	ON
BELL	<19 to 10,000, 2 to 19> (frequency, duration)	512, 2
BLINK	ON | OFF	ON
BLOCKSIZE	<expN>	64
BORDER	<attribute>	SINGLE
BRSTATUS	OFF | ON	OFF
CARRY	OFF | ON	OFF
CENTURY	OFF | ON	OFF
CLEAR	ON | OFF	ON
CLOCK	OFF | ON	OFF
CLOCK	<coord>	0, 69
COLOR	<color attrib>	
COLOR OF BOX	<color attrib>	
COLOR OF FIELDS	<color attrib>	
COLOR OF HIGHLIGHT	<color attrib>	

SET Command Values		
<item>	<value>	default
COLOR OF INFORMATION	<color attrib>	
COLOR OF NORMAL	<color attrib>	
COLOR OF MESSAGES	<color attrib>	
COLOR OF TITLES	<color attrib>	
COLOR OF SCHEME <expN>	<ColorPairList>	Current settings
COLOR SET	<ColorSetName>	DEFAULT
COMPATIBLE	OFF|ON (FOXPLUS/DB4)	OFF (FOXPLUS)
CONFIRM	OFF|ON	OFF
CONSOLE	ON|OFF	ON
CURRENCY	<char>	"$"
CURRENCY	<position>	LEFT
CURSOR	ON|OFF	ON
DATE	<format>	AMERICAN
DEBUG	ON|OFF	ON
DECIMALS	<0 to 18>	2
DEFAULT	<drive/dir>	
DELETED	OFF|ON	OFF
DELIMITERS	OFF|ON	OFF
DELIMITERS	<expC>/DEFAULT	":"
DEVELOPMENT	ON|OFF	ON
DEVICE	SCREEN/PRINT /FILE <file>	SCREEN
DISPLAY	<type>	Installed
ECHO	OFF|ON	OFF
ESCAPE	ON|OFF	ON
EXACT	OFF|ON	OFF
EXCLUSIVE	ON|OFF	ON

SET Command Values		
<item>	<value>	default
FULLPATH	ON|OFF	ON
FUNCTION<num>	<char_str>	
HEADING	ON|OFF	ON
HELP	ON|OFF	ON
HELP	<filename>	FOXHELP
HOURS	12/24	12
INTENSITY	ON|OFF	ON
LOGERROR	ON|OFF	ON
MACKEY	<key>	F10
MARGIN	<0 to 254>	0
MARK	<char>	"/"
MEMOWIDTH	<8 to 32,000>	50
MOUSE	<1 to 10>	5
NEAR	OFF|ON	OFF
NOTIFY	OFF|ON	ON
ODOMETER	<1 to 32767>	100
OPTIMIZE	OFF|ON	ON
PATH	<path>	
POINT	<char>	"."
PRINT	ON|OFF	ON
RESOURCE	ON|OFF	ON
RESOURCE	<filename>	FOXUSER
SAFETY	ON|OFF	ON
SCOREBOARD	OFF|ON	OFF
SEPARATOR	<char>	","
SPACE	ON|OFF	ON
STATUS	OFF|ON	OFF
STEP	OFF|ON	OFF
STICKY	ON|OFF	ON
SYSMENU	ON|OFF	ON

SET Command Values		
<item>	<value>	default
TALK	ON | OFF	ON
TABS	<char_str>	null string
TEXTMERGE	OFF | ON	OFF
TRBETWEEN	ON | OFF	ON
TYPEAHEAD	<0 to 32000>	20
UDFPARMS	VALUE | REFERENCE	VALUE
UNIQUE	OFF | ON	OFF

Special CONFIG Items

While most items that may be set or changed through the CONFIG.FP file are options that can also be controlled with SET commands (see the previous section for full details), some CONFIG items can be specified *only* through the CONFIG.FP file.

Special Config Items

<item>	<value>	Default
COMMAND	<command>	
DOSMEM	ON|OFF|<expN>	OFF
EDITWORK	<dir>	startup directory
EMS	ON|OFF|<expN>	ON
EMS64	ON|OFF	ON
F11F12	ON|OFF	ON
INDEX	<extension>	IDX
LABEL	<extension>	LBX
MVCOUNT	<128 to 3600> <128 to 65,000> FoxPro (X)	256
OUTSHOW	ON|OFF	ON
OVERLAY	<dir> [OVERWRITE]	FoxPro directory
PROGWORK	<dir>	startup directory
REPORT	<extension>	FRX
RESOURCE	<pathname>	FOXUSER (in startup directory)
SORTWORK	<dir>	startup directory
TEDIT	[/<expN>] <editor>	
TIME	<1 to 1000000>	6000
TMPFILES	<drive:>	startup directory

COMMAND Executes a valid FoxPro command after all other configuration settings are established. For instance, if you'd like FoxPro to clear the screen every time it starts, you could use the statement:

```
COMMAND = CLEAR
```

Or, if you'd like FoxPro to start a certain program automatically every time, you could use:

```
COMMAND = DO <program>
```

If you include a DO statement like the previous one, you can bypass the named startup program by including the name of a different program when you initially start FoxPro:

```
FOXPRO <program>
```

DOSMEM This parameter applies to FoxPro (X) only. When DOSMEM is on, FoxPro (X) will access and utilize all DOS memory along with any extended memory. The default is OFF; when DOSMEM is off, FoxPro (X) uses 60K of DOS memory and the rest is available for execution of the RUN commmand. If you want to reserve part of DOS memory, you can use a numeric argument <expN>, where <expN> is the amount of memory in kilobytes to reserve.

EDITWORK Specifies where the text editor should place its work files. In some circumstances, the work file can become as large as the original file; for that reason, this option should only be specified if the alternate location has plenty of free space.

EMS Determines whether or not FoxPro will take advantage of expanded memory (EMS) and can limit the amount of EMS that FoxPro uses. This option is not recognized in the Extended version of FoxPro. Generally, FoxPro can coexist with other programs that use expanded memory. However, if you want specifically to reserve all or part of expanded memory for use by other programs, you have several options:

```
EMS = OFF
EMS = <expN>
```

When you use the optional numeric argument, you limit the amount of EMS that FoxPro takes advantage of. The value is expressed in kilobytes,

and it should range from 0 (zero) to the total amount of EMS installed in your computer. The value should be some multiple of 16, although FoxPro reduces nonincremental values to the nearest 16K boundary.

If you have LOAD and CALL routines that use expanded memory, you should be aware that FoxPro always places its own memory back into the expanded memory page frame after a CALL. If your routines are not prepared for this, you may want to set EMS OFF to prevent it. The default setting for EMS is ON.

EMS64

On machines with expanded memory that's compatible with LIM 4.0 (or higher), FoxPro automatically uses the first 64K of expanded memory as "general purpose" memory. However, certain older EMS emulators cause problems for FoxPro. If you have an older EMS emulator that causes FoxPro to behave strangely, you can specify that FoxPro not use the first 64K of expanded memory as general purpose memory with:

```
EMS64 = OFF
```

or you can ignore all expanded memory *except* for the first 64K of "general purpose" memory with:

```
EMS = 64
```

This option is ignored in FoxPro (X).

F11F12

Prevents FoxPro from attempting to use the F11 and F12 keys. You may need to use this option if your computer does not have F11 and F12 keys and no cursor appears in the Command window when you start FoxPro. This happens on computers without F11 and F12 keys that have certain older versions of the BIOS which erroneously return "yes" when FoxPro tests for the presence of these keys.

INDEX

Specifies the extension for FoxPro index files. The default extension is .IDX.

LABEL Specifies the extension for FoxPro label definition files. The default extension is .LBX.

MVCOUNT Sets the maximum number of memory variables that FoxPro can maintain. This value may range from 128 to 3,600 in FoxPro standard, or to 65,000 in FoxPro (X); the default is 256.

OUTSHOW Disables the Shift+Ctrl+Alt (hide all windows in front of the current output window) feature.

When several windows are open, the current output window may sometimes become hidden behind some of the other windows. By pressing Shift+Ctrl+Alt, you can momentarily hide all windows that are in front of the current output window. Use OUTSHOW to disable this feature.

OVERLAY Specifies where FoxPro should place its .OVL (overlay) file. This applies to standard FoxPro only; the extended version does not use overlay files. At startup, the .OVL file in the same directory as the FOXPRO.EXE file is used. Specifying OVERLAY causes FoxPro to look in the new location for the files.

If the files are not present in this directory, the originals are copied to the new location. If the files exist in the new directory but are not dated identically to the originals, FoxPro asks for permission to overwrite the files. If you give permission, FoxPro overwrites the files with the new copies; otherwise, it uses the originals.

If you include the optional OVERWRITE keyword, FoxPro automatically overwrites the existing files without prompting for permission. See the discussion of these commands following TMPFILES below.

PROGWORK Specifies where the program cache file will be placed. Users may wish to put this file in a RAM disk or on a local workstation drive. FoxPro tries to keep the size of this file less than 256K, but it can grow larger if necessary. It may be especially useful to specify PROGWORK when using

FoxPro/LAN. See the discussion of this and related commands below.

REPORT Specifies the extension for FoxPro report definition files. The default extension is FRX.

RESOURCE Specifies where FoxPro is to find the FOXUSER resource file. <pathname> may be a directory or a fully-qualified pathname. If <pathname> is a directory, then a file named FOXUSER.DBF is searched for. Otherwise, a file with the specified name is searched for. If the file does not exist it is created.

SORTWORK Specifies where commands such as SORT and INDEX will place their temporary work files. SORT and INDEX can require work space up to twice the size of the file being sorted or indexed, so be sure there is enough room in this directory. This parameter can be useful when you are operating in a FoxPro/LAN environment. SORTWORK and related commands are discussed below.

TEDIT Specifies the external text editor used when you edit program files with MODIFY COMMAND.

You can include the optional clause /<expN> with TEDIT to specify the amount of memory FoxPro should make available for an external text editor. <expN> specifies the amount of memory in kilobytes (K) that is made available.

To make as much memory available to the text editor as possible, specify a value of 0 (zero):

```
TEDIT = /0 <editor>
```

This is used in the Standard version FoxPro only.

TIME Establishes the amount of time that FoxPro waits for the print device to accept a character. If the printer is not ready, this value dictates the number of retries that FoxPro will make to the print device. If the retry count is exhausted, FoxPro issues the error "Printer not ready. Retry? (Y/N)".

The value of TIME may range from 1 to 1,000,000 retries. The default is 6,000.

TMPFILES Sets the drive to which the EDITWORK, SORTWORK and PROGWORK files will be stored if they have not been specified otherwise with any of the other options. This can be useful for optimizing performance in a FoxPro/LAN environment.

The EDITWORK, OVERLAY, PROGWORK, SORTWORK and TMPFILES special configuration items let you specify where temporary work files and frequently accessed FoxPro files will be placed. Careful placement of these files can improve FoxPro's performance, particularly on a network.

If possible these files should be placed on the fastest drive available. On networks the fastest drive is usually a workstation's local drive. Be sure to provide ample disk space for the temporary work files as some can grow quite large.

Further performance information can be found in the chapters Optimizing Your System and Optimizing Your Application in this manual.

Startup Switches

When FoxPro is started, you can include a number of command line switches that will control how FoxPro operates. A command line switch consists of a hyphen, followed by an upper- or lower-case letter, optionally followed by any additional information that may be needed (such as file or path name). No embedded spaces are allowed. These switches will override any environment variables that may configure the same FoxPro operation (e.g., the FOXPROCFG environment variable will be overridden by -C).

The command line switches can also be specified using the DOS environment variable FOXPROSWX, discussed later in this section.

The table that follows lists the startup switches and their effects.

Startup Switches	
Switch	**Effect**
-C<pathname>\<file>	Specify configuration file
-E	Prevent use of all expanded memory
-K	Prevent attempts to use F11 or F12
-T	Suppress FoxPro sign-on screen

Specify Configuration File

You can specify the name of the configuration file to be used with the -C startup switch. If you do not include the <file>, you must include a "\" after the <pathname> for FoxPro to look for an existing CONFIG.FP.

This is equivalent to setting the environment variable FOXPROCFG. However, if the environment variable exists, this switch overrides the variable's effect.

Turn Off Use of Expanded Memory

You can prevent FoxPro from using expanded memory by including the -E switch.

This is equivalent to using the EMS = OFF statement in your CONFIG.FP file.

Prevent Attempts to Use F11 and F12 Keys

If your computer does not have F11 and F12 keys and no cursor appears in the Command window when you start FoxPro, use the optional -K switch to prevent FoxPro from trying to use these keys.

This is equivalent to using the F11F12 = OFF statement in the CONFIG.FP file.

Suppress Sign-on Screen

You can prevent the display of the sign-on screen by including the optional -T switch.

This switch has no equivalent CONFIG statement.

Loaders

FoxPro includes four loader programs: FOX.EXE, FOXS.EXE, FOXL.EXE and FOXR.EXE. A loader checks the amount and type of memory available and the FoxPro versions you've installed (Single-User, Multi-User or Runtime — Standard or Extended). It then invokes the most advanced version of FoxPro available that can run on your computer.

When a loader has the option of executing different versions of FoxPro, the most advanced version is executed. Here is the order of precedence:

- Extended FoxPro/LAN
- FoxPro/LAN
- Extended FoxPro
- FoxPro
- Extended FoxPro Runtime
- FoxPro Runtime

The loaders search your DOS path for your FoxPro versions. The most advanced version is loaded, not the first version located along the DOS path.

You can always execute a specific version of FoxPro by typing the full name of the version. For example, if you want to execute the single-user Extended version of FoxPro, type the following at the DOS prompt:

```
FOXPROX
```

The loaders are provided especially for the convenience of network users. Because the loader requires an extra 5K of memory, if you know what version you want to run, type the explicit name of that version instead of using the loader.

To use a loader, type the loader's file name at the DOS prompt and press Enter. The appropriate version of FoxPro is executed. You can also include optional command line switches after the loader name to execute a specific version of FoxPro.

The following table lists the names of the four loaders, the versions of FoxPro that they invoke and the command line switches that can be included.

File Name	Loads	Switches
FOX.EXE	All versions	+L–Load LAN version - L–Don't load LAN version +R–Load runtime version - R–Don't load runtime version +X–Load extended version - X–Don't load extended version /?–Display help text
FOXS.EXE	FoxPro (Single-User)	+X–Load extended version - X–Don't load extended version /?–Display help text
FOXL.EXE	FoxPro/LAN (Multi-User)	+X–Load extended version - X–Don't load extended version /?–Display help text
FOXR.EXE	FoxPro Runtime	+X–Load extended version - X–Don't load extended version /?–Display help text

Command Line Switches

You can force or prevent the execution of certain FoxPro versions by including optional command line switches. The switches are placed after the loader name. Separate the loader name and the first switch with a space. More than one switch can be included — be sure to separate multiple switches with spaces.

To see a listing of switches available for a loader, type the loader's name at the DOS prompt followed by a space and /? and then press Enter.

To force a version to execute, include + before the switch. To prevent a version from executing, include – before the switch.

Examples

FOX.EXE can be used to execute any FoxPro version. In this DOS command line example it is used without any of the optional switches to run the most advanced version of FoxPro available.

```
FOX
```

The following example uses ? to display the switches that are available for the FOX.EXE loader.

```
FOX /?
```

In this example, FOX.EXE is used with the +X and +L switches to force the Extended version of FoxPro/LAN to run.

```
FOX +X +L
```

This example uses the Multi-User loader FOXL.EXE to run the standard version (not the Extended version) of FoxPro/LAN.

```
FOXL -X
```

Other FoxPro DOS command line switches (-C, -E, -K and -T) can be combined with the loader switches. After the switches, you can also include the name of a FoxPro program to execute after FoxPro is loaded. The following example uses the Multi-User loader FOXL.EXE to run the standard version (not the Extended version) of FoxPro/LAN, the sign-on screen is suppressed (-T option) and the program ORGANIZE is run.

```
FOXL -X -T ORGANIZE
```

Specifying Switches with FOXPROSWX

Switches can be specified in a DOS environment variable named FOXPROSWX. You can create this DOS environment variable with the DOS SET command.

If FOXPROSWX exists, the loader uses the switches it contains. Other FoxPro DOS command line switches (-C, -E, -K and -T) can be combined with the loader switches in FOXPROSWX. You can also include the name of a FoxPro program to execute after FoxPro is

loaded. Switches can be specified using both FOXPROSWX and command line arguments. If a conflict exists, the command line arguments prevail.

In the following example, FOXPROSWX is created at the DOS prompt and the switches +X -L -T (Extended Single-User version, suppress the sign-on screen) are stored in it. The loader FOX.EXE is executed. Since switches are not included after the loader, the loader uses the switches in FOXPROSWX.

```
SET FOXPROSWX= +X -L -T
FOX
```

The Extended Single-User version of FoxPro is invoked and the sign-on screen is suppressed.

Do not include spaces between FOXPROSWX and the equal sign. For additional information on creating a DOS environment variable with SET, consult your DOS manual.

Function Keys and Macros

In addition to the various configurations and hardware modifications that you can use to optimize performance, you can modify several other program options to suit your needs and tastes.

Function Keys

FoxPro comes to you with default values or actions assigned to ten of the available function keys.

Function Key Assignments	
Key	**Assignment**
F1	(Activate on-line help)
F2	set;
F3	list;
F4	dir;
F5	display structure;
F6	display status;
F7	display memory;
F8	display;
F9	append;
F10	(Activate/deactivate the system menu bar — same as Alt)

The semicolon (;) at the end of each character string represents a carriage return. With the semicolons, FoxPro outputs the string just as if you had typed it in and pressed Enter. Without the semicolons the character strings are simply typed out for you, but do not include an Enter needed for them to execute.

FoxPro allows you to assign a different character string, up to 254 characters maximum, to F1 through F12 with the SET FUNCTION

command or by including the appropriate statements in the CONFIG.FP file. (If your computer has F11 and F12 keys, they have no default in FoxPro but you can define them.) You may also use the **Macros...** option on the **System** menu popup to assign new meanings to function keys.

The current function key settings can be viewed in the Keyboard Macros dialog, reachable from the **Macros...** option on the **System** menu.

Macros

FoxPro contains a powerful, time-saving macro definition feature that allows you to assign a collection of FoxPro actions to a single keystroke. By defining and using macros you can avoid retyping a series of keystrokes over and over.

The macro definition and playback feature can be used to record and play back the keystrokes that are necessary to carry out an entire task. You can create macros that emulate the commands of your favorite word processor or text editor, store the steps that are needed to save and close a text file or assemble various collections of text that you use often.

A collection of macro definitions can be saved to a macro file (extension .FKY) and restored for later use. For more information, refer to Macros in the System Menu Chapter of the FoxPro *Interface Guide.*

FOXUSER Resource File

The FoxPro resource file FOXUSER.DBF holds a variety of resource information (e.g., window positions, color sets, Browse window configurations, label definitions, etc.). FOXUSER is a standard FoxPro database, with an associated memo file (.FPT) and index (.CDX). If the index file does not exist, it is created when the FOXUSER file is opened.

While RESOURCE is SET ON, FoxPro handles all resource management. Users at the beginning to intermediate levels need not concern themselves with the contents and workings of this file. However, more advanced users may wish to inspect and modify the data contained in this file in two ways:

1. Special window configurations or other window preferences, once established and saved in the resource file, can be protected from subsequent interactive changes by modifying the READONLY field for the specific resource (see the example in the Predefining Browse Window Configuration section).
2. Resources that are no longer needed or wanted may be deleted and the resource file may be packed.

Modifying the READONLY field and/or deleting records from the resource file are the only two actions that you should perform on the FOXUSER file. We encourage you to make a copy of the resource file before you make any modifications.

Structure of FOXUSER

The FOXUSER database has the following structure:

```
Structure for database: C:\FOXPRO\FOXUSER.DBF
Field  Field Name    Type        Width    Dec
    1  TYPE         Character      12
    2  ID           Character      12
    3  NAME         Character      24
    4  READONLY     Logical         1
    5  CKVAL        Numeric         6
    6  DATA         Memo           10
    7  UPDATED      Date            8
** Total **                       74
```

TYPE This field identifies the type of information the resource holds.

ID This field identifies the record within the TYPE.

NAME This field contains the name assigned to the resource, for instance the name assigned to color sets, BROWSE and system windows.

READONLY This logical field can be set to true (.T.) to indicate that the resource data should be used for read-only access and cannot be changed.

CKVAL This field is used by FoxPro to verify that the data (in the memo field DATA) is valid.

DATA This is a memo field that holds the actual data for the named resource.

UPDATED The date the current record was last changed.

The FOXUSER resource file can be made read-only by marking it at the DOS level. The DOS command ATTRIB uses the <+r> setting to mark a file as read only; see your DOS manual for instructions. Marking a file read-only is useful in multi-user situations, because the FOXUSER file can be shared on a network.

Modifying the FOXUSER Resource File

Several SET commands are available that allow you to open and easily modify the FOXUSER file's contents, as well as use a resource file other than FOXUSER. These commands permit you direct access to the resource database of your choice.

```
SET RESOURCE TO <resourcefile>
SET RESOURCE TO
SET RESOURCE ON
SET RESOURCE OFF
```

If you want to make changes to your resources, use the various dialogs in the interface to turn off access to the resource file and open it with its associated index as described below:

1. In the On/Off panel of the View window, uncheck the **Resource** check box. This makes the resource file available to be opened as a database.
2. In the View panel of the View window, select an unused work area and open FOXUSER.DBF.

3. Choose the **Setup** push button and open the index file FOXUSER.IDX in the Setup dialog.
4. Now you can use the Browse window to modify the READONLY field or mark records for deletion.

If you prefer to work through the Command window, or you want to create a utility program to do all of this for you, use the following commands:

```
SET RESOURCE OFF
USE FOXUSER
BROWSE
```

With the resource file open, you can use CHANGE, EDIT, BROWSE, DELETE, REPLACE and other FoxPro commands to modify or delete data. You can even use COPY TO to create a new resource database. When you've finished modifying the resource file, pack it if you marked records for deletion, and close it. Then, set the resource feature on through the interface dialogs or with the following commands:

```
USE
SET RESOURCE ON
```

You can open the FOXUSER file in a work area without setting resource OFF by issuing the command `USE SYS(2005) AGAIN`. SYS(2005) returns the name of the current resource file.

Predefining Browse Window Configuration

One handy modification that you can make to the FOXUSER file is changing the READONLY field for predefined Browse window configurations.

First, adjust a BROWSE window to display specific fields that are sized for easy manipulation, position the window and size it, split the window into two partitions with one partition in BROWSE mode and one in CHANGE, then close the window. When the BROWSE window is closed, its configuration is saved in the current resource file, provided the field named READONLY is not set to T and provided the window *was not* closed with Ctrl+Q.

Then, issue `USE SYS(2005) AGAIN` and edit the READONLY field, setting it to true ("T"). To do this, look for the BROWSE window resource that has a TYPE of PREF2.0, an ID of WINDBROW and a NAME field with the alias of the database you just browsed. When the READONLY field is true (T), the corresponding resource data will not be overwritten if the Browse window configuration is later changed interactively.

Now, whenever this database is browsed with the BROWSE LAST command, the Browse window appears in the same position with all its field, size and placement configurations the same as the last session. No matter how much the Browse window is changed by the user, it will always return according to the data in the resource when you BROWSE LAST.

Extended Display Modes

FoxPro can take advantage of special modes on EGA and VGA monitors that show many more characters on the screen.

Display Modes

The CONFIG.FP file provides you with the ability to set your display to the following display modes:

- COLOR
- EGA25
- EGA43
- CGA
- VGA25
- VGA50

For complete information on these DISPLAY options see the SET DISPLAY TO command in the FoxPro *Commands & Functions* manual.

Additional Display Modes Also Supported

If you have a video card and monitor that can operate in even greater resolution modes (for instance 132 columns by 43 rows), FoxPro can take advantage of this, too. All you need to do is switch to the mode then run FoxPro.

Running FoxPro under one of these extended display modes gives you a much larger work area. With a larger work area you can open and manipulate more windows, display more data and generally have greater room in which to work.

Color

In FoxPro, colors may be set in two ways: through the Color Picker dialog available by choosing **Color...** from the **Window** menu popup, or with the SET COLOR commands. The Color Picker dialog is the quickest way to set colors, and color sets may be saved for later use in the Color Picker dialog. Refer to the FoxPro *Interface Guide* for more information on setting colors through the Color Picker dialog.

FoxPro provides a default set of colors that are loaded and used when you start FoxPro. If you would prefer a different set of colors to be used as the default when FoxPro is started, you can create and save your own custom default color set. Your default color set can be specified in the Color Picker or in the FoxPro CONFIG.FP configuration file.

When you manipulate colors in FoxPro, it is important to understand the following color terms: color pair, color pair list, color scheme and color set.

Color Pair

A color pair consists of a foreground and background color combination. Colors are specified with a character abbreviation. An asterisk (*) is used to denote blinking or bright (depending on the state of SET BLINK) and a plus sign (+) denotes high intensity. The following is a table of the colors that are available and their codes:

Color	Code	Color	Code
Black	N	Green	G
Blank	X	Magenta	RB
Blue	B	Red	R
Brown	GR	White	W
Cyan	BG	Yellow	GR+

On monochrome monitors, only four colors are available – white (W), black (N), underlined (U) and inverse video (I).

The blank (X) color is useful for entry of passwords.

On monochrome monitors the BLINK setting (OFF or ON) has no affect upon the ability to brighten a color or to make it blink. On a monochrome monitor the background color will always be "normal" — it cannot be brightened or made to blink. Including a plus sign after a foreground or background color brightens the *foreground* color. To make the *foreground* color blink, place an asterisk after the foreground or background color.

With most color monitors you may "brighten" the colors that are available. The state of the SET BLINK command (OFF or ON) determines whether colors can be "brightened", effectively doubling the number of colors available, or made to blink.

Color Pair List

A color pair list consists of one to ten color pairs separated by commas. A color scheme is composed of ten color pairs. For example, W+/B, W+/BG, GR+/B, GR+/B, R+/B, W+/GR, GR+/RB, N+/N, GR+/B, R+/B. A monochrome color pair list might look like W/N, N+/W, W+/N, W+/N, W/N, U+/N, W+/N, –, W+/N, W/N.

Color Scheme

A color scheme is a set of ten color pairs. Color schemes control the colors of interface elements such as System Windows (color scheme 8). Each color pair in a color pair list correspond to an item in a color scheme. The color pair list for a color scheme can be returned with SCHEME( ). The color pair number and the interface element they control is available in the Color Picker dialog, which can be accessed by choosing **Color...** from the **Window** menu popup.

Color schemes have a dash (-) or plus (+) in the eleventh color pair location. This indicates whether or not a window or menu popup created with the color pair list casts a shadow.

When FoxPro is first started, a default set of color schemes is loaded. To specify you own startup color schemes in CONFIG.FP, include the line COLOR OF SCHEME <expN> = <color pair list> for each color scheme you would like to define.

Each color scheme has a number from 1 to 24. The following table lists each color scheme number and objects the scheme controls.

Color Scheme #	FoxPro Objects
1	User-Defined Windows
2	User-Defined Menus
3	System Menu Bar
4	System Menu Popups
5	System Dialogs
6	System Dialog Popups
7	System Alerts
8	System Windows
9	System Window Popups
10	Browse Windows
11	Report Layout Windows
12	Alert Dialog Popups
13 – 16	Reserved for future use
17 – 24	User Schemes

Color Scheme Descriptions and Considerations

Before you modify a color scheme, refer to the descriptions below. You should also read the following information regarding color schemes before you make any changes.

Some schemes are related and are used in the same dialogs or windows. For example, dialogs containing scrollable lists or menu popups use two color schemes — the dialogs use Scheme 5, while the scrollable lists and menu popups within dialogs use Scheme 6. Schemes 1 and 2, 3 and 4, 5 and 6, and 8 and 9 are related, and it's recommended that each pair of color schemes be coordinated.

Scheme 1 User Winds This scheme is used for the color of the clock, the underlying screen and user-defined windows. Schemes 12-24 also use the same colors as FoxPro's original colors for Scheme 1.

Scheme 2 User Menus This scheme is used for all user-defined menus and popups. The colors should coordinate with Scheme 1.

Scheme 3 Menu Bar This scheme controls the colors of the system menu bar. It only uses four of the ten color pairs.

Scheme 4 Menu Pops This scheme is for the system menu popups assigned to the menu bar. It should be coordinated with the colors of Scheme 3.

Scheme 5 Dialogs This scheme is for dialogs and system messages. Popups and scrollable lists that appear in dialogs are controlled by Scheme 6. For best visual appearance, we recommend that Color Pairs 1, 3, 9 and 10 use the same background color.

Scheme 6 Dialog Pops The scrollable lists and menu popups that appear within dialogs use this scheme. Color Pairs 1, 2 and 3 should have the same background color, which is different from the background color of Color Pairs 1, 3, 9 and 10 in Scheme 5. If you use a bright foreground color for Color Pairs 1, 2, and 3 (in Scheme 6), please see the Color Pair Exceptions section above. These colors should be coordinated with Scheme 5.

Scheme 7 Alerts These are similar to dialogs but have no text input regions.

Scheme 8 Windows Windows that use this scheme include the Command window, editing windows, Debug, Trace, Help, View and desk accessories.

We recommend that the background color of Color Pairs 3, 4 and 5 be the same. Best visual appearance is obtained when Color Pairs 1, 7, 9 and 10 have the same background color. Selected text (Color Pair 6) should have a background color that's different from editable text (Color Pair 1).

The View window and the Label Layout window use

this color scheme also. However, Color Pair 2, rather than Color Pair 1, is used for editable text. In these two windows, Color Pair 1 is used for static text.

For windows with grids using Scheme 8, the grid backround will be the background of Color Pair 1. The grid lines (foreground) will be the background of Color Pair 3.

Scheme 9 Window Pops This scheme controls the color of scrollable lists and popups that appear in windows, such as the Work Areas list and the Relations area of the View window. The colors in this scheme should be coordinated with Scheme 8.

Scheme 10 Browse The Browse window has a color scheme of its own. In the Browse window you can select a line, a field in the line and text in the field. The colors in Scheme 10 are specifically designed to identify which of the three selections is currently in effect.

For windows with grids using Scheme 10, the grid backround will be the background of Color Pair 1. The grid lines (foreground) will be the background of Color Pair 3.

The bullet that appears in the delete/recall column of the Browse window when you mark a record for deletion appears as the background color of Color Pair 7.

Scheme 11 Report The Report Layout window has many details and uses all of the color pairs. The bands of a report are identified as Band A or Band B, alternating from the top down (PgHead = A, Detail = B, PgFoot = A, etc.). The titles of the bands use four Color Pairs (1, 7, 9 and 10), with one color pair for each band type (A and B) when the band is empty and when it has objects in it.

Scheme 12 Alert Pops This scheme controls editing regions and scrollable lists.

A user defined object (GET, check box, list, push button) takes on the color scheme of the window it is drawn in. A user-defined menu takes on the colors of scheme 2.

You can override the default scheme of a user defined object by including the color scheme clause with the command. The following commands support the color scheme clause:

@ ... SAY/GET

@ ... GET – Check boxes

@ ... GET – Invisible buttons

@ ... GET – Lists

@ ... GET – Popups

@ ... GET – Push Buttons

@ ... GET – Radio Buttons

@ ... GET – Text Edit Regions

@ ... FILL

@ ... TO

BROWSE

CHANGE

DEFINE BAR

DEFINE MENU

DEFINE PAD

DEFINE POPUP

DEFINE WINDOW

EDIT

SHOW GET

SHOW GETS

SHOW OBJECT

Color Set

A color set consists of 24 color schemes. The complete color environment may be saved in a color set. Color sets, like keyboard macros and memory variables, may be saved for later use. A color set can be assigned a name of up to ten characters and saved. Color sets are stored in the FOXUSER.DBF resource file.

You may load a color set with SET COLOR SET TO or through the Color Picker dialog.

Color sets also contain the setting of SET BLINK (ON or OFF). When a color set is loaded, the BLINK setting is also restored. See SET BLINK for more information.

When FoxPro is first started, a default color set is loaded unless a different color set is specified in the FoxPro configuration file (CONFIG.FP). To specify a startup color set in the configuration file, put the line COLOR SET = <ColorSetName> where <ColorSetName> is a color set that you previously defined and saved.

Commands that manipulate color schemes include:

SET COLOR OF SCHEME

SET COLOR SET TO

SCHEME( )

These commands are described in FoxPro *Commands & Functions*.

For complete information about color, see the FoxPro *Interface Guide*.

Specifying Colors in CONFIG.FP

You can customize your FoxPro environment, including colors, with CONFIG.FP. In CONFIG.FP, you can specify a default color set that is loaded when you start FoxPro, or the colors for individual color schemes.

To specify a default color set for use when FoxPro starts, include the folowing line in your CONFIG.FP file:

COLOR SET = <mycolorset>

where <mycolorset> is the name of the color set you would like to use when starting FoxPro.

The default color set you specify in your CONFIG.FP must be present in the FOXUSER.DBF file and FoxPro must be able to locate the resource file. If FoxPro cannot find the color set you specify in FOXUSER.DBF or cannot locate FOXUSER.DBF, FoxPro's default colors are used.

To specify startup colors for an individual color scheme, you can include the following line in CONFIG.FP:

COLOR OF SCHEME <expN> = <ColorPairList>

where <expN> is the number of the color scheme (1 through 24) you would like to define, and <ColorPairList> is a set of color pairs. You can include a line in CONFIG.FP for each color scheme. The <ColorPairList> is a set of 1 to 10 color pairs and you can include a trailing plus or minus sign to specify a shadow.

If you include lines to designate a color set *and* color schemes in your CONFIG.FP, the color set takes precedence over the color schemes.

If you only designate color schemes in your CONFIG.FP and you have a default color set in your resource file, the color schemes take precedence over the default color set. Your default color set will not be loaded — the FoxPro default color set will.

Specifying Colors in FOXUSER

The FoxPro resource file FOXUSER.DBF saves information about the FoxPro environment. For example, the positions and sizes of system windows, text editing preferences, Calendar/Diary entries, etc., are kept in FOXUSER.DBF. Color sets are also stored in FOXUSER.DBF.

In FOXUSER.DBF, you can designate a default color set that is loaded when FoxPro starts. When you create or modify a color set to be your default color set, save the color set in the Color Picker with the name DEFAULT. This default color set will now be loaded when you start FoxPro. If you have specified a color set in both the CONFIG.FP file and the FOXUSER.DBF file, the default color set in the CONFIG.FP file takes precedence.

Because the FOXUSER.DBF file is a regular FoxPro database file, it may be manipulated like any other database file. To open FOXUSER.DBF you must first issue the command SET RESOURCE OFF in the Command window or in a program. You can then open FOXUSER.DBF with the USE command.

Color Table by Radio Button

Color Scheme	Usr Wind (Scheme 1)	Usr Menus (Scheme 2)	Menu Bar (Scheme 3)	Menu Pops (Scheme 4)	Dialogs (Scheme 5)	Dlog Pops (Scheme 6)
Color Pair						
Status/ Hot Key (Color Pair 7)	Clock & Hot Keys		Hot keys	Hot keys	Hot keys	
Enhanced Text (Color Pair 2)	GET field	Enabled opt.	Enabled pads	Enabled opt.	Text box	Enabled opt.
Message (Color Pair 5)	Title, idle & message	Message				
Title Heading (Color Pair 4)	Title, active	Menu titles				
Standard Text (Color Pair 1)	SAY field	Disabled opt.	Disabled pads	Disabled opt.	Normal text	Disabled opt.
Selected (Color Pair 6)	Selected item	Selected opt.	Selected pad	Selected opt.	Selected item	Selected opt.
Enabled Control (Color Pair 9)					Enabled ctrl.	
Shadow (Color Pair 8)	Shadow	Shadow		Shadow	Shadow	Shadow
Disabled Control (Color Pair 10)					Dsabl'd ctrl.	
Border (Color Pair 3)	Border	Border		Border	Border	Border [1]

1 The background color of a scrollable list is the background color of Color Pair 3. The foreground color is the foreground color of Color Pair 3 — not bright. Hence, even if you select a bright foreground for Color Pair 3, it will be coerced to the normal color in the scrollable list border.

2 For the View window and the Label Layout window only.

3 For the View and Label Layout windows, this color pair controls static text.

Alert (Scheme 7)	Windows (Scheme 8)	Wind Pops (Scheme 9)	Browse (Scheme 10)	Report (Scheme 11)	Alert Pops (Scheme 12)
Hot keys	Hot keys		Curr. record[5]	Band A, empty	
Text box	Text box [2]	Enabled opt.	Curr. field	Report field	Enabled opt.
	Title, idle		Title, idle	Title, idle	
	Title, active		Title, active	Title, active	
Normal text	Normal text [3,6]	Disabled opt.	Other records	Text & B full	Disabled opt.
Selected item	Selected text	Selected opt.	Selected text	Selected item	Selected opt.
Enabled ctrl.	Enabled ctrl. [6]			Band A, full	
Shadow	Shadow	Shadow	Shadow	Shadow	Shadow
Dsabl'd ctrl.	Dsabl'd ctrl.			Band B, empty	
Border	Border	Border [1]	Border [4]	Border	Border [1]

4 For windows with grids using Scheme 8 or Scheme 10, the grid background will be the background of Color Pair 1. The grid lines (foreground) will be the background of Color Pair 3.

5 The bullet that appears in the delete/recall column of the Browse window when you mark a record for deletion appears as the background color of Color Pair 7 in Scheme 10.

6 In the Label Layout window, the lines and arrows that show the label dimensions are the foreground color of Color Pair 9 in Scheme 8. The background area around these lines is the background color of Color Pair 1 in Scheme 8.

Color Table by Color Pair

Color Scheme	Usr Wind (Scheme 1)	Usr Menus (Scheme 2)	Menu Bar (Scheme 3)	Menu Pops (Scheme 4)	Dialogs (Scheme 5)	Dlog Pops (Scheme 6)
Color Pair						
Color Pair 1	SAY field	Disabled	Disabled pads	Disabled opt.	Normal text	Disabled opt.
Color Pair 2	GET field	Enabled opt.	Enabled pads	Enabled opt.	Text box	Enabled opt.
Color Pair 3	Border	Border	Border	Border	Border	Border [1]
Color Pair 4	Title, active	Menu titles	Title	Title	Title	Title
Color Pair 5	Title, idle, & message	Message	Message	Message	Message	Message
Color Pair 6	Selected item	Selected opt.	Selected pad	Selected opt.	Selected item	Selected opt.
Color Pair 7	Clock, Hot Keys	Hot Key	Hot keys	Hot keys	Hot keys	Hot Key
Color Pair 8	Shadow	Shadow	Shadow	Shadow	Shadow	Shadow
Color Pair 9	Enabled Control	Enabled Control	Enabled Control	Enabled Control	Enabled ctrl.	Enabled Control
Color Pair 10	Disabled Control	Disabled Control	Disabled Control	Disabled Control	Dsabl'd ctrl.	Disabled Control

1 The background color of a scrollable list is the background color of Color Pair 3. The foreground color is the foreground color of Color Pair 3 — not bright. Hence, even if you select a bright foreground for Color Pair 3, it will be coerced to the normal color in the scrollable list border.

2 For the View window and the Label Layout window only.

3 For the View and Label Layout windows, this color pair controls static text. Shaded cells correspond to empty cells in the table on pages 36-37. Color assignments made to shaded color pairs will not alter the color of objects that by default use the scheme.

Alert (Scheme 7)	Windows (Scheme 8)	Wind Pops (Scheme 9)	Browse (Scheme 10)	Report (Scheme 11)	Alert Pops (Scheme 12)
Normal text	Normal text [3,6]	Disabled opt.	Other records	Text & B full	Disabled opt.
Text box	Text box [2]	Enabled opt.	Curr. field	Report field	Enabled opt.
Border	Border	Border [1]	Border [4]	Border	Border [1]
Title	Title, active	Title	Title, active	Title, active	Title
Message	Titles, idle	Message	Title, idle	Title, idle	Message
Selected item	Selected text	Selected opt.	Selected text	Selected item	Selected opt.
Hot keys	Hot keys	Hot Key	Curr. record [5]	Band A, empty	Hot Key
Shadow	Shadow	Shadow	Shadow	Shadow	Shadow
Enabled ctrl.	Enabled ctrl. [6]	Enabled Control	Enabled Control	Band A, full	Enabled Control
Dsabl'd ctrl.	Dsabl'd ctrl.	Disabled Control	Disabled Control	Band B, empty	Disabled Control

4 For windows with grids using Scheme 8 or Scheme 10, the grid background will be the background of Color Pair 1. The grid lines (foreground) will be the background of Color Pair 3.

5 The bullet that appears in the delete/recall column of the Browse window when you mark a record for deletion appears as the background color of Color Pair 7 in Scheme 10.

6 In the Label Layout window, the lines and arrows that show the label dimensions are the foreground color of Color Pair 9 in Scheme 8. The background area around these lines is the background color of Color Pair 1 in Scheme 8.

15 Optimizing Your System

This chapter discusses various ways that you can optimize your system. Topics covered in this chapter include:

- Types of memory and their use
- CONFIG.SYS settings
- Improving startup speed
- Files and directories
- Free disk space
- RAM disks and disk caches
- Math coprocessor
- TSRs
- Loaders
- LAN considerations

Memory

FoxPro can take advantage of *enormous* amounts of computer memory. If your computer contains extra memory and is configured correctly, FoxPro's performance can be enhanced even further.

If you want FoxPro to run at absolute top speed, or want to run FoxPro along with memory-resident programs like spoolers or key-redefinition programs, you will want to increase your computer's memory beyond the conventional limit of 640K.

Types of Memory

The PC class of microcomputers can contain three types of memory: conventional, expanded and extended.

Conventional Memory

All PC-class computers contain conventional memory (up to 640K). This is the memory that programs typically load into and run in. The Standard version of FoxPro requires that you have at least 512K of conventional memory with at least 420K of it free after memory resident programs have loaded. Once the maximum 640K of conventional memory has been installed in your computer, you can also add expanded or extended memory to increase available memory space.

Expanded Memory

FoxPro runs more than twice as fast if it has access to lots of expanded memory (EMS).

On machines with expanded memory that is LIM EMS 4.0 compatible (or higher), FoxPro uses the first 64K of expanded memory as "general purpose" memory and any remaining expanded memory to speed file I/O. General purpose memory is used to manage Browse windows, user-defined windows, menus, memory variables, programs, etc.

Extra general purpose memory is particularly beneficial in networks situations because it provides FoxPro with more room to run after the network shell is loaded.

If you do not have LIM EMS 4.0 compatible memory, expanded memory will be used only to speed file I/O. To check how much EMS is currently being used by FoxPro, use the function SYS(23).

Extended Memory

Extended memory is memory that lies above the 1MB address range. It can be used directly by some operating systems (OS/2 and UNIX), but standard DOS cannot address it.

Extended memory cannot be used directly by the Standard version of FoxPro until it is made to act like EMS. How you make extended memory act like expanded memory is dependent on your system, but typically you install a memory manager — software that provides an EMS style interface to add as much extended memory as you specify. Once the extended memory is mapped to EMS memory, FoxPro will sense that it is there and make good use of it.

The Extended version of FoxPro can take advantage of extended memory or extended memory made to act like expanded memory, but it cannot mix and match the two. If only part of your extended memory is acting like expanded memory, the Extended version will only use the memory that is acting like expanded memory. For example, if you have 8MB of memory and only 1MB of it is seen as expanded memory, the Extended version of FoxPro will not be able to access the other 7MB.

Memory and Standard FoxPro

The following suggestions will help you optimize a system that runs the Standard version of FoxPro.

EMS

If you run on an 80386 or 80486 you're in luck! There are many inexpensive programs that use extended memory to emulate EMS, such as QEMM from Quarterdeck and 386MAX from Qualitas.

If you have enough memory, you should be running the Extended version of FoxPro.

If you use non-80386 processors you have several options. First, you could invest in an EMS board. All the boards we've tested work fine so you can probably choose based on price, speed and reputation. Programs are also available for 80286 (AT-class) processors that use extended memory to emulate EMS.

Expand Beyond 640K

There are several ways to access memory in the first megabyte, above 640K. Such memory is not contiguous with memory in the lower 640K and many programs can't use it. However, FoxPro can — and it's exceedingly valuable memory.

Additional non-EMS memory beyond 640K speeds FoxPro I/O performance and — often more important — significantly expands the maximum number of windows, files, indexes, strings, etc., available. For 80386 processors, there are several programs which provide access to memory beyond 640K such as 386MAX and QEMM. For non-80386 processors, there are specialized boards that provide access memory to beyond 640K such as MAXIT from McGraw-Hill Software.

FoxPro and Older EMS Emulators

Certain older, incomplete EMS emulators cause problems for FoxPro because the emulators do not fully support all of the LIM 4.0 features. Special options are provided to allow FoxPro to operate with incomplete EMS emulators. These options don't represent the optimum configuration for FoxPro.

If FoxPro behaves strangely and you have an older EMS emulator, add one of the following to your CONFIG.FP file:

EMS64=OFF All expanded memory will be used to speed file I/O.

EMS=64 All expanded memory is ignored except for the first 64K, which is used as "general purpose" memory.

Specifically, some users with older versions of QEMM reported problems running FoxPro. If you use QEMM, we recommend that you upgrade to the latest version of QEMM.

Extended FoxPro

The Extended version of FoxPro 2.0 can be run on 386 and 486 machines equipped with as little as 1.5MB of memory with little or no performance degradation. If necessary, Extended FoxPro enables demand paging to allow it to run in as little as 1.5MB.

Demand Paging

Demand paging is a more sophisticated substitute for the segment-loading technique used by the Standard version; it permits a 1.3MB program (FoxPro) to run in much less than 1.3MB. If you have roughly 2MB or less of free memory available for FoxPro after it is up and running, you may see a message indicating that demand paging is in effect. FoxPro performs better if you can supply enough memory to run without demand paging because all of FoxPro can reside in memory at once.

When exactly 2MB is available, memory is allocated between FoxPro code and user storage as follows:

FoxPro code: 1.33MB
User storage: .67MB

As less and less memory is available, user allocation is scaled back to 1/3 of available memory. Therefore, for instance, when 1.5MB is availables 500KB is available for user programs and storage, whereas FoxPro itself is operating in 1MB. If only 1MB is available, the user has .33MB available and FoxPro has .67MB. Of course, the less memory available to FoxPro, the slower it will run.

Accessing DOS Memory

DOSMEM is a CONFIG.FP parameter for the Extended version that controls how much DOS memory is accessed. DOSMEM has three settings:

ON When DOSMEM is on, FoxPro accesses and utilizes all DOS memory along with any extended memory.

<expN> If you want to reserve part of DOS memory, you can use a numeric argument. <expN> is the amount of memory in KB to reserve. If you execute the RUN command in FoxPro, reserve the amount of memory needed by specifying <expN>.

OFF By default, DOSMEM=OFF. FoxPro will not access DOS memory if DOSMEM=OFF.

General Considerations

The following suggestions will help you optimize your system when using any version of FoxPro.

CONFIG.SYS

One easy way to automatically establish FoxPro's working environment is through the startup system configuration file, CONFIG.SYS.

The system configuration file contains certain commands that are checked and executed when you start up your computer. These settings change your computer's default configuration. For example, you can increase or decrease the amount of memory that is set aside to hold data when reading or writing from disk, or you can add a device driver that allows extended memory to be used as expanded memory (EMS).

CONFIG.SYS is not a FoxPro file. It's a file that is used by your computer's operating system to establish the working environment. Because FoxPro interacts with this environment, you must be sure that certain settings are properly established. For complete information on CONFIG.SYS, refer to the documentation that accompanied your computer's operating system.

BUFFERS= and FILES= are CONFIG.SYS statements of immediate importance to FoxPro.

Buffers The BUFFERS statement contains the number of disk buffers that DOS sets aside in memory when your computer is started. A disk buffer is a block of memory (typically 512 bytes) that DOS uses to hold data when reading and writing from disk. For best performance with FoxPro, the CONFIG.SYS file should contain a BUFFERS statement with a number between 20 and 40.

Remember that buffers take up memory so do not specify too many.

Files The FILES statement sets the number of files that the operating system can open and access at one time. This number is directly related to the number of files that FoxPro will be able to open.

The FILES statement in CONFIG.SYS should *always* be *at least* 10 more than the greatest number of files that you'll have open at one time. We recommend a *minimum* of 40.

Improving Startup Speed

FoxPro looks for certain files upon startup:

- FOXDOC.EXE
- FOXGRAPH.EXE
- FOXHELP
- FOXUSER
- CONFIG.FP

For faster startup, place these files in the startup directory or near the front of your DOS path. If you don't use FoxDoc or FoxGraph, include a text file in the FoxPro directory named for each of these applications so that FoxPro will not spend time searching the DOS path for the files. If your application does not use the FOXUSER or FOXHELP file, set them off in the CONFIG.FP file.

The DOS environment variable FOXPROCFG can be used to explicitly specify the location of CONFIG.FP. For details about FOXPROCFG, refer to the Customizing FoxPro chapter in this manual.

Files and Directories

A common cause of disappointment in FoxPro performance is operating in overpopulated directories. A typical scenario might run like this:

- Your application system consists of 150 programs with 30 database and/or index files.
- The 150 source program files (.PRG, .SPR and .MPR) are accompanied by 150 compiled program files (.FXP, .SPX, .MPX) for a total of 300 files.
- If there are any .NDX index files, the databases are reindexed to create FoxPro index files (.IDX).
- In addition, when the application runs it creates numerous temporary databases and/or temporary indexes.

As the directory becomes increasingly congested with entries, the user calls our technical support line and says "FoxPro isn't running my application as fast as you say it should." In this case, the user is looking at how fast the operating system can search its directories — a function that's not under FoxPro's control.

To correct this situation, you must reduce the number of files in your directory. To reduce the number of files in your directories and speed up file search times, you can:

- Put your compiled files and .PRG, .SPR and .MPR files in two separate directories. Better yet, you can combine your source files in a project file.
- Combine multiple .PRG files into a single file.
- Increase the number of BUFFERS specified in CONFIG.SYS to help MS-DOS keep more directory information in memory.

Deleting files from a directory does not immediately speed directory searching. When a file is deleted, the entry remains as a "hole" in the directory. MS_DOS still examines all entries, including such "holes," when searching directories.

To achieve the anticipated performance increases, you may need to copy your files into a new directory or compress the directory with a special utility program that's designed for that purpose such as Norton Utilities, DOS2OOLS, VOPT, etc.

Free Disk Space

In all processing environments, from mainframe to micro, disk input/output performance degrades significantly when a disk drive becomes nearly full. If you find that your disk drive has little free space and you wish to increase FoxPro's performance, either remove unnecessary data to increase free disk space or get a larger capacity drive.

RAM Disks and Disk Caches That Use EMS

Several products create RAM disks or disk caches using expanded (EMS) memory that are not compatible with FoxPro. As far as we can determine, the problem is that certain RAM disk or cache software doesn't follow the "rules of the road" for EMS.

If you are running a program that creates a RAM disk or disk cache out of EMS, and if data on the RAM disk is damaged, or if (in the case of disk caches) other odd disk behavior is noted, you may have this problem. We have seen no situations where the symptoms are subtle — FoxPro has difficulty instantly. This means that there is little danger of long-term data corruption.

Technical Description

If EMS is ON, FoxPro does much of its I/O directly into EMS, i.e., performs direct memory access input/output (DMA I/O) directly into the EMS page frame. It appears that FoxPro is one of very few application programs using this technique (although it is a perfectly legitimate and desirable use of EMS specifically encompassed in the EMS specification).

However, RAM disk and disk cache programs (which also perform I/O directly into EMS), conflict with FoxPro's use of EMS.

In the cases we've examined, these other programs have been written under the assumption that nobody else is performing direct EMS I/O. Here are details of what goes wrong:

1. FoxPro requests that DOS perform an I/O transfer into the EMS page frame.
2. The RAM disk or cache program intercepts this call to DOS and takes appropriate action to effect the I/O data transfer. We've seen three ways these programs go wrong:
   - Some fail to allocate the EMS they're using, causing immediate difficulties when FoxPro tries to use the same memory.
   - Some fail to restore the EMS page frame to print to the same memory it did prior to the I/O request. They fool FoxPro by changing the memory referenced in the page frame during the course of an I/O operation.
   - Some fail to note that the addresses into which data will be read are in the page frame itself. They remap the page frame so it contains their I/O buffers, then copy data from their buffers to the I/O transfer address — failing to note that this address is in their own buffer. Of course, this instantly corrupts the data in their buffers.

There's really nothing Fox can do to correct these deviations from proper EMS usage (as defined by the EMS specification).

What Fox Software Will Do

Fox Software will:

- Test as many of these products as possible to verify correct operation or identify problems
- Aggressively investigate reports of difficulties
- Maintain and circulate a list of products known to function correctly and of products for which problems have been reported
- Work with the publishers of incompatible products to assist them in correcting their products

Of course, if any of these problems are found to originate with FoxPro, or if there are reasonable adjustments that can be made to FoxPro to correct problems, we will make the appropriate changes as quickly as possible.

What You Can Do If You Run Into These Problems

Here are some suggestions to resolve these difficulties:

- If possible, configure your RAM disk or cache utility to use *extended*, not expanded, memory
- Don't use your RAM disk or disk cache program simultaneously with FoxPro. Neither one has much effect on FoxPro's performance one way or the other.
- Include EMS=OFF in your CONFIG.FP file. This solution is not recommended since FoxPro can run more than twice as fast if EMS is available. Moreover, neither RAM disk nor disk cache programs have much effect on FoxPro's performance.
- Contact the publisher of your RAM disk or cache software to obtain an updated version
- Purchase one of the products which we have verified operate correctly with FoxPro

Products Known to Work Correctly

The following list is provided solely for the convenience of Fox Software's customers and is subject to change at any time as problems are corrected and/or new revisions of these products are released.

> Some reported problems may result from incorrect operation, installation, or other misuse of a particular product; some problems may exist only in obsolete versions of a product.

Current versions of these RAM disks are known to work correctly *with DOS versions through 4.xx*:

- Standard VDISK
- Compaq VDISK
- 386DISK (using *extended* memory only)
- PC-Kwik RAM disk

Current versions of these disk caches are known to work correctly:

- Compaq disk cache on Compaq machines
  (Problems have been observed on non-Compaq machines.)
- ACER-provided disk cache
- PC-Kwik disk accelerator
- PC-Tools PC-Cache

Doubtless, there are many other utilities that work.

When to Use RAM Disks

If you use a RAM disk, it is suggested that you store the following files to it to increase FoxPro's performance:

- .OVL file
- Edit work

We recommend that you don't keep your program work or sort work files created by FoxPro on a RAM disk because they will grow in size.

Math Coprocessor

Unless you are doing intensive mathematical calculations (SIN( ), COS( ), TAN( ), etc.), a math coprocessor will not significantly improve performance. Additional memory might be a better investment.

TSRs and Memory

As their name implies, "terminate and stay resident" programs execute and remain in memory. If you need to use TSRs, load them high so FoxPro can take advantage of as much "general purpose" memory as possible.

Loaders

Loaders can take up 5-6K of memory — memory that FoxPro can use. If you know which version of FoxPro you want to run, specifically name it instead of using a loader.

FoxPro/LAN and Temporary Files

FoxPro/LAN creates its temporary files in the current working directory unless you specifically designate an alternate location by including one or more of the EDITWORK, SORTWORK, PROGWORK and/or TMPFILES statements in the CONFIG.FP file.

These temporary files are created in the course of editing, indexing, sorting, etc. Text editor sessions can also temporarily create a complete copy of the file being edited (if .BAKs are being created).

If local work stations have sizable hard drives with plenty of free space, you can improve performance in the multi-user environment by placing FoxPro/LAN's temporary work files on the local drive or on a RAM drive. Redirecting these files to a local drive or a RAM drive provides additional speed by reducing the need to access the network drive and, therefore, reducing the amount of network traffic.

16 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
- FoxBASE+ performance considerations

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 2.0 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 2.0 (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 certain commands 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 *Commands & Functions* manual.

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.

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 that 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.
- INSERT is much faster than using APPEND BLANK then REPLACE, particularly with an indexed database.
- 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

17 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:

- Overall changes in FoxPro 2.0
- Language additions and enhancements
- Approaches to achieving FoxBASE+ compatibility
- Importing files from earlier versions of FoxPro

Additions and Enhancements to FoxPro 2.0

FoxPro's Extended Version — FoxPro 2.0 (X)

FoxPro 2.0 (X) uses all available extended memory. It is a true 32-bit product. All features in the Standard Version of FoxPro are supported, and in addition:

- The number of indexes is limited only by memory. This is true in the Standard Version as well, but you can have many more in FoxPro 2.0 (X).
- The number of windows, the number of Browse sessions, and memory variable string length is limited only by memory. (Maximum string length is actually two gigabytes.)

All in all, the result is maximum performance.

Less Memory

FoxPro 2.0's root is about 280K, which makes more user memory available.

Segment Loader

FoxPro's segment loader virtually eliminates overlay contention and speeds up application execution, especially in a network environment. The segment loader also provides the general linkage mechanism used by the library mechanism.

The Rushmore Technology

The Rushmore Technology is a data access technique that permits sets of records to be retrieved very efficiently, at speeds comparable to single-record indexed access. Using Rushmore, some complex database operations run hundreds or even thousands of times faster than before. FoxPro 2.0 enables personal computers to handle truly gigantic databases, containing millions of records, at speeds comparable to mainframe database systems.

The Extended Version is required for Rushmore to function properly where large databases are used. A good rule of thumb is to use the Extended Version if your databases have, in aggregate, more than 500,000 records.

SQL

FoxPro 2.0 includes the following SQL commands: SELECT, CREATE TABLE and INSERT.

SELECT is a major extension to FoxPro's relational capability. The SELECT command can perform complex single- or multi-database queries expressed in the non-procedural SQL language. SELECT queries are automatically optimized for outstanding performance using details about key distribution, available indexes, table sizes, memory size and utilization, disk space available, etc. Many of these factors are known only to FoxPro itself; some actually vary from moment to moment. SELECT will even create ad-hoc indexes if required for optimal performance. Also, when using SQL commands you are relieved of the necessity of opening databases and other housekeeping activities.

CREATE TABLE creates a database. Each new database field is defined with a name, type, precision and scale. These definitions can be obtained from the command itself or from an array.

INSERT appends a record to the end of an existing database. The new record includes data listed in the INSERT command or included in the specified array.

RQBE

The RQBE (Relational Query By Example) facility permits creating of complex single- or multi-table queries (i.e., SQL SELECT commands) by use of the simple, intuitive RQBE window. This permits beginners to access databases without programming, experienced users to create complex queries to incorporate in programs, and end-users to effortlessly create ad-hoc queries or reports.

RQBE is also a marvelous tutorial device for users unfamiliar with SQL — the generated SELECT command can be viewed at any time to observe the effect of changes made interactively through the interface.

Screen Builder and Menu Builder

These power tools give you a new way to design menus and screens. Not only do you define the objects to be included on screens and menus, but you include code to be executed with a screen or a menu. Screens and menus are now actually pieces of your application and can be combined with programs, reports, labels and other types of files in a project.

Project Manager

Projects keep track of all programs, screens, menus, reports, labels, libraries, queries, formats and other files required to create an application. The project file can be used to create an application (.APP) or an executable program (.EXE).

API (Application Program Interface)

FoxPro 2.0's External Routine API is an interface definition that allows programmers to seamlessly extend the capabilities of the FoxPro language and user interface by creating routines in C and assembly language. The External Routine API is available by purchasing FoxPro's Library Construction Kit.

New Index File Type and New Index Options

FoxPro indexes now include:

- Standard (.IDX) indexes as utilized in FoxBASE+ and FoxPro versions 1.xx.
- Compact (.IDX) indexes. 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. This allows them to be processed much faster solely because they are physically smaller.
- Compound (.CDX) indexes. A compound index file contains multiple index entries called tags. In addition, a special compound index file, called a *structural* compound index file, is automatically opened when the associated database file is opened.

Language Enhancements

FoxPro 2.0 has over 100 new and enhanced commands and functions, surveyed later in this chapter. For details on these see the Languages Changes section that follows.

Note that maximum command line length has been increased to 2K (2048 bytes).

Interface Enhancements

Enhancements too numerous to mention have been added to the FoxPro interface — new features, new dialogs, new capabilities.

Examples include:

- FoxPro 2.0 offers context-sensitive help and help topics for the interface. In addition, **See Also** and **Lookup** buttons are available in the Details panel of the Help window.
- A keyboard macros editor allows you to manually create and edit keyboard macros.
- Text editor enhancements include the automatic continuation of hanging indents.
- Windows can now be minimized and docked.

Report Writer

- Report variables. Memory variables can now be created within a report with the **Variables...** option on the **Report** menu popup.
- The ability to open multiple reports and cut and paste between them.
- Object grouping allows you to take a set of objects and create a single object, or take a grouped set and ungroup it.
- A selection marquee, which allows multiple objects to be selected in the Report Layout window.
- Additional field calculations: deviation and variance.
- Repeating of a group header on additional pages.
- Restarting page numbering at group break.
- Specification of the minimum number of lines to follow a group.
- Storage of a report definition in a database file.

Label Designer

- Label definition stored in a database file.
- Specify style of text in labels.

View Window

- Can set one-to-many relationships between databases.

Language Changes FoxBASE+, FoxPro 1.xx

This section lists commands, functions, and system memory variables that are new or enhanced since FoxPro 1.XX, or new to FoxBASE+ users. Each is described concisely, then marked with a circle and a square.

A circle represents FoxBASE+ and a square represents FoxPro 1.XX. If the circle or square is filled in, then the command will be new to the corresponding group of users. If the graphic is empty, the command is already available, from the perspective of that user group.

For a detailed explanation of any command, function, or system memory variable, refer to the FoxPro *Commands & Functions* manual.

Graphic	*Means This Feature Is ...*
● ■	New to users of FoxBASE+ and FoxPro 1.XX
● ❑	New to FoxBASE+ users, not new to 1.XX users

New Operator

% ● ■

Modulus operator. Returns the remainder of a division operation.

New Commands

= <exp1>[,<exp2>, ...] ● ❑

Evaluates list of expressions. This is useful when you want to take advantage of a function's side-effects, particularly a user-defined function, but the actual value returned by the function is not of interest.

?/?? ● ❑

? and ?? commands have been enhanced with AT, FUNCTION, PICTURE clauses, and a new STYLE clause.

**\ | ** ● ■

For data formatting; outputs lines of text.

@ ... EDIT ● ■

Creates a text editing region.

@ ... GET Enhancements ● ■

Check boxes, invisible buttons, lists, popups, push buttons, radio buttons.

@ ... SAY/GET ● ■

For FoxBASE+ users: This command has already been enhanced with clauses for error and help message handling, custom prompts.

New in version 2.0: The SIZE clause, and K PICTURE clause.

@ ... FILL TO ... ● ❑

Changes colors on an area of the screen.

ACTIVATE MENU ● ■

For FoxBASE+ users: Command. Displays and activates a menu bar.

New in version 2.0: NOWAIT option.

ACTIVATE POPUP <name> ● ❑

Displays and activates a menu popup.

ACTIVATE POPUP ● ■

AT, BAR and NOWAIT clauses. ACTIVATE POPUP will not "rewind" popups defined from field if the record pointer is not at the top of the file.

ACTIVATE SCREEN ● ❑

Directs output to the FoxPro background screen.

ACTIVATE WINDOW ... NOSHOW ● ❑

Displays and activates a window. If present, the NOSHOW option designates a window as the output window without making it visible.

APPEND FROM ARRAY <name> [FOR <expL>] ● ❑

Adds records from an array. APPEND FROM allows a field list as an optional argument, and it can handle text files that are tab delimited.

APPEND FROM ● ■

For FoxBASE+ users: Allows (from 1.02) a field list as an optional argument, and can handle text files that are tab delimited.

New in 2.0: Now allows additional file types, and different date formats are available in delimited fields.

APPEND MEMO <name> FROM <file> [OVERWRITE] ● ❑

Adds data to a memo field from a file.

AVERAGE ● ■

If the destination array does not exist, it is automatically created.

BROWSE ● ■

For FoxBASE+ users: The BROWSE command already contains *many* enhancements. Among these are the ability to have one Browse window open per work area, complete ability to rearrange fields, change their widths, split the Browse window into two partitions (either linked or independently scrollable), and select whether to operate in EDIT or BROWSE mode in the two partitions independently.

New in version 2.0: FOR clause, NOLINK, NOLGRID, NOOPTIMIZE, NOREFRESH, NORGRID, LEDIT, LPARTITION, REDIT and REFRESH keywords, WHEN and VALID clauses, support for SET SKIP, ROW( ), and COL( ).

BUILD APP ● ■

Allows you to create an application (.FXP) from a project.

BUILD PROJECT ● ■

Allows you to create a project database by opening and processing one or more files you specify.

CALCULATE ● ❑

Performs financial and statistical calculations. If the destination array does not exist, it is automatically created.

CHANGE ● ■

For FoxBASE+ users: Activates the Browse window in EDIT mode. See BROWSE above.

New in version 2.0: FOR clause, NOLINK, NOOPTIMIZE, LEDIT, LPARTITION, and REDIT keywords, WHEN and VALID clauses, support for SET SKIP, ROW( ), and COL( ).

CLEAR MACROS ● ❑

Releases all keyboard macros from memory, including any Function key assignments.

CLEAR MENUS ● ❑

Releases all user-defined menus from memory and erases them from the screen.

CLEAR POPUPS ● ❑

Releases all user-defined popups from memory and erases them from the screen.

CLEAR READ ● ■

Terminates READ on the current level and returns control to the previous READ level (if any).

CLEAR WINDOWS ● ❑

Releases all user-defined windows from memory and erases them from the screen.

CLOSE MEMO ● ❑

Closes memo editing windows for a list of memo fields, or closes all memo editing windows.

COMPILE ● ❑

For FoxBASE+ users: Compiles a program file.

COPY INDEXES ● ■

Copies single entry index files to a compound index file.

COPY MEMO ● ❑

Copies the contents of a memo field to a file.

COPY STRUCTURE ● ■

New CDX and PRODUCTION keywords, for copying an identical structural index to a new database.

COPY TAG ● ■

Creates a single entry index file from a tag in a compound index file.

COPY TO ● ■

For FoxBASE+ users: The clause DELIMITED WITH [TAB] to output data fields that are tab delimited.

New in version 2.0: Additional file types.

COPY TO ... TYPE FOXPLUS ● ❑

Copies the selected database to another database of type FOXPLUS.

COPY TO ARRAY ● ❑

Moves data from a database record to an array. This command is similar to SCATTER.

CREATE MENU ● ■

Opens a Menu Design window.

CREATE PROJECT ● ■

Opens a Project window.

CREATE QUERY ● ■

Opens an RQBE window so you can create a SQL SELECT command.

CREATE REPORT ● ■

Programmatic Quick Report capability with the FROM <file2> [FORM | COLUMN] clause. FORM, COLUMN, FIELDS, ALIAS, NOOVERWRITE and WIDTH options.

CREATE SCREEN ● ■

Activates the Screen Builder so you can create screens.

CREATE VIEW [<file>] ● ❑

Saves current environment settings to a view definition file with a .VUE extension. This file now includes the settings of all SET options.

CREATE TABLE – SQL ● ■

Creates a database having the specified fields.

DEACTIVATE MENU ● ❑

Deactivates a menu bar and removes it from the screen.

DEACTIVATE POPUP ● ❑

Deactivates a menu popup and erases it from the screen.

DEACTIVATE WINDOW <list>|ALL ● ❑

Deactivates specific windows and erases them from the screen.

DECLARE ● ❑

Creates a one- or two-dimensional array.

DEFINE BAR ● ■

For FoxBASE+ users: Defines an option on a menu popup.

New in version 2.0: BEFORE, AFTER, KEY and MARK clauses.

DEFINE BOX ● ❑

Draws a box around printed text.

DEFINE MENU ● ■

For FoxBASE+ users: Defines a menu name.

New in version 2.0: BAR, AT LINE, IN WINDOW, KEY, MARK and NOMARGIN clauses.

DEFINE PAD ● ■

For FoxBASE+ users: Defines a menu pad on a menu bar.

New in version 2.0: BEFORE, AFTER, KEY, and MARK clauses.

DEFINE POPUP ● ■

For FoxBASE+ users: Command creates a menu popup.

New in version 2.0: IN WINDOW, FOOTER, KEY, MARGIN, MARK, MOVER, MULTI, RELATIVE, SCROLL and TITLE clauses.

DEFINE WINDOW ● ■

For FoxBASE+ users: Creates a window with the specified characteristics. This command contains many additional clauses that allow a window to be moved, sized, closed, zoomed, etc. A border option, SYSTEM, displays the specified controls on the window border (close box, size control, zoom control).

New in version 2.0: FOOTER, MINIMIZE and FILL clauses.

DELETE TAG ● ■

Removes a tag or tags from open .CDX compound index files.

DIMENSION/DECLARE ● ■

Arrays may now be redimensioned without data loss.

DISPLAY STATUS ● ■

Displays information about the current FoxPro environment in more detail than in FoxPro 1.02.

DO ● ■

IN <file> clause.

EDIT

● ■

FOR clause, NOLINK, NOOPTIMIZE, LEDIT, LPARTITION, and REDIT keywords, PARTITION, WHEN and VALID clauses, support for SET SKIP, ROW( ) and COL( ).

EXPORT

● ■

Copies records from the a database file to a new file, allowing you to use data from a FoxPro database in other software packages.

EXTERNAL

● ■

Used by the Project Manager to include files and resolve undefined references in a project.

FILER

● ❑

Activates the Filer desk accessory so you can perform a variety of operations on files and directories.

FOR ... ENDFOR

● ❑

This new control structure is used to implement a loop equipped with a loop-counter variable. The variable is automatically incremented or decremented each time through the loop.

FOXSWAP

● ❑

Makes memory available to run large external programs or DOS commands. Not used by FoxPro 2.0 (X).

GATHER ... [MEMVAR]

● ❑

Transfers data from an array or from individual memory variables to database records.

GETEXPR ...

● ❑

Invokes the FoxPro Expression Builder dialog with which you can interactively build an expression string.

HIDE MENU

● ❑

Removes a specified menu bar, several menu bars or all menu bars from the screen or a window, but not from memory.

HIDE POPUP ● ❑

Removes a specified menu popup, several menu popups or all menu popups from the screen or a window, but not from memory.

HIDE WINDOW ● ❑

Hides without closing a specified window or windows.

IMPORT ● ■

Creates a new FoxPro database from data in file formats FoxPro can't read.

INDEX ● ■

ADDITIVE, TAG, COMPACT, ASCENDING and DESCENDING clauses. Index expression may now contain a user-defined function.

KEYBOARD ● ■

Allows the keyboard buffer to be stuffed with an arbitrary character string, which remains in the buffer until FoxPro looks for input. The KEYBOARD character expression can now contain key labels or a UDF.

LABEL ● ■

PREVIEW and NOCONSOLE options.

LABEL FORM ... OFF ● ❑

Inclusion of the OFF clause causes console output to be suppressed.

LIST STATUS ● ■

Equivalent in most respects to DISPLAY STATUS.

MODIFY COMMAND/FILE ... SAVE ● ❑

New SAVE option keeps the program or text window open on the screen when a different window is selected.

MODIFY LABEL ● ■

For FoxBASE+ users: New SAVE option keeps the label window open on the screen when a different window is selected.

New in version 2.0: NOENVIRONMENT clause, NOWAIT option.

MODIFY MENU ● ■

Opens a Menu Design window.

MODIFY PROJECT ● ■

Opens a Project window.

MODIFY QUERY ● ■

Opens an RQBE window so you can modify a query or create a new query.

MODIFY REPORT ● ■

New to FoxBASE+ users: SAVE option keeps the report window open on the screen when a different window is selected.

New in version 2.0: NOENVIRONMENT clause, NOWAIT option.

MODIFY SCREEN ● ■

Opens a Screen Layout window; used for modifying existing screen definition files.

MOVE POPUP ● ■

Moves a popup to a new location.

MOVE WINDOW ● ❑

Moves a window to a new screen location.

ON BAR ● ■

Activates a popup or menu bar when a popup option is chosen.

ON KEY LABEL ● ❑

ON KEY LABEL now allows multiple ON KEY statements. LBRACE and RBRACE key labels have been added.

ON PAD ● ■

For FoxBASE+ users: Assigns a menu popup to a menu pad on a menu bar.

New in version 2.0: The ACTIVATE MENU clause

ON PAGE ● ❑

Executes a routine at a specified line number in reports.

ON READERROR ● ❑

Executes a routine upon an input error.

ON SELECTION BAR ● ■

Assigns a routine to a popup option.

ON SELECTION MENU ● ■

Assigns a routine to a menu bar.

ON SELECTION PAD ● ❑

Assigns a routine to a menu pad on a menu bar.

ON SELECTION POPUP ● ❑

Assigns a routine to a menu popup.

PACK ● ■

MEMO and DBF options; interruptable by pressing Escape.

PARAMETERS ● ■

Entire arrays may be passed.

PLAY MACRO ● ❑

Executes a series of keystrokes.

POP KEY ● ❑

POP KEY restores ON KEY LABEL commands previously placed on the stack with PUSH KEY.

POP MENU ● ■

Pulls a menu bar off the stack.

POP POPUP ● ■

Pulls a popup placed on a stack of menus.

PRINTJOB ... ENDPRINTJOB ● ❑

Activates print job settings.

PUBLIC ● ❑

The ARRAY keyword is optional when defining a public array.

PUSH KEY ● ■

PUSH KEY places all current ON KEY LABEL commands on a stack in memory. PUSH KEY, when used with POP KEY, lets you save ON KEY LABEL commands, change these commands and then restore the previous commands.

PUSH MENU ● ■

Places a menu bar on the stack.

PUSH POPUP ● ■

Places a popup on a stack of popups.

READ ● ■

For FoxBASE+ users: New NOMOUSE option restricts users to moving between fields with the keyboard, but allows them to use the mouse within fields to edit. New TIMEOUT option specifies how long the read will be in effect.

New in version 2.0: The ACTIVATE, CYCLE, DEACTIVATE, MODAL, OBJECT, SHOW, VALID, WHEN, WITH and COLOR clauses.

REGIONAL ● ■

Creates regional memory variables and memory variable arrays.

REINDEX ● ■

COMPACT option converts regular .IDX files to compact .IDX files.

RELEASE MENUS | POPUPS | WINDOWS ● ■

For FoxBASE+ users: Releases memory used by memory variables, menus, popups and windows.

New in version 2.0: EXTENDED releases all the subordinate pads, popus, bars and ON routines associated with the object.

RELEASE BAR | PAD

RELEASE BAR removes options from popups. RELEASE PAD removes pads from menu bars.

REPORT

New to FoxBASE+ users: The OFF option in REPORT FORM suppresses console output.

New in version 2.0: PREVIEW and NOCONSOLE options.

RESTORE MACROS

Restores keyboard macros from a file. This command can be used to clear all macros from memory and restore the default macros.

RESTORE WINDOW

Restores window definitions to memory from disk file.

RETURN

Omitting RETURN in a procedure or UDF automatically returns a logical true (.T.).

SAVE MACROS

Saves keyboard macros to a disk file.

SAVE WINDOW

Saves windows to a disk file.

SCAN ... ENDSCAN

Moves through a database and conditionally executes commands.

SCATTER ... [MEMVAR [BLANK]]

SCATTER (implemented in FoxBASE+ 2.10) transfers data from a database record to an array. The new MEMVAR keyword specifies that SCATTER is to create individual memory variables named identically to the corresponding database fields. If the BLANK keyword is specified, the generated individual variables are blank.

SCROLL

Scrolls horizontally in a screen or window.

SELECT – SQL ● ■

The SQL command SELECT poses a query to one or more databases.

SET ANSI ● ■

Specifies how SQL string comparisons are made with the = operator.

SET AUTOSAVE ON | OFF ● ❑

This option causes data contained in FoxPro's buffers to be flushed to disk more frequently than normal. Fox products have always incorporated a sophisticated algorithm to insure that data on disk matches data in memory. This minimizes vulnerability to power interruptions and other unplanned shutdowns.

SET BELL TO ● ❑

Modifies frequency and duration of the bell.

SET BLINK ● ❑

Allows users of EGA and VGA monitors to make screen elements blink or appear bright.

SET BLOCKSIZE ● ❑

Specifies the disk space that FoxPro allocates for storage of memo fields.

SET BORDER TO ● ❑

Defines the borders of menus, windows, popups, lines and boxes.

SET CARRY TO ● ❑

Specifies which fields will be carried forward.

SET CLOCK ON | OFF ● ❑

Displays or hides the system clock.

SET CLOCK TO ● ❑

Displays clock on the screen at specified location.

SET COLOR OF BOX | FIELDS | HIGHLIGHT |INFORMATION | MESSAGES | NORMAL | TITLES ● ❑

Defines various screen attributes for color and monochrome monitors.

SET COLOR OF SCHEME TO ● ❑

Defines or modifies a system color scheme.

SET COLOR SET TO ● ❑

Specifies a color set to be used for screen displays.

SET COMPATIBLE ON | OFF ● ❑

Setting COMPATIBLE ON resolves the few areas where prior FoxBASE+ usage conflicts with dBASE IV usage, so your dBASE IV programs will run unchanged under FoxPro. Setting COMPATIBLE OFF insures that your existing FoxBASE+ programs will run unchanged. By default, FoxPro operates with COMPATIBLE OFF. This command also supports the keywords FOXPLUS and DB4, which are synonyms for OFF and ON respectively.

SET CURRENCY LEFT | RIGHT ● ❑

Positions the currency symbol.

SET CURRENCY TO ● ❑

Specifies the currency symbol.

SET DEBUG ● ❑

Allows access to the Debug and Trace windows.

SET DEVELOPMENT ON|OFF ● ❑

Compares creation date/time of source to compiled object file.

SET DISPLAY TO MONO | COLOR | EGA25 | EGA43 | MONO43 | CGA | VGA25 | VGA43 | VGA50 ● ❑

Changes a monitor's display modes.

SET FILTER TO FILE ● ❑

Activates a filter condition contained in a file.

SET FULLPATH ON | OFF ● ❑

Causes FoxPro functions to return a full pathname (i.e., D:\PATH\FILENAME).

SET HELP TO ● ❑

Designates the file containing on-line help text.

SET HELP ON | OFF ● ❑

Enables/disables the FoxPro on-line help feature.

SET HELPFILTER ● ■

Evaluates a filter expression to display a subset of help topics in the Help window.

SET HOURS TO 12 | 24 ● ❑

Sets clock to 12- or 24-hour time.

SET INDEX ● ■

ADDITIVE, ORDER, ASCENDING and DESCENDING clauses.

SET LIBRARY ● ■

Specifies an external API routine library.

SET LOGERRORS ● ❑

Allows you to save compilation error messages to a log file.

SET MARK OF ● ■

Specifies a menu pad or popup option mark character.

SET MARK TO ● ❑

Set the date separator character.

SET MESSAGE TO ... ● ❑

Designates the row for the message line and specifies its alignment. New WINDOW <window name> clause, and new keywords LEFT, CENTER and RIGHT are supported.

SET MOUSE ● ❑

Specifies the sensitivity of the mouse.

SET NEAR ON | OFF ● ❑

Locates the nearest value in an index, when the original value is not found.

SET OPTIMIZE ● ■

Enables or disables Rushmore globally.

SET ORDER ● ■

TAG, IN <work area | alias>, ASCENDING and DESCENDING clauses.

SET POINT TO ● ❑

Specifies the character to be used as a decimal point.

SET PRECISION TO ● ❑

Specifies the precision to use in fixed point arithmetic.

SET RELATION OFF INTO ... ● ❑

Removes the relation between the selected work area and a specified work area.

SET RESOURCE ON | OFF ● ❑

Enables or disables access to the user's resource file.

SET RESOURCE TO ... ● ❑

Establishes which file FoxPro should search for needed user-defined resources.

SET SEPARATOR TO ... ● ❑

Specifies the character to be used as a numeric separator.

SET SKIP

● ■

Lets you establish a one-to-many relationship between one record in a parent database and multiple records in a child database.

SET SPACE ON | OFF

● ❑

Allows the ? and ?? commands to print a space between expressions.

SET STICKY

● ❑

Specifies whether system menu popups displayed with the mouse will remain on the screen when the mouse button is released.

SET SYSMENU

● ■

For FoxBASE+ users: Allows access to the FoxPro system menu bar during program execution.

New in version 2.0: AUTOMATIC and DEFAULT clauses, FoxPro system menu control.

SET TALK

● ■

WINDOW sends system status information to a window.

SET TEXTMERGE

● ■

Enables or disables evaluation of database fields by TEXT ... ENDTEXT commands.

SET TEXTMERGE DELIMITERS

● ■

Specifies the text merge delimiters to be used when TEXTMERGE is SET ON.

SET TOPIC TO ...

● ❑

Sets the inital help topic that will be displayed.

SET VIEW ON | OFF

● ❑

Opens/closes the View window.

SET WINDOW OF MEMO TO ...

● ❑

Lets the named window be used to edit a memo field. This only applies in the APPEND, BROWSE, CHANGE, EDIT or READ modes.

SHOW GET ● ■

Redisplays a single GET object: a field, push, radio or invisible button, check box, popup, list or text editing region.

SHOW GETS ● ■

Redisplays all GET objects.

SHOW MENU ● ❑

Displays but does not activate a menu bar.

SHOW OBJECT ● ■

Redisplays a single GET object; similar to SHOW GET except that SHOW OBJECT references objects by object number.

SHOW POPUP ● ❑

Displays but does not activate a menu popup.

SHOW WINDOW ● ■

For FoxBASE+ users: Controls the display and front-to-back placement of windows on the screen.

New in version 2.0: REFRESH keyword.

SIZE POPUP ● ■

Lets you change the size of a user-defined popup.

SUM ... TO ARRAY ● ❑

Places the results of the SUM operation into an array. If the destination array does not exist, it is automatically created.

TEXT ... ENDTEXT ● ■

Used to send lines of text to the current output device. For merging text with contents of database fields, memory variables, or evaluated expressions.

TOTAL ON <expr> ... ● ❑

In the ON clause, the TOTAL command now takes an expression instead of only a field name.

USE <expC>

For FoxBASE+ users: The USE command accepts a character-valued expression as its argument. It also accepts the INTO <work area> clause which opens the database in the named work area.

New in version 2.0: AGAIN, ORDER, ASCENDING and DESCENDING clauses.

WAIT WINDOW

For FoxBASE+ users: WINDOW option displays the WAIT message in a window in the upper right corner of the screen.
New in version 2.0: NOWAIT option for the WINDOW clause, WAIT CLEAR.

ZOOM WINDOW

Changes the size of a system or user-defined window.

New Functions

UDFs

May be used in FOR and WHILE clauses and INDEX expressions; no longer require a RETURN statement.

ACOPY( )

Copies a series of elements from one array to another.

ACOS( )

Returns the angle size from the cosine (in radians).

ADEL( )

Deletes an element, row or column from an array without changing the size of the array.

ADIR( )

Places information on matching files into an array.

AELEMENT( )

Returns an array element's number from its row and column.

AFIELDS( ) ● ■

Places information about database structure 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.

ALLTRIM( ) ● ❑

String function returns <expC> minus leading and trailing spaces.

ASCAN( ) ● ■

Searches elements of a memory variable array for a matching expression.

ASIN( ) ● ❑

Returns the angle size from the sine (in radians).

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.

AT( ) ● ❑

Can now return the starting position of the n-th occurrence of a character string.

ATAN( ) ● ❑

Returns the angle size from the tangent (in radians).

ATC( ) ● ❑

A variant of AT for which the search is case-insensitive.

ATCLINE( ) ● ❑

A variant of ATLINE for which the search is case-insensitive.

ATLINE( ) ● ❑

Returns the number of the line where a string first occurs.

ATN2( ) ● ❑

Returns the angle size from the sine and cosine (in radians).

BAR( ) ● ❑

Returns the number of the last prompt bar selected from the popup menus.

BETWEEN( ) ● ❑

Returns .T. if the value of an expression falls between the values of two other expressions.

CAPSLOCK( ) ● ❑

Tests or sets the state of the keyboard CapsLock mode setting.

CDX( ) ● ■

Returns the names of open .CDX compound index files. Identical to the MDX( ) function.

CEILING( ) ● ❑

Returns the smallest integer greater than or equal to the value.

CHRTRAN( ) ● ❑

Translates the characters of a string using a translate table.

CHRSAW( ) ● ❑

Returns .T. if a character is present in the keyboard buffer but *does not* affect the buffer contents.

CNTBAR( ) ● ■

Returns the number of bars (options) in a popup.

CNTPAD( ) ● ■

Returns the number of pads (items) in a menu.

COS( ) ● ❑

Returns the cosine value from the angle (in radians).

CURDIR( ) ● ❑

Returns the current DOS directory either on the default drive or on an indicated disk drive.

DIFFERENCE( ) ● ❑

Returns the difference between literal strings.

DMY( ) ● ❑

Executes a date format conversion to DD, Month, YY.

DTOR( ) ● ❑

Converts degrees to radians.

DTOS( ) ● ❑

Returns its date argument formatted in a manner suitable for use in indexing. This format (YYYYMMDD) is identical to that returned by DTOC when the optional second argument is specified.

EMPTY( ) ● ❑

Returns .T. if the argument expression is "blank". Works for all types of data.

EVALUATE( ) ● ■

Evaluates a character expression and returns the result.

FCLOSE( ) ● ❑

Closes a file specified by a file handle.

FCREATE( ) ● ❑

Creates or truncates a file and returns its file handle.

FEOF( ) ● ❑

Returns .T. if a file, specified by its file handle, is at end-of-file.

FERROR( ) ● ❑

Indicates if an error occurred in a low-level file I/O function and what type of error it was.

FFLUSH( ) ● ❑

Flushes a file specified by a file handle.

FGETS( ) ● ❑

Reads data until either a specified number of characters have been read, or until a carriage return is encountered.

FILTER( ) ● ❑

Returns the filter expression for a specified work area.

FLOOR( ) ● ❑

Returns the largest integer less than or equal to the value.

FOPEN( ) ● ❑

Opens a file and returns its file handle.

FPUTS( ) ● ❑

Outputs a line of text followed by a carriage return and line feed.

FREAD( ) ● ❑

Reads a specified number of bytes from a file.

FSEEK( ) ● ❑

Moves a file pointer and returns its new value.

FSIZE( ) ● ❑

Returns the size in bytes of a specified database field as shown in MODIFY STRUCTURE.

FULLPATH( ) ● ■

For FoxBASE+ users: Returns the fully-qualified DOS pathname for a file. The file can be sought using either the DOS environment PATH or FoxPro's SET PATH.

New in version 2.0: Relative path between two files can be returned.

FV( ) ● ❑

Returns the future value of an investment, at a given time and at a fixed interest rate.

FWRITE( ) ● ❑

Writes a specified number of bytes to a file.

GETBAR( ) ● ■

Returns the number of a bar in a specific popup position.

GETFILE( ) ● ❑

The GETFILE function displays the Open File dialog and returns the name of the file that the user chooses.

GETPAD( ) ● ■

Returns the name of a menu pad from its position in a menu bar.

GOMONTH( ) ● ❑

Returns the date which is exactly n months before or after a given date.

HEADER( ) ● ❑

Returns the number of bytes in the header of the database opened in the default work area or, optionally, in a designated work area.

INKEY( ) ● ❑

This function is enhanced so you can include the optional character expression to show or hide the cursor or to check for a click of the mouse button.

INLIST( ) ● ❑

Returns .T. if an expression is contained in a given expression list.

INSMODE( ) ● ❑

Tests or sets the insert/overwrite mode.

ISDIGIT( ) ● ❑

Returns .T. if the first character of an expression is a digit (0 - 9).

KEY( ) ● ■

For FoxBASE+ users: Returns the index expression for an open index in a work area.

New in version 2.0: Compound index support.

LASTKEY( ) ● ❑

Returns the decimal ASCII value of the key last pressed.

LIKE ( ) ● ❑

Uses wildcards to compare strings.

LINENO( ) ● ❑

Returns the line number to be executed in the program.

LOCFILE( ) ● ■

Locates a file on disk and returns the file name with its fully qualified path.

LOG10(<expN>) ● ❑

Returns the logarithm (base 10) of <expN>.

LOOKUP( ) ● ❑

Looks for a record in an unselected database.

MAX( ) ● ❑

The MAX function has been extended to accept any number of arguments and to operate on character data as well as numbers and dates.

MCOL( ) ● ❑

Returns the column position of the mouse pointer on the screen or in a window.

MDOWN( ) ● ❑

Returns a logical value corresponding to the state of the mouse button (pressed or not pressed).

MDX( ) ● ■

Returns the names of open .CDX compound index files.

MDY( ) ● ❑

Changes the date format to Month, DD, YY.

MEMLINES( ) ● ❑

Returns the number of word-wrapped lines in a memo field.

MEMORY( ) ● ❑

Returns the amount of RAM (in kilobytes).

MENU( ) ● ❑

Returns the name of the active menu bar.

MIN( ) ● ❑

Extended to accept any number of arguments and to operate on character data as well as numbers and dates.

MLINE( ) ● ❑

Retrieves a line from a memo field.

MLINE(<memo field>, <expN1>, <expN2>) ● ■

<expN2> returns memo line offset.

MRKBAR( ) ● ■

Returns true if a popup bar (option) is marked.

MRKPAD( ) ● ■

Returns true if a menu pad is marked.

MROW( ) ● ❑

Returns the row position of the mouse pointer on the screen or in a window.

NDX( ) ● ■

Compound index support.

NUMLOCK( ) ● ❑

Tests or sets the keyboard NumLock mode.

OBJNUM( ) ● ■

Returns the object number of a GET object.

OCCURS( ) ● ❑

Returns the number of occurrences of one string in another.

ORDER( ) ● ■

For FoxBASE+ users: Returns the name of the primary order index file.

New in version 2.0: Compound index support.

PAD( ) ● ❑

Returns the chosen menu pad name of the active menu bar.

PADL( ) ● ❑
PADC( )
PADR( )

Pads (or truncates) an expression to a specified length. Padding may be done on the left (PADL), both sides (PADC), or on the right (PADR).

PARAMETERS( ) ● ❑

Returns the number of parameters passed to the last FoxPro procedure. This is useful in conjunction with FoxPro's capability to pass variable numbers of parameters to a procedure.

PAYMENT( ) ● ❑

Returns periodic payment on a fixed interest loan.

PI( ) ● ❑

Returns the mathematical constant for the ratio of circumference to diameter.

POPUP( ) ● ❑

Returns the name of the active menu popup.

PRINTSTATUS( ) ● ❑

Returns the status of the printer.

PRMBAR( ) ● ■

Returns the prompt for a popup bar, i.e., text displayed on the bar.

PRMPAD( ) ● ■

Returns the prompt for a menu pad.

PROGRAM( ) ● ❑

Returns the name of the program that was running when an error occurred.

PROMPT( ) ● ❑

Returns the prompt of the last selected menu option or popup.

PROPER( ) ● ❑

Capitalizes an expression as is appropriate for proper names (i.e., with the first letter in each word capitalized).

PUTFILE( ) ● ❑

Displays the Save File dialog, allowing the user to name an output file.

PV( ) ● ❑

Returns the present value of equal payments invested for a given time at a fixed interest.

RAND( ) ● ❑

Returns a random number. Optionally includes specification of a seed value.

RAT( ) ● ❑

Returns the starting position of one string in another, *starting with the occurrence closest to the end.*

RATLINE( ) ● ❑

Returns the number of the line in which one character string *last* occurs in another.

RDLEVEL( ) ● ■

Returns the current READ level.

RECNO( ) ● ❑

An optional argument of 0 can be used with this function to specify use of "soft seek" logic. That is, if the record being sought is not found, the number of the next highest record in index-expression order is returned.

RELATION( ) ● ❑

Returns the linking expression for the n-th relation in either the default work area or a specified work area.

RTOD( ) ● ❑

Changes radians to degrees.

SCHEME( ) ● ❑

Returns the specified color SCHEME or the value of a single entry in the specified SCHEME.

SCOLS( ) ● ❑

Returns the number of columns on the console screen.

SECONDS( ) ● ❑

Returns the system time in seconds and milliseconds since midnight. The resolution of this timer is really one thousandth of a second.

SEEK( ) ● ❑

Determines if the index key is found or not.

SELECT( ) ● ■

For FoxBASE+ users: Returns the number of the currently selected work area when SET COMPATIBLE is OFF, and returns the highest numbered work area that's free when SET COMPATIBLE is ON.

New in version 2.0: 0 (zero) and 1 options.

SET( ) ● ■

For FoxBASE+ users: Returns the parameters of SET ON|OFF commands. If the optional second argument of 1 is specified, the function also returns other settings associated with the option. For instance, SET("CLOCK",1) will also return the row and column where the clock is being displayed.

New in version 2.0: Can return the procedure specified by the SET PROCEDURE command.

SIGN( ) ● ❑

Returns the mathematical sign of a number or expression.

SIN( ) ● ❑

Returns the sine from an angle (in radians).

SROWS( ) ● ❑

Returns the number of rows on the console screen.

STRTRAN( ) ● ❑

Searches one string for occurrences of another string, replacing them with a third string.

SYS(2005) ● ❑

Returns the name of the resource file.

SYS(2006) ● ❑

Returns the type of video adaptor and monitor in use.

SYS(2007) ● ❑

Returns the check sum of the string.

SYS(2008) ● ❑

Specifies the shape of the cursor in insert and overwrite modes.

SYS(2009) ● ❑

Swaps insert and overwrite cursors.

SYS(2011) ● ■

FoxPro/LAN environment: Returns current record or file lock status for the current work area.

SYS(2013) ● ■

Returns character string containing names of system menu bar, each of its pads, system menu popups, and each option on each popup.

SYS(2014) ● ■

Returns the minimum path between a file and the current directory, or the minimum path between a file and a specified directory.

SYS(2015) ● ■

Returns a unique 10-character procedure name beginnin with an underscore. The name is created from system date and system time.

SYS(2016) ● ■

Returns the window name included in the last SHOW GETS WINDOW command. Returns value only in a READ SHOW routine.

SYS(2017) ● ■

Clears the screen and displays the FoxPro sign-on screen.

SYS(2018) ● ■

Returns the name of the last parameter that was not found.

SYS(2019) ● ■

Returns the path of the CONFIG.FP file.

SYS(2020) ● ■

Returns total size of the disk in bytes.

TAG( ) ● ■

Returns tag names from .CDX compound index files, or names of .IDX index files.

TAN( ) ● ❑

Returns the tangent from an angle (in radians).

TARGET( ) ● ❑

Returns the target work area of the n-th relation in either the default work area or in a specified work area.

USED( ) ● ❑

Returns .T. if a database is open in a specified work area.

VARREAD( ) ● ❑

Returns the name of the field or memory variable being edited.

WBORDER( ) ● ■

Returns true if a window has a border.

WCHILD( ) ● ■

Returns either the number of child windows in a parent window or the names of the child windows in the order they are stacked in the parent window.

WCOLS( ) ● ❑

Returns the number of columns currently available in a particular window.

WEXIST( ) ● ❑

Returns .T. if a particular window exists.

WLAST( ) ● ■

Returns the name of the window last (not currently) active.

WLCOL( ) ● ❑

Returns the column location of the upper left corner of the window.

WLROW( ) ● ❑

Returns the row location of the upper left corner of a window.

WONTOP( ) ● ❑

Returns .T. if a particular window is frontmost.

WOUTPUT( ) ● ❑

Returns .T. if output is currently being directed into a specific *user* window. It can also change the window into which output is directed.

WPARENT( ) ● ■

Returns the name of a parent window for a child window.

WROWS( ) ● ❑

Returns the number of rows currently available in a specific window.

WVISIBLE( ) ● ❑

Returns .T. if a specific window is visible.

New System Memory Variables

_CUROBJ ● ■

Returns the object number of the current GET object.

_GENMENU ● ■

Holds the names and locations of the programs that are called when FoxPro generates a menu.

_GENSCRN

Holds the names and locations of the programs that are called when FoxPro generates a screen.

_MLINE

Stores the location of MLINE( )'s memo field offset.

_PRETEXT

Specifies a character expression to preface text merge lines.

_TALLY

Stores the number of records processed by the last database command, e.g., the last INDEX, JOIN, SORT command.

_TEXT

Stores the file handle of a low-level file to which text merge output is directed.

_THROTTLE

Specifies execution speed of programs when the Trace window is open.

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 TO 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 GOODIES directory and is called FOXPLUS.FP. 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+/Mac 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 GOODIES 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 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. SET COMPATIBLE supports a FOXPLUS or DB4 keyword. These keywords correspond to OFF and ON respectively. For example, you may use SET COMPATIBLE OFF and SET COMPATIBLE FOXPLUS interchangeably. You may also use SET COMPATIBLE ON or SET COMPATIBLE DB4 identically.

The default setting for COMPATIBLE is OFF (or FOXPLUS).

<table>
<tr><th>Command or Function</th><th>SET COMPATIBLE</th><th>Effect</th></tr>
<tr><td rowspan="2">All file processing commands with a drive reference.</td><td>DB4
(ON)</td><td>If a FoxPro path is set and a command specifies a drive, then only the drive specified in the command will be searched when looking for a file.</td></tr>
<tr><td>FOXPLUS
(OFF)</td><td>If a path is set and the command specifies a drive, the specified drive is searched, then the FoxPro path.</td></tr>
<tr><td colspan="3">An example:
SET PATH TO D:\TESTDIR
DO C:TEST

In this example, TEST is a program file located in D:\TESTDIR. It will not be found and executed if SET COMPATIBLE is DB4, because only the specified drive is searched. If SET COMPATIBLE is FOXPLUS, however, the file will be found and executed because the FoxPro path will also be searched.</td></tr>
<tr><td rowspan="2">PARAMETERs passed by reference</td><td>DB4
(ON)</td><td>PARAMETERs that are passed by reference (SET UDFPARMS ON) remain available to the called procedure.</td></tr>
<tr><td>FOXPLUS
(OFF)</td><td>PARAMETERs that are passed by reference (SET UDFPARMS ON) become “hidden” to the called procedure.</td></tr>
</table>

Command or Function	SET COMPATIBLE	Effect
@...GET...RANGE	DB4 (ON)	The RANGE is always checked.
	FOXPLUS (OFF)	The RANGE is only checked if data has been changed.
@...SAY (Special Characters)	DB4 (ON)	All special characters are output except CHR(7), which rings the bell.
	FOXPLUS (OFF)	All special characters are output including CHR(7), which does not ring the bell.
@...SAY (SCREEN SCROLL)	DB4 (ON)	Output that extends beyond the bottom right corner of the screen will be displayed, causing the screen to scroll upward.
	FOXPLUS (OFF)	Output that extends beyond the end of the screen is truncated.
@...SAY (SET STATUS ON)	DB4 (ON)	Output can overwrite the status bar. Text that extends beyond the end of the status display wraps above the status bar, scrolling upward from that point.
	FOXPLUS (OFF)	Output can not overwrite the status bar. Text that extends beyond the end of the screen is truncated.
@...SAY (w/ PICTURE)	DB4 (ON)	When numeric data is displayed with a PICTURE clause, the right-most digit in the PICTURE clause is rounded.
	FOXPLUS (OFF)	When data is displayed using a PICTURE clause, the value is not rounded — extra digits are simply truncated.

Command or Function	SET COMPATIBLE	Effect
ACTIVATE SCREEN and ACTIVATE WINDOW	DB4 (ON)	When a screen or window is activated, the cursor position is set to 00,00.
	FOXPLUS (OFF)	The cursor remains at its current position.
APPEND MEMO	DB4 (ON)	The default extension for the file will be .TXT if one is not specified.
	FOXPLUS (OFF)	There is no default extension and none is used if one is not specified.
GO | GOTO (with TALK ON)	DB4 (ON)	FoxPro outputs a message that indentifies the current work area alias and the record number positioned to.
	FOXPLUS (OFF)	FoxPro does not output the message indicating position.
MENUs and POPUPs	DB4 (ON)	Popups are placed in the active output window and, once activated, the cursor is positioned on an option in the popup. If the popup is placed on row zero, the entire row is used as part of the menu bar.
	FOXPLUS (OFF)	Popups are placed in their own windows and the cursor is not moved from its position in the active output window. If placed on row zero, only the pads of the menu are treated as part of the menu bar.

Command or Function	SET COMPATIBLE	Effect
PLAY MACRO	DB4 (ON)	When a keyboard macro in the range of A–Z is played (e.g., PLAY MACRO A), an implicit Alt+F10 is added before the letter and FoxPro executes the macro associated with the Alt+F10+letter combination, e.g., Alt+F10+A. With F1 through F9 keyboard macros, an implicit Alt is added before the Function key and FoxPro executes the macro associated with the Alt+Fkey combination (e.g., Alt+F1).
	FOXPLUS (OFF)	FoxPro executes the macro associated with the given name, without adding any implicit keystrokes. See the System Menu chapter in the FoxPro *Interface Guide* for a list of definable macro key combinations.

Command or Function	SET COMPATIBLE	Effect
READ (with a VALID clause on a GET)	DB4 (ON)	If Escape is pressed when positioned in a field that has a VALID clause, validation will be performed. If the input is invalid, the "Invalid Input" alert is displayed.
	FOXPLUS (OFF)	Validation is not performed if Escape is pressed, and if validation is performed (Escape is not pressed) the "Invalid Input" alert is not displayed if input is invalid.
READ and TRANSFORM (with a numeric PICTURE clause)	DB4 (ON)	The value of the variable or expression will be rounded to the number of decimal places specified in the PICTURE clause.
	FOXPLUS (OFF)	The value of the variable or expression will be truncated to fit the PICTURE clause.
READs (nested)	DB4 (ON)	When returning to a higher level READ, FoxPro does an implicit CLEAR GETS on the lower level before returning.
	FOXPLUS (OFF)	When returning to a higher level READ, pending GETs in the lower level will remain pending.
RUN/!	DB4 (ON)	The cursor moves to row 24, column 0. All subsequent output begins displaying from this point.
	FOXPLUS (OFF)	The cursor remains at its current position. All subsequent output begins displaying from this point.

Command or Function	SET COMPATIBLE	Effect
RUN/! (with STATUS ON)	DB4 (ON)	Output from the program that was RUN is scrolled up three lines before control returns to FoxPro.
	FOXPLUS (OFF)	Output from the program that was RUN is scrolled up two lines before control returns to FoxPro.
SET COLOR TO	DB4 (ON)	If SET STATUS is ON, the last line on the screen (n) and the second from last line (n-2) are redrawn with the new SET COLOR TO colors.
	FOXPLUS (OFF)	If SET STATUS is ON, the last three lines on the screen (n, n-1 and n-2) are all redrawn with the new colors.
SET FIELDS	DB4 (ON)	Using SET FIELDS TO without a field list or the ALL option will SET FIELDS OFF. This also removes all fields from the fields list.
	FOXPLUS (OFF)	Using SET FIELDS TO without a field list or the ALL option will SET FIELDS TO the null string (all fields are removed from the fields list).
SET MESSAGE	DB4 (ON)	The character expression specified in this command will appear immediately on the last line of the screen and the text will be output in a different color.
	FOXPLUS (OFF)	The character expression is only displayed when you SET STATUS ON, and both the character expression and the line it's on are displayed in a different color.

Command or Function	SET COMPATIBLE	Effect
SET PRINT TO <file>	DB4 (ON)	The output file is given a .PRT file extension, unless another extension is explicitly given.
	FOXPLUS (OFF)	The output file is given a file extension only when one is explicitly given.
STORE	DB4 (ON)	STORE cannot be used to initialize all elements of an array.
	FOXPLUS (OFF)	STORE may be used to initialize all elements of an array to a specified value.
SUM	DB4 (ON)	The number specified in SET DECIMALS is the number of decimal places that will be output in the SUM.
	FOXPLUS (OFF)	The number of decimal places (specified in the database structure) for the field being summed determines the number of decimal places that will be output.
INKEY( ) and LASTKEY( )	DB4 (ON)	Home and Shift+Home key combination will return the value 26 and Ctrl+Left returns a value of 1.
	FOXPLUS (OFF)	Refer to the table under INKEY( ) in the *Commands & Functions* manual for values.
LIKE( )	DB4 (ON)	The pattern and target are both trimmed of trailing blanks before the comparison is made.
	FOXPLUS (OFF)	The pattern and target are used "as is," and trailing blanks will be significant.

Command or Function	SET COMPATIBLE	Effect
SELECT( )	DB4 (ON)	SELECT( ) returns the number of the highest unused work area.
	FOXPLUS (OFF)	SELECT( ) returns the number of the currently selected work area.
SYS(2001,'COLOR')	DB4 (ON)	The value returned is the current setting of SET COLOR ON|OFF.
	FOXPLUS (OFF)	The value returned is the SET COLOR TO color pairs.

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 under 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 under 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 *Commands & Functions* manual or the Program Menu chapter in the FoxPro *Interface 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 *Interface 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 DOS Prompt

Another method for executing a program under FoxPro is to include the name of the program on the 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 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 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 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 have created in FoxPro 1.XX generally work in FoxPro 2.0 without any conversion. However, please read the following tips:

- Keyboard macros – Keyboard macros defined in earlier FoxPro versions may not work in FoxPro 2.0, 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 – Object files compiled in FoxPro 1.02 are automatically recompiled to run under version 2.0. FoxPro 1.XX will not automatically recompile FoxPro 2.0 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 2.0 should be run to allow 2.0 compatibility. 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: Browse preferences, Color sets, Calculator preferences, Puzzle, Modify memo windows, and Label layout windows.

- Labels – Label files from 1.XX can be opened transparently in FoxPro 2.0. For labels in 2.0 you can also add aliases to fields. In addition, the environment is saved differently.

  Once you save a label file in 2.0, 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 2.0 without conversion. Once saved in 2.0, attempting to open the same report file in 1.XX generates the message "Report file invalid." If you need to use a report in 2.0 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 2.0. For better performance, you

may wish to recreate your existing (.IDX) indexes in 2.0. 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+/Mac, you must use non-compact .IDX index files. Otherwise, if you build .IDX files always include COMPACT to benefit from FoxPro 2.0'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.

18 Multi-User FoxPro

The Multi-User version of FoxPro (FoxPro/LAN) has all the features of single-user FoxPro while allowing users to share database files.

The information provided here details the many features and functions of FoxPro/LAN. Information regarding the various networks supported by FoxPro/LAN is outside the scope of this documentation. If you have questions concerning the maximum number of users supported by a network, the minimum amount of memory needed or additional hardware that may be required by such systems, you should refer to your network documentation.

System Requirements

In order to operate FoxPro/LAN on a network, the following hardware and software requirements must be met.

Hardware

The minimum hardware requirements to run FoxPro/LAN are listed below. Your choice of Multi-User operating systems and networks may create additional hardware requirements. You should consult your network documentation for additional information regarding hardware specifications.

- A computer supported by FoxPro/LAN.
- To run the Standard version

At least 480K of free memory at the work stations — this is *after* the network shell has been loaded — or 440K free with expanded memory that is LIM 4.0 compatible.

- To run the Extended version

At least 2 MB of expanded memory

- A hard disk and expanded memory at each work station is recommended. Neither is mandatory, but optimum performance is achieved when both are available.

Software

Network software that supports the standard NETBIOS interface and is compatible with DOS 3.1 (or higher) is required. Most networks that are currently available meet these requirements. These include:

- 3Com 3+
- Banyan Vines
- IBM PC Network
- Novell Advanced NetWare (revision 1.02 or higher)
- Novell Netware 286
- Novell NetWare 386

- Lantastic
- Invisible Net
- *... and many, many others.*

NETBIOS Compatibility

While FoxPro/LAN requires network software that supports standard NETBIOS calls, this compatibility is often provided as an integral part of the network shell (as with Novell and 3Com). In these cases, the NETBIOS extensions (packaged in a module separate from the network shell) *do not* have to be loaded before FoxPro/LAN will operate correctly in the network environment.

Your network, however, may require that the NETBIOS module be loaded in order to provide basic NETBIOS compatibility. In these cases, the NETBIOS module must be loaded for FoxPro/LAN to operate correctly.

SHARE Module

Depending on the network being used, the SHARE module may or may not be required (for instance, it isn't required for Novell networks). When SHARE is not loaded, files located on local work stations are always opened for exclusive use. However, when SHARE is loaded files on the local drive can be opened non-exclusively. If data files are opened at a local work station when SHARE is loaded, they should be opened for exclusive use. (By default, FoxPro/LAN opens files for exclusive use unless you SET EXCLUSIVE OFF.)

If you are using DOS 4.x, consult your operating system documentation for recommendations on the use of SHARE with volumes larger than 32 megabytes.

Installation and Work Station Setup

The installation instructions for FoxPro/LAN can be found in the FoxPro *Installation Guide*. In addition, note that

- You may install FoxPro/LAN onto the file server's hard disk from a network work station or at the file server.
- You must obtain the proper file and directory creation rights for your network.
- Your license agreement allows you to install FoxPro/LAN on *one* machine only — typically the file server. As many work stations (or terminals) as you like may access a properly licensed copy on the server. However, no other copies of FoxPro/LAN should be stored on unlicensed machines.

When you're ready to install FoxPro/LAN, follow the installation instructions provided.

ADDUSER.PRG

Each work station on the network may want to have its own CONFIG.FP configuration file and FOXUSER resource file. The CONFIG.FP file contains a set of options that specify the default environment when FoxPro/LAN starts up. The FOXUSER file contains a variety of information affecting the local work station's FoxPro environment, including color sets, keyboard macros, preferences, system window locations and sizes, diary entries and so on.

Information in the FOXUSER file can be shared if the file is flagged READ ONLY with DOS ATTRIB. Information can be read but not saved to a read-only FOXUSER file. For a network application, this allows users to share color sets, keyboard macros, preferences, etc.

To make it easier to set up work stations, we've provided a program called ADDUSER. This program automates the creation of a custom CONFIG file and an initial FOXUSER file for each work station. It also allows you to optimally configure the placement of various temporary work files for each work station.

ADDUSER should be run at each work station that's attached to your network. Log the work station onto the network and execute FoxPro/LAN, then DO ADDUSER.

If either the configuration file or the resource file that's specified in the ADDUSER configuration screen does not exist, ADDUSER will use one or both of the files below, if available, for initial setup information.

CONFIG.DEF This file can be the default configuration file for each work station. Using this file as a starting point, ADDUSER creates the specified configuration file at the work station. It then adds the configuration information specifying where to place the overlay and work files.

DEFUSER.DBF This file can be the default resource file. If this file is found in the same directory from which ADDUSER was started, it's copied to the directory specified in the ADDUSER configuration screen using the specified resource file name.

If the configuration and/or resource file specified in the ADDUSER configuration screen already exists at the work station, ADDUSER uses the existing file(s) for initial setup information.

You create the CONFIG.DEF and DEFUSER.DBF files. When you create these files, they must be in the same directory from which ADDUSER is started.

Once started, ADDUSER presents an introductory screen which describes the ADDUSER program. At the bottom of the screen is the default directory that will be used in the next screen.

Make any changes to the drive or path specification, then press Enter and choose the appropriate option to continue or exit ADDUSER.

A screen is then displayed in which you specify a directory and/or file name for six configuration options. As you move through the options, the available disk space in each directory and a brief description of each option is displayed.

Configuration File

The configuration file contains a set of options that determines the default environment when FoxPro/LAN starts up. This file is typically small (less than 1024 bytes) but can be several thousand bytes if it contains many options. Because it's accessed only on start up, the file can be located on the network drive without degrading performance. However, the configuration file that's specified should have either a unique location or file name.

The new configuration file that's created by ADDUSER will contain the contents of the CONFIG.DEF file (if one exists in the startup directory). The data in CONFIG.DEF is followed by the additional settings for RESOURCE, OVERLAY, EDITWORK, SORTWORK and PROGWORK.

ADDUSER also adds a DOS SET command (if you choose OK) to the AUTOEXEC.BAT file on the work station which tells FoxPro/LAN where to find this configuration file.

Resource File

The resource file, FOXUSER, contains system and user-defined information that affects the local work station's FoxPro environment. This information includes color sets, keyboard macros, preferences, system window locations and sizes, diary entries and so on.

ADDUSER looks for the file DEFUSER.DBF in the startup directory. If the file exists, it copies the file to the specified directory, using the specified file name (FOXUSER is the default).

Each work station can have its own resource file, or can share a resource file residing on the file server.

A resource file with read-only privileges residing on the filer server can be shared by multiple work stations. When work stations share a common read-only resource file, any changes made on a work station to information normally saved in the resource file is discarded.

If a resource file on the file server has read and write privileges, the first user to gain access to the file opens it for exclusive use. When subsequent users attempt to gain access to the file, they cannot. Instead, FoxPro/LAN will start with no resource file open (RESOURCE will be set off). If your application requires the use of a resource file, you may want to check for its existence with SET('RESOURCE'). SYS(2005) will return the name and location of the resource file in use.

Because of the specific nature of the information contained in the resource file, you may want each work station to have its own FOXUSER file. Or, if your application does not rely on resources in the FOXUSER file, you may SET RESOURCE OFF and bypass the use of a resource file altogether. This can be accomplished by including the following line in the CONFIG file.

```
RESOURCE = OFF
```

The Overlay and Work Files

The FoxPro/LAN overlay and temporary work files should be placed on the local work station's hard drive (if possible) to reduce the amount of traffic between the server and the work station and to improve performance.

Exiting ADDUSER

When the configuration and file settings contain the proper file and path names, press F10 or PgDn. Four options are displayed.

These options allow you to **Modify** the settings, **Proceed** with the process, **Abort** the process or bring up the **Filer** to view files on the local work station or the file server. If you choose the Filer, press Escape when you're ready to return to this screen.

If you choose **Proceed**, ADDUSER changes the work station's AUTOEXEC.BAT file (with permission) and creates the configuration and resource files. It then displays a message indicating whether or not the process was successful.

Further Modifications

You can place any items you choose in the CONFIG.DEF and DEFUSER file. Because ADDUSER.PRG is a FoxPro program, it can be modified to provide further customization of the user setup process.

Running FoxPro/LAN

On a network, FoxPro/LAN is a shared application that resides on the file server's hard disk. When you invoke FoxPro/LAN, the application is loaded into the memory of the local work station from the file server.

Data files that will be used by FoxPro/LAN can reside on the file server or on the local work station. Data files on the file server can be shared and updated by more than one user, while files on a work station are available to just the work station.

To start FoxPro/LAN, type FOXL at the DOS prompt and press Enter. This executes the FoxPro/LAN loader. The loader checks the amount and type of memory available on your work station and then invokes the most advanced version of FoxPro/LAN available.

The loader allows you to invoke the most powerful version of FoxPro that will run on the machine without knowing the machine's configuration. This is especially useful in a LAN environment where the system administrator may be setting up on BAT file for many workstations.

You can execute a specific version of FoxPro/LAN (standard FoxPro/LAN or the Extended version of FoxPro/LAN) by typing the full name of the version. For example, if you want to execute the Extended version of FoxPro/LAN, type

```
FOXPROLX
```

or if you want to execute the Standard version, type

```
FOXPROL
```

at the DOS prompt.

For additional information on starting FoxPro/LAN and loader use, see the FoxPro *Installation Guide*.

System Configuration

Network performance can be greatly enhanced by using some or all of the FoxPro/LAN system configuration options.

FoxPro/LAN Components

When you install FoxPro/LAN, the three program files (FOXPROL.EXE, FOXPROL.OVL, FOXPROLX.EXE) and the loader (FOXL.EXE) are placed in a directory on the file server where they are accessible to all work stations. These four files should be situated in the same directory. However, FoxPro/LAN may also make copies of the .OVL overlay file elsewhere when it's started.

FOXPROL.EXE, FOXPROLX.EXE The main executable programs. One of these two files is read from the server into work station memory one time only when the program is started. FOXPROL.EXE is the executable program for the standard version of FoxPro/LAN; FOXPROLX.EXE is the executable program for the Extended version of FoxPro/LAN

FOXPROL.OVL The standard version's overlay file. This file should be available for rapid access while FoxPro/LAN is running. To optimize performance, it should be situated on the fastest available storage device that has enough room to contain it.

Temporary Work Files

During execution, FoxPro/LAN 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, this file should be situated 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 may require different color sets, certain users may wish to automatically execute a certain program at startup or a personalized keyboard macro definition file may 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/LAN. FoxPro/LAN looks here first.

If FoxPro/LAN cannot find a CONFIG file, FoxPro/LAN's default settings are used.

To use a configuration file that resides in a directory other than the one from which FoxPro/LAN is started, use one of the following:

- Include the directory in your PATH. FoxPro/LAN automatically searches the DOS PATH for a copy of CONFIG.FP if it cannot be found in the current working directory. FoxPro/LAN will use the first CONFIG.FP file it encounters. (See your DOS manual for instructions on setting a DOS PATH.)

- Use the DOS SET command to specify a DOS environmental variable that tells FoxPro/LAN which configuration file to use:

```
SET FOXPROCFG=<path name>
```

  Place this command in the work station's AUTOEXEC.BAT file. (See your DOS manual for full details on the SET command.)

- You can also specify the configuration file to use at startup by using the -C command line option:

```
FOXPROL -C<path name>
```

  or

```
FOXPROLX -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 DOS FOXPROCFG environment variable if one has been created.

Special CONFIG Options

Several CONFIG.FP network configuration options are available.

OVERLAY = <directory> [OVERWRITE]

With this option you specify a location for the FoxPro/LAN .OVL (overlay) file (Standard version only). At startup, FoxPro/LAN uses the .OVL file located in the same directory as the .EXE file. After reading the CONFIG.FP file, FoxPro/LAN looks in the specified directory for the .OVL file.

If the .OVL file is not present in the directory, the original .OVL file is copied to the directory. If the .OVL file exists in the directory, but its time stamp does not match the original's, FoxPro/LAN asks for permission to overwrite the .OVL file. If permission is granted, FoxPro/LAN overwrites the .OVL file with the new file.

If the optional OVERWRITE keyword is specified, FoxPro/LAN automatically overwrites the existing .OVL file without prompting for permission.

The Extended version of FoxPro ignores OVERLAY.

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 which 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 may want to put this file in a RAM disk or on a local work station drive. FoxPro/LAN 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 *Developer's Guide.*

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/LAN is started, it looks for the resource file that's specified in the CONFIG.FP file with:

RESOURCE = <pathname>

<pathname> may be either a directory or a fully-qualified path with the resource file name. If it's a directory, FoxPro/LAN looks for a resource file named FOXUSER in that directory. If a fully-qualified path and file name is included, FoxPro/LAN looks for the specified file.

If the CONFIG.FP configuration file does not contain a resource specification, FoxPro/LAN then searches the default directory for the resource file. If it cannot find the resource file in the default directory, FoxPro then searches the DOS path. If a FOXUSER file cannot be located, one is created in the default directory using default settings.

Multi-User Programming

On a network you must manage the inevitable collisions that occur when users share database files. To help you manage contention for database files, FoxPro/LAN provides complete record and file locking.

Exclusive Use versus Shared Use

FoxPro/LAN provides two methods of accessing database files: exclusive use and shared use.

Exclusive Use

When a database is opened for exclusive work station, 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

There are a few commands that require you to open a database for exclusive use.

- 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 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 under the section Automatic Locking Commands.)
- 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 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/LAN 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 (TOTAL, SUM, SELECT – SQL, SORT, etc.) that only read files.

In cases where completely current information is required (e.g., 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 command in this chapter.

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/LAN 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 on these commands, see the Multi-User Commands section in this chapter.

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 MULTILOCKS is SET OFF, locking another record (either manually or automatically) will release a record lock.
- Toggling MULTILOCKS 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 MULTILOCKS 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 may 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/LAN 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/LAN 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 may 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 is 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 you to handle 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 handle 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/LAN.

* 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
```

```
**************************************************
* 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 errnum, 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 errnum = 108
            line1 = "File cannot be locked."
            line2 = "Try again later..."

      * Error: Record in use by another.
      CASE errnum = 109 .OR. errnum = 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/LAN. Refer to the chapter Optimizing Your System in this manual for more information on improving performance.

Place the Temporary Files on a Local Drive

FoxPro/LAN creates its temporary files in the current 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 provides 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 are executed when no other users require access to the data (e.g., overnight updates) can benefit by opening the data files for exclusive use. When files are open for exclusive use, performance improves because FoxPro/LAN 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

This section describes the commands and functions that are relevant to FoxPro/LAN. Accompanying each command and function is a description of its operation. For further information on any of these commands or functions, see the FoxPro *Commands & Functions* manual.

BROWSE/CHANGE/EDIT

Display, edit and append database records

Format: **See the FoxPro *Commands & Functions* manual**

These commands let you display, edit and append database records.

When you BROWSE, CHANGE or EDIT a record in a database opened for shared use, FoxPro/LAN automatically attempts to lock the record when the you edit it. You can attempt to lock a record in BROWSE, CHANGE or EDIT by pressing Ctrl+O before editing the record. This record lock is an automatic lock; when you move to another record the lock is released. If you don't press Ctrl+O to lock the record, it will be autolocked when you begin typing.

You can also explicitly lock a record before entering BROWSE, CHANGE or EDIT. If MULTILOCKS is SET ON, the explicit record lock remains in place even if you move the record pointer to edit a different record. If MULTILOCKS is SET OFF, the explicit record lock remains in place if you move the record pointer, but the lock is released if you edit a different record. In both cases, the explicit record locks can be unlocked with the UNLOCK command.

Because one or more users on the network can have concurrent use of a database and its records, it's possible that a record you are viewing is currently being changed by another user on the network. SET REFRESH lets you specify if your screen will be updated with changes made by other users, and how often the updates will occur.

DISPLAY or LIST STATUS

Display the status of the FoxPro environment

Format: **DISPLAY STATUS**
or
LIST STATUS

These commands supply status information on the current FoxPro environment, including the following network specific items:

- The shared attribute status of each open database file
- The currently locked records in each database file
- The SET REPROCESS value
- The SET REFRESH value
- The SET EXCLUSIVE value
- The SET LOCK value
- The SET MULTILOCKS value

DISPLAY STATUS and LIST STATUS display the same information. With LIST STATUS, however, the information scrolls without pausing.

ERROR

Returns the number of the error that triggered an ON ERROR routine

Format: **ERROR( )**

ERROR( ) returns the number of the error which caused an ON ERROR condition. An ON ERROR routine must be active for ERROR( ) to return a value other than 0. This function is useful for determining the cause of recoverable error conditions, such as failed lock attempts.

ERROR( ) is reset by the RETURN and the RETRY commands.

See also: MESSAGE( ), ON ERROR, RETRY, RETURN

FLOCK

Attempt to lock a database file

Format: **FLOCK([<expN> | <expC>])**

FLOCK( ) attempts to place a lock on a database file. If a file lock is successfully placed, a logical true (.T.) is returned. If the lock attempt is unsuccessful, false (.F.) is returned. You can use FLOCK( ) from within both the Command window and programs.

If a file lock is successfully placed on a database, a logical true (.T.) is returned. The file is now available for both read and write access. Other users on the network will have read only access to the database. See the SET EXCLUSIVE or USE commands for information on how to lock a file and prevent all forms of access by other users.

A database remains locked until it is unlocked by the user who placed the lock. The database may be unlocked by issuing the UNLOCK command, locking an individual record, closing the database or quitting out of FoxPro/LAN. Databases may be closed with the USE, CLEAR ALL or CLOSE DATABASE commands.

If a record or file lock has already been placed by another user on the database you are attempting to lock, a logical false (.F.) is returned.

SET REPROCESS affects the number of lock attempts FLOCK( ) makes. With SET REPROCESS you may continue attempting to lock a file if the initial file lock attempt is unsuccessful. For more information about how SET REPROCESS controls file locking attempts, see SET REPROCESS.

You can place file locks on databases open in other work areas. If you don't specify a work area or alias, FLOCK( ) attempts to place a lock on the database file in the currently selected work area. To place a lock on a database in another work area, include the work area number <expN>, or the database alias <expC>.

You can establish relations between two or more database files with the SET RELATION command. Placing a file lock on a database that is related to one or more databases does not place a file lock on the related databases. You must explicitly place and remove file locks on the related databases.

In the following example, the SALESREP.DBF database is initialized at the end of the year. To reset all sales figures to zero REPLACE ALL is used. REPLACE ALL requires that a database file be locked. If FLOCK( ) is successful, the replacement is done. If the file cannot be locked, additional file locks are attempted for 3 seconds. If the file still cannot be locked, a error message is displayed.

Example

```
CLEAR
SET CURSOR OFF
SET REPROCESS TO 3 SECONDS
DEFINE WINDOW errwind FROM 2,2 TO 7,36 DOUBLE
USE salesrep.dbf

IF FLOCK( )
      *** Year end sales and commissions initialization ***
      REPLACE ALL ytdcomm  WITH 0.00
      REPLACE ALL ytdsales WITH 0.00
      WAIT "Initialization Done - Press any key" WINDOW
ELSE
      *** File is locked, warn user ***
      ACTIVATE WINDOW errwind
      @ 1,2 SAY "Unable to open the sales file"
      @ 2,2 SAY "      Please try later!"
      WAIT WINDOW
      DEACTIVATE WINDOW errwind
ENDIF
CLEAR WINDOW errwind
SET CURSOR ON
```

See also: LOCK( ), RLOCK( ), SET REPROCESS, SET RELATION, SYS(2011), UNLOCK, USE

MESSAGE

Returns the current error message or the contents of the line that caused the error

Format: **MESSAGE([1])**

MESSAGE( ) without an argument returns the current error message string. This form of MESSAGE( ) may be used with or without an ON ERROR condition in effect. With an ON ERROR condition, MESSAGE( ) and ERROR( ) can be used in error handling routines.

MESSAGE( ) with an argument of 1 returns the source code for the last line that caused an error in an active ON ERROR routine if the source code is available.

Unlike ERROR( ), MESSAGE( ) is not reset by the RETURN or RETRY commands.

See also: ERROR( ), ON ERROR, SET MESSAGE, SET NOTIFY

NETWORK

Returns true if you are using FoxPro/LAN

Format: **NETWORK( )**

This function returns a value of true (.T.) if you're using FoxPro/LAN. If you're running single-user FoxPro, it returns false (.F.).

This function provides the ability to conditionally execute routines based on the logical value returned by NETWORK( ), thus allowing both single-user and network specific commands and functions to be included in the same program.

RETRY

Re-execute the previous command

Format: **RETRY**

RETRY returns control to the calling program, and the last line executed in the calling program is re-executed. RETRY is similar to RETURN except that RETURN executes the next line.

RETRY is useful when a command should be repeated until it is successfully executed. RETRY is often used in an error recovery routine to retry execution of a command or function until a record or file can be locked.

SET REPROCESS can optionally be used to control the retries of a command that was unable to access a record or file. In most network situations SET REPROCESS is preferred.

See also: ON ERROR, RETURN, SET REPROCESS

RLOCK or LOCK

Attempt to lock a database record or records

Format: **RLOCK([<expN> | <expC1>]**
| [<expC2>, <expN> | <expC1>])
or
LOCK([<expN> | <expC1>]
| [<expC2>, <expN> | <expC1>])

RLOCK( ) and LOCK( ) are identical. With RLOCK( ), you may attempt to place a lock on a database record or a set of records. FoxPro/LAN, the network version of FoxPro, is required to lock database records or files.

If the lock or locks are successfully placed, a logical true value (.T.) is returned. Locked records are available for both read and write access to the user who placed the locks – read-only access to the records is available to all other users on the network. RLOCK( ) always returns true in the single-user version of FoxPro.

Executing RLOCK( ) does not guarantee that the attempted record lock or locks will be successfully placed. A record lock cannot be placed on a record already locked by another user, or on a record in a database file locked by another user. If the record lock or locks cannot be placed for any reason, a false (.F.) is returned.

You can issue RLOCK( ) from the Command window and programs.

SET REPROCESS affects the number of lock attempts RLOCK( ) makes. With SET REPROCESS you may continue attempting to lock a record if the initial lock attempt is unsuccessful. For more information about how SET REPROCESS controls record locking attempts, see the SET REPROCESS command in this chapter.

SET MULTILOCKS determines whether you may lock multiple records in a database. If MULTILOCKS is SET OFF (the default), you may only lock a single record in a database. Multiple records in a database may be locked when MULTILOCKS is SET ON. See SET MULTILOCKS in this manual for further information.

Parameters

<expN> | <expC1>

If you don't specify a work area or alias, RLOCK( ) attempts to lock the current record in the database file in the currently selected work area.

To attempt a record lock on the current record in a database open in another work area, include the work area number <expN>, or the database alias <expC1>.

<expC2>

To lock multiple records, include <expC2>. You must also include the work area or alias of the database in which you attempt multiple record locks, and MULTILOCKS must be SET ON.

<expC2> is a list of one or more record numbers. RLOCK( ) attempts to lock all of these records. The record numbers in <expC2> must be separated by commas. For example, to attempt record locks on the first four records in a database, <expC2> would contain '1,2,3,4'.

If MULTILOCKS is ON, you may also lock multiple records by moving the record pointer to the record you would like to lock, issuing RLOCK( ) or LOCK( ), and then repeating this for each record you want to lock.

If *all* records specified in <expC2> are successfully locked, true (.T.) is returned.

If just one of the records specified in <expC2> cannot be locked, false (.F.) is returned and none of the records will be locked. However, any previously existing record locks will remain in place. Multiple record locking is an additive process – placing additional record locks does not release locks on other records.

The maximum number of records that may be locked in each work area is approximately 8,000. The amount of memory on your server may reduce this number.

Unlocking Records

A database record may only be unlocked by the user who placed the lock. Record locks may be released by issuing UNLOCK , closing the database, quitting FoxPro/LAN, issuing SET MULTILOCKS OFF or ON, or FLOCK( ).

UNLOCK may be used to release record locks in the current work area (UNLOCK), a specific work area (UNLOCK IN <expN> | expC1>) or in all work areas (UNLOCK ALL). See UNLOCK in this guide for more information on its use and syntax.

Toggling MULTILOCKS from ON to OFF or from OFF to ON implicitly performs an UNLOCK ALL – all record locks in all work areas are released.

Databases may be closed with the USE, CLEAR ALL or CLOSE DATABASE commands.

Example

```
SET MULTILOCKS ON
SET REPROCESS TO AUTOMATIC
STORE '1,2,3,4' TO mreclist

USE customer IN A
USE states   IN B

? LOCK('1,2,3,4', 'A')    && Lock the first 4 records in customer
? RLOCK('1,2,3,4', 2)     && Lock the first 4 records in states
UNLOCK IN customer
UNLOCK IN states

? LOCK(mreclist, 'customer')    && Lock the 1st 4 records in customer
? RLOCK(mreclist, 'B')    && Lock the first 4 records in states
UNLOCK IN customer
UNLOCK IN states
```

See also: CLEAR, CLOSE, FLOCK( ), LOCK( ), SET MULTILOCKS, SET REPROCESS, SET RELATION, SYS(2011), UNLOCK, USE

SET EXCLUSIVE

Specify that databases be opened for exclusive or shared use

Format: **SET EXCLUSIVE ON | OFF**

SET EXCLUSIVE determines if database files opened on a network will be available for exclusive use by one user, or may be shared by every user.

Changing the setting of EXCLUSIVE does not change the status of previously opened databases. For example, if a database is opened with EXCLUSIVE SET ON, and EXCLUSIVE is later SET OFF, the database will still retain its exclusive use status until it is reopened.

Clauses

ON

A database opened on a network when EXCLUSIVE is SET ON is made available only to the user who opened the database. The database is not accessible to other users on the network. Unlike FLOCK( ), SET EXCLUSIVE ON also prevents read-only access to all other users. A file may also be opened on a network for exclusive use by including EXCLUSIVE with USE. It is not necessary to perform record or file locking on a database opened for exclusive use.

Opening a database for exclusive use assures that the file will not be changed by other users. Some FoxPro/LAN commands require that a database be opened for exclusive use before they may be executed. These commands are INSERT [BLANK], MODIFY STRUCTURE, PACK, REINDEX and ZAP.

You cannot open a file exclusively that another user has open.

The default for SET EXCLUSIVE is ON.

OFF

If a database is opened on a network with EXCLUSIVE SET OFF, the database may be shared and modified by any user on the network.

See also: FLOCK( ), LOCK( ), RLOCK( ), USE

SET LOCK

Enable/disable automatic record or file locking

Format: **SET LOCK OFF | ON**

You may enable or disable automatic file locking with SET LOCK. FoxPro/LAN does not place a lock on a file when executing commands which require read-only access to a database file. Here is a list of commands which require read-only access to a database:

Command		
AVERAGE	DISPLAY with a scope	REPORT
CALCULATE	INDEX	SORT
COPY TO	JOIN (both files)	SUM
COPY TO ARRAY	LIST	TOTAL
COUNT	LABEL	

These commands do not change the contents of a database file while they execute; by default access to the database is still available to other users on the network. It is possible that the database may be changed while you are executing one of these commands. For example, you may be printing a report and another user changes a record that has already been included in the report. Your report now contains dated information.

To insure that you are using the most current data you can SET LOCK ON. If LOCK is SET ON these FoxPro/LAN commands will automatically lock a database file. This prevents all access to the database file except for read-only use by other users on your network.

The same results in the above example can be achieved by issuing FLOCK( ), followed by a command in the table and UNLOCK. In this case the file locking is not done automatically.

To allow shared access of database files with the commands listed above, SET LOCK OFF. SET LOCK OFF may be used if it isn't essential to be using the most current information from a database file.

The default for SET LOCK is OFF.

SET MULTILOCKS

Enable/disable multiple record locking

Format: **SET MULTILOCKS OFF | ON**

FoxPro/LAN provides the option of attempting to lock more than one record in a database file. Depending on the setting of MULTILOCKS you may attempt to lock either a single record or a set of records. Records may be locked with the LOCK( ) or RLOCK( ) functions (LOCK( ) and RLOCK( ) can be used interchangeably).

> Note that toggling MULTILOCKS from ON to OFF or from OFF to ON implicitly performs an UNLOCK ALL – all record locks in all work areas are released.

ON

If MULTILOCKS is SET ON you may attempt to lock a set of records. Multiple record locks may be attempted by including in LOCK( ) or RLOCK( ) a set of record numbers or moving from record to record and issuing RLOCK( ) or LOCK( ). The record locks are attempted in the current work area, or if a work area alias, work area letter or number is included they may attempted in another work area.

OFF

If MULTILOCKS is SET OFF (the default value), a single record lock may be attempted with LOCK( ) or RLOCK( ) on the current database record. A record lock may be attempted in the current work area or in another work area if a work area alias, work area letter or number is included. If a record is locked, its lock will be released when a different record in the same database is locked.

Example

```
SET MULTILOCKS ON
SELECT A
USE customer
? LOCK("1,2,3,4,5", 1)
? RLOCK("1,2,3,4,5", "customer")
```

See also: LOCK( ), RLOCK( )

SET NOTIFY

Enable/disable the display of system messages

Format: **SET NOTIFY ON | OFF**

SET NOTIFY suppresses the display of system messages. If you SET NOTIFY ON, system messages are displayed. If you SET NOTIFY OFF, the display of system messages is suppressed. System messages are displayed in a window in the upper right corner of the screen.

Examples of system messages are the "Expression is valid" message in the Expression Builder dialog and the "Do Cancelled" message displayed when program execution is cancelled. WAIT WINDOW displays its message in the system message window.

When using FoxPro/LAN, the network version of FoxPro, SET NOTIFY affects the "Attempting to lock..." message that appears when REPROCESS is SET TO AUTOMATIC or 0. If NOTIFY is SET OFF, the message is not displayed. If NOTIFY is SET ON, the message is displayed.

The default for SET NOTIFY is ON.

See also: SET MESSAGE

SET PRINTER

Enable/disable output to a printer or port

Format 1: **SET PRINTER TO**
[\\<machine name>\<printer name>=<dest>]

Format 2: **SET PRINTER TO [\\SPOOLER[\NB]**
[\F=<n>][\B=<c>][\C=<n>][\P=<n>]
[\S=<server>][\Q=<queue>]]

Format 3: **SET PRINTER TO [<device> | <file>]**

FoxPro/LAN printer output can be redirected to a network, local device or to a file (as in the single-user version) using the SET PRINTER command. Once SET PRINTER has been issued, output prints or collects in a particular spool file until a new SET PRINTER command is issued. Issuing SET PRINTER without an option returns the printer to the default device (PRN).

Format 1

This command spools printer output to a network printer. The machine name is the network name assigned to the work station. This name is assigned by the network administrator and must be unique. The printer name is a name assigned to the printer and is also assigned by the network administrator. LPT1, LPT2 or LPT3 is used to identify the destination of the installed printer.

Format 2

This command is specifically designed for printing under Novell Advanced Netware revision 2.00 or higher. SPOOLER must be present for compatibility but is ignored — any character string may be substituted.

The options for this format are:

NB Suppresses printing a banner page (No Banner).

F=n This option specifies the form (by number) that the output should be printed on (0 to 255). This option is useful where printers are used for printing many different types of output: checks, invoices, letters, etc.

B=c Banner name. This option sets the heading printed on the banner page to an ASCII string of up to 12 characters. The default heading is the user name.

C=n This option specifies the number of copies to be queued (from 1 to 255). The default is 1.

P=n This option specifies the network printer where output will be sent, where n is the printer number assigned by the network. The default is 0.

S=Server This option specifies the network server the printer is attached to where output will be sent.

Q=Queue This option specifies the queue name assigned to the printer you will be sending output to.

Format 3

This command is for setting the printer to a local device or filename. Refer to the SET PRINTER command in the FoxPro *Commands & Functions* manual for more information.

See also: PRINTSTATUS( ), SYS(6), SYS(13), SYS(102)

SET REFRESH

Display changes made to a record by other users on a network

Format: **SET REFRESH TO <expN>**

FoxPro/LAN allows sharing of database files on a local area network. One or more users on the network may have concurrent use of a database and its records. It's possible that records you are viewing are simultaneously being edited by another user on the network. With SET REFRESH you may specify to update open BROWSE and memo windows with changes made by other users, and how often the updates will occur.

SET REFRESH affects records displayed on your screen with BROWSE and memo fields opened with MODIFY MEMO. Also affected are records displayed by CHANGE and EDIT commands (unformatted only). SET REFRESH does not affect any other commands.

<expN>

SET REFRESH accepts a numeric argument <expN> between 0 and 3,600. When REFRESH is SET TO 0 (the default value) and another user has changed a record displayed on your screen, it is updated with the changes when the other user unlocks the record.

A value between 1 and 3,600 causes the screen to be updated every <expN> seconds. If REFRESH is SET TO 1 screen updates occur every second, if REFRESH is SET TO 3,600 updates occur once an hour.

Regardless of the setting of REFRESH, if a record you are trying to lock is locked by another user on the network, you will receive the updated record when the lock is released by the other user. You will also receive the message "Record has changed."

See also: BROWSE, CHANGE, EDIT, MODIFY MEMO

SET REPROCESS

Specify action to take on unsuccessful record or file locks

Format: **SET REPROCESS TO AUTOMATIC**
| TO <expN> [SECONDS]

A record or file lock is not always successful on the first attempt. Frequently a record or file has been locked by another user on the network. SET REPROCESS lets you control how unsuccessful record and file locks are handled. If an initial record or file lock is unsuccessful, SET REPROCESS controls whether additional attempts will be made to lock the record or file. You may specify either the number of times additional attempts will be made, or how long the attempts will be made. The presence of an ON ERROR routine affects the reaction to an unsuccessful lock attempt.

The SET REPROCESS value, <expN>, can be set to an integer value between -2 and 32,000. You can also SET REPROCESS TO AUTOMATIC. The following describes the effect of different values of <expN> and AUTOMATIC on SET REPROCESS.

Clauses

TO <expN>

Specifies how many times FoxPro/LAN will try to lock a record after an unsuccessful attempt. The default value for <expN> is 0.

The following examples discuss the effect of specifying different values for <expN>.

SET REPROCESS TO 0

If you SET REPROCESS TO 0 (the default value) and issue a command or function that attempts to lock a record or file, FoxPro/LAN will attempt to lock the record or file indefinitely. The system message "Attempting to lock... Press ESC to Cancel." is displayed while FoxPro/LAN attempts to lock the record or file. If the record or file becomes available for locking while you wait, the lock is placed and the system message is cleared. True (.T.) is returned if a function placed the lock.

If you press Escape in response to the system message an appropriate alert is displayed (for example, "Record is in use by another."). If a function attempted to place the lock, the alert is not displayed and a false (.F.) is returned.

If an ON ERROR routine is in effect and a command is attempting to lock the record or file, the ON ERROR routine takes precedence over additional attempts to lock the record or file. The ON ERROR routine is immediately executed – no additional record or file locks are attempted and the system message is not displayed.

If you automatically attempt to place a lock when REPROCESS is set to 0 and an ON ERROR is in effect, the system message is not displayed, an ON ERROR routine will not be executed and a logical false (.F.) is immediately returned.

SET REPROCESS TO -1

If you SET REPROCESS TO -1, FoxPro/LAN attempts to lock the record or file indefinitely. You cannot cancel the locking attempts by pressing the Escape key, and an ON ERROR routine will not be executed.

A system message ("Waiting for lock...") will only be displayed if STATUS is SET ON.

If a lock has been placed by another user on the record or file you are attempting to lock you must wait until that user's lock is released. This could be a long, long time.

SET REPROCESS TO <expN> [SECONDS]

If the SET REPROCESS value <expN> is greater than 0 you may specify how many times or for how long FoxPro/LAN will attempt to lock a record or file.

For example, if you SET REPROCESS TO 30 FoxPro/LAN will attempt to lock a record or file up to 30 times. If you also include the optional keyword SECONDS, (e.g., SET REPROCESS TO 30 SECONDS), FoxPro/LAN will continuously attempt to lock a record or file for up to 30 seconds.

A system message ("Waiting for lock...") will only be displayed if STATUS is SET ON.

If an ON ERROR routine is not in effect and the record or file lock cannot be placed after the specified retries an appropriate alert is displayed (for example, "Record is in use by another."). If you manually attempted to place the lock, the alert is not displayed and a logical false (.F.) is returned after the specified retries.

If an ON ERROR routine is in effect and the lock attempts are unsuccessful, the ON ERROR routine is then executed. If you manually attempt the lock, an ON ERROR routine will not be executed and a logical false (.F.) is returned after the specified retries.

TO AUTOMATIC

If you SET REPROCESS TO AUTOMATIC (or identically SET REPROCESS TO -2), FoxPro/LAN will attempt to lock the record or file indefinitely. The system message "Attempting to lock... Press ESC to Cancel" is displayed while FoxPro/LAN attempts to lock the record or file. If the record or file becomes available to lock while you wait, the lock is placed and the system message is cleared. If you manually attempt to place the lock a logical true (.T.) is returned once the lock is successful.

If an ON ERROR routine is not in effect and you press Escape in response to the system message, an appropriate alert is displayed (for example, "Record is in use by another"). If you manually attempt to place the lock, the alert is not displayed and a logical false (.F.) is returned.

When an ON ERROR routine is in effect and Escape is pressed the ON ERROR routine is executed. If you manually attempt to place the lock an ON ERROR routine is not executed and a logical false (.F.) is returned.

See also: FLOCK( ), LOCK( ), ON ERROR, RLOCK( ), SET STATUS

SET STATUS

Enable/disable display of the status bar

Format: **SET STATUS OFF | ON**

SET STATUS affects the display of a status bar at the bottom of the screen. If STATUS is SET ON, the status bar is displayed showing the current drive, the active database file, the current record pointer position, number of records in the database file and the state of the NumLock, CapsLock and Insert keys. In FoxPro/LAN, the record or file lock status is also shown in the status bar. The status bar is updated each time a command is executed which changes the status information.

Issuing SET STATUS OFF removes the status bar from the screen.

With SET STATUS ON, messages such as "Exclusive Use," "File Locked," "Record Locked" and "Record Unlocked" are displayed in the center of the status line when appropriate.

Also, SET STATUS determines whether or not the message from SET REPROCESS TO -1 is displayed.

The default for SET STATUS is OFF.

See also: SET BRSTATUS

SYS(0)

Returns machine number

Format: SYS(0)

The SYS(0) function returns as a character string the machine number and machine name when using FoxPro/LAN. A machine number and name must first be assigned by the network software and the network shell must be loaded. If a machine number or name have not been assigned or the network shell has not been loaded, SYS(0) returns zero. Consult your network manual for further information on defining a machine number and name.

SYS(0) returns 1 when using single-user FoxPro.

SYS(2011)

Returns the current record or file lock status

Format: **SYS(2011)**

SYS(2011) returns, as a character string, the current record or file lock status for the current work area. Unlike FLOCK( ), LOCK( ) and RLOCK( ), SYS(2011) does not attempt to lock the file or record.

The character string returned by SYS(2011) is identical to the message displayed in the status bar (Exclusive, Record Unlocked, Record Locked ...).

UNLOCK

Release record or file locks

Format: **UNLOCK [IN <expN> | <expC> | ALL]**

UNLOCK releases a record lock, multiple record locks or a file lock from the currently selected database, from a database in another work area, or from databases in all work areas. Record and file locks may only be removed from databases by the user who issued the locks. Databases opened for exclusive use may not be unlocked with this command.

If you issue UNLOCK with no additional arguments, a record lock (or locks) or a file lock is released from the database in the currently selected work area.

When relations have been established between database files, releasing a record lock (or locks) or a file lock from one of the databases does not release locks from related records or files. You must explicitly release the record or file locks in each of the related files. You may release all locks on any related database files by issuing UNLOCK ALL (note that this will also release locks in unrelated files).

Clauses

IN <expN> | <expC>

If you don't specify a work area or alias, UNLOCK releases a record lock (or locks) or a file lock from a database in the currently selected work area. To release a record lock (or locks) or a file lock from a database in another work area, include the work area number <expN>, or the database alias <expC>.

ALL

Releases all record or file locks in all work areas.

See also: FLOCK( ), LOCK( ), RLOCK, SET EXCLUSIVE

USE ... EXCLUSIVE

Open a database file

Format: **USE <file> [EXCLUSIVE]**

The optional EXCLUSIVE keyword, in conjunction with USE, opens a database and its associated files for EXCLUSIVE use only. A database opened for exclusive use is made available only to the user who opened the database. The database is not accessible to other users on the network.

See also: CREATE, CLOSE, DBF( ), INDEX, SELECT, SET INDEX

Multi-User Error Messages

What follows is a list of the additional error messages that are specific to FoxPro/LAN. The corresponding error number is in parentheses following the error message text. See the appropriate appendix of the FoxPro *Developer's Guide* for a complete listing of FoxPro error messages.

Cannot write to a read-only file. (111)

An attempt was made to write to a file created for read-only purposes.

Exclusive open of file is required. (110)

INSERT[BLANK], MODIFY STRUCTURE, PACK, REINDEX or ZAP was attempted on a file that was not opened for exclusive use. Reopen the file exclusively and try again.

File is in use by another. (108)

An attempt was made to open a file that another user has open for exclusive use, or an attempt was made to exclusively open a file which another user has open.

Invalid printer redirection. (124)

Either a path to a printer is not established or the print device cannot be shared.

Record is in use by another. (109)

An attempt has been made to lock a record locked by another user.

Appendices

Appendix A – Customer Support

We sincerely hope you are pleased with your decision to purchase FoxPro 2.0 and that you find its use as rewarding as we do. We believe that FoxPro 2.0 is the finest database management system available on the market today and we stand behind our product and this claim by making ourselves ready to assist you in any way we can.

Please feel free to contact our Order Entry, Sales, Customer Service, Event Coordination, Developer Services and Technical Support Departments for the types of questions listed below. When you determine the department you need, read the appropriate section in this appendix before calling.

Ask the Order Entry department about

Upgrades and updates for Fox products you own and purchasing manuals, *Fox Software Developers Directory*, or items from the Fox Software Boutique (mugs, t-shirts and so forth).

Ask the Sales department about

Demonstration disks, general information and pricing about Fox products you don't currently own, information about forthcoming products and training centers in your area.

Ask the Customer Service department about

Replacing lost activation keys, changing your address, and orders made through Fox Software.

Ask the Event Coordination department about

Upcoming events in your area, Fox User Groups and Fox Software's Developer Conference.

Ask the Developer Services department about

Being listed as a developer or training site in the *Fox Software Developers Directory*.

Ask the Technical Support department about

Questions you have with Fox products you own.

Order Entry Department

The Order Entry department is there to help you process any orders you'd like to make through Fox Software. If you have questions about the products, contact the Sales Department.

Contact the Order Entry department for the following reasons:

- If you receive your Fox Software product on a media other than what you need. Fox example, FoxPro comes on 1.2 MB 5.25" disks. Order Entry can send you 720K 3.5" disks or 360K 5.25" disks free of charge.
- If you would like to order an update or upgrade for a product you already own.
- If you own a single-user version of a Fox Software product and would like to upgrade it to a network version. These upgrades are usually offered for the difference in retail price.
- If you would like to purchase extra sets of manuals for products you own.
- If you would like to order items from Fox Software's Boutique such as mugs, t-shirts, and so forth or the *Fox Software Developers Directory*.

Contact Fox Software Order Entry Department in the following ways:

Phone:	(419) 874-0162
Fax:	(419) 874-8678
CompuServe:	GO FOXFORUM
Mail:	Fox Software Order Entry Department 134 West South Boundary Perrysburg, OH 43551

The Order Entry Department is available Monday through Friday from 8:00 a.m. to 5:00 p.m. EST/EDT. Please ask for the Order Entry Department when you call.

Sales Department

You should contact the Sales Department for the following reasons:

- If you would like to know what features will be added in an update or upgrade Fox Software has announced.
- If you would like information on pricing for educational and non-profit copies of Fox products.
- If you tried the FoxGraph demo and would like to order the full package. Fox Software offers a special price for registered FoxBASE+ or FoxPro users.
- If you would like to receive literature or demo disks on a Fox Software product you don't currently own or would like to have one sent to a friend who isn't currently using a Fox Software database product.
- If you have questions about one of the FoxBASE+ or FoxBASE+/Mac Unlimited Runtime licenses, the FoxPro Distribution Kit or the FoxPro Library Construction Kit.
- If you have questions about products Fox Software has "in the works" or what platforms we are planning to support.
- If you are looking for a training center in your area.

Contact Fox Software Sales Department in the following ways:

Phone:	(419) 874-0162
or in the U.S.A.:	(800) 837-FOX2 – (800) 837-3692
Fax:	(419) 874-8678
CompuServe:	GO FOXFORUM, Section 10
Mail:	Fox Software Sales Department 134 West South Boundary Perrysburg, OH 43551

The Sales Department is available Monday through Friday from 8:00 a.m. to 5:30 p.m. EST/EDT. Please ask for the Sales Department when you call.

Customer Service

The Customer Service Department can assist if you have changed your address, lost your activation key or have questions about product registration.

Contact Fox Software Customer Service in the following ways:

Phone:	(419) 874-0162
Fax:	(419) 874-8678
CompuServe:	GO FOXFORUM
Mail:	Fox Software, Inc. Customer Service 134 West South Boundary Perrysburg, OH 43551

Customer Service is available Monday through Friday from 8:00 a.m. to 5:30 p.m. EST/EDT. Please ask for the Customer Service Department when you call.

Event Coordination Department

Contact Event Coordination for the following reasons:

- To find out when Fox Software will be in your area next. Fox attends most major trade shows and speaks at many database user groups around the country
- To find out if a Fox user group is located in your area, start a Fox user group or start a Fox special interest group (SIG) for your current user group
- To invite Fox Software to give a presentation at your user group
- To receive information about our annual Developer Conference

Fox Software Developer Conference

FoxPro, FoxBASE+/Mac, and FoxBASE+ users are welcome to attend Fox Software's annual Developer Conference. The conference features presentations by noted experts in the industry, professional developers and Fox Software personnel. Attenders also have an opportunity to meet Fox employees

Contact the Conference Coordinator through the Developer Conference Hotline year round to order previous conference documentation or ask questions about the upcoming conference.

Contact Event Coordination in the following ways:

Phone:	(419) 874-0162
Developer Conference Hotline:	(419) 872-0855
Fax:	(419) 874-8678
CompuServe:	GO FOXFORUM
Mail:	Fox Software, Inc. Event Coordination Department 134 West South Boundary Perrysburg, OH 43551

The Event Coordination Department is available Monday through Friday from 8:30 a.m. to 5:00 p.m. EST/EDT. Please ask for the Event Coordination Department when you call.

Developer Services

The Developer Services department coordinates the *Fox Software Developers Directory*. The current directory contains listings for over 750 developers and 350 third-party vertical market applications. It offers the large established base of Fox developers an opportunity to generate significant new revenues by opening up additional avenues for their consulting and custom application services.

Arranged geographically, the developer section of the *Fox Software Developers Directory* allows end users to locate firms that can assist in the development of custom applications or other consulting functions. The applications and utilities section of the directory logically groups database applications that address the needs of specific markets. The directory can be purchased through book stores and the Order Entry Department.

In addition, the Developer Services department maintains a current list of available trainers that includes both professional training centers and independent developers. This list, which places customers in touch with trainers, is packaged with our software, distributed at trade shows and user groups and mailed to customers upon request.

If you would like to be listed in the next *Fox Software Developers Directory*, contact our Developer Services department in one of these ways:

Phone:	(419) 874-0162
Fax:	(419) 874-8678
CompuServe:	GO FOXFORUM
Mail:	Fox Software, Inc. Developer Services Department 134 West South Boundary Perrysburg, OH 43551

The Developer Services Department is available Monday through Friday from 8:30 a.m. to 5:00 p.m. EST/EDT. Please ask for the Developer Services Department when you call.

Technical Support

The Technical Support Department at Fox Software is available to answer your questions about FoxPro 2.0.

All *registered* customers are eligible for free, unlimited technical support. Please remember to mail or fax in your registration card as soon as possible.

You can contact Technical Support in the following ways:

Phone:	(419) 874-0162
Fax:	(419) 874-4358
CompuServe:	GO FOXFORUM
Mail:	Fox Software, Inc. Technical Support/DOS 134 West South Boundary Perrysburg, OH 43551

Technical Support is available Monday through Friday from 8:30 a.m. to 5:00 p.m. EST/EDT. Please ask for the Technical Support Department and indicate the product for which you require assistance (FoxPro 2.0, FoxPro/LAN, and so forth).

When you call Technical Support:

- Be at your computer
- Have FoxPro 2.0 running
- Have your manuals within reach

You will be asked to provide information about your hardware and software configuration. For your convenience, we have included a Software Support Form at the end of this appendix that lists all of the questions Technical Support may need to ask you. Please have this form filled out when you call.

If you call Fox Software for Technical Support and all support technicians are currently handling other calls, your name goes into a call-back queue. The first call received is the first call returned.

The busiest time of day is from 11:00 a.m. to 4:00 p.m. EST/EDT. The best time to call is before 11:00 a.m. EST/EDT. Technical Support stops accepting incoming calls at 5:00 p.m. EST/EDT.

Faxes are handled in the same manner as telephone calls. You will receive confirmation that Fox Software's Technical Support Department *has received* your fax. Please fax in only reasonable amounts of code.

If you need to send in large amounts of code, send it on a disk.

Other Sources of Information

FoxPro's documentation is an excellent source of information about the product. If you are interested in additional sources of information, consider one of the following avenues:

- Classes on FoxPro in your area. Please consult the training list included with the product and the *Fox Software Developers Directory* for trainers in your area. Also check with local colleges and universities.
- *Fox Software Developers Directory*. This book can assist you in finding consultants and third party FoxPro applications.
- Become a member of the FOXFORUM on CompuServe. For more information about CompuServe, see the section later in this appendix.
- Attend Fox Software's Developers Conference. See the section on the Event Coordination Department in this appendix for more information.
- Current FoxPro books. A Third Party Book List is included with the product.
- User groups. Please contact the Event Coordination Department at Fox Software for a list of Fox user groups in your area.
- Database oriented magazines.

CompuServe

As a member you will be able to communicate directly with Fox Software and with thousands of other Fox users and developers.

Fox Forum — A Place to Share Ideas

- Thousands of CompuServe members use Fox products to develop database applications. Share your knowledge or tap theirs for programming tips that will save you time and money.
- Communicate directly with Fox Software. Members of the FOXFORUM can leave messages to Fox Software 24 hours a day.
- Professional developers and Fox Software support staff share hints and tips on applications created with FoxPro, FoxBASE+ and FoxBASE+/Mac.

Fox Forum — A Place to Receive Technical Support

- Questions are answered on an ongoing basis — within 24 hours (Monday-Friday). Special forum libraries contain files of commonly asked questions, hints, and tips.
- Obtain free software. Download software from professional developers and Fox Software in the Fox Forum libraries.

Free Introductory Offer

As a Fox Customer, you're entitled to a free introductory membership to CompuServe including:

- A usage credit
- A personal user ID number and password
- A complimentary subscription to CompuServe Magazine

> Receive your Free CompuServe Introductory Membership by calling 800-848-8199 and ask for representative #170. If you're already a CompuServe member, enter GO FOXFORUM at any ! prompt to access the Fox Forum.

Look Up Table

This section is designed to assist you in finding answers to questions you may have about FoxPro.

Environment Settings		
Topic	**Chapter**	**Source**
CONFIG.SYS	Customizing FoxPro Optimizing Your System	*Developer's Guide* *Developer's Guide*
CONFIG.FP	Customizing FoxPro "Program Limitations and Miscellaneous Notes" in Documenting Applications with FoxDoc	*Developer's Guide* *Developer's Guide*
AUTOEXEC.BAT	"Startup Files" in Customizing FoxPro	*Developer's Guide*
Extended and Expanded Memory	Optimizing Your System Customizing FoxPro	*Developer's Guide* *Developer's Guide*
Error messages/ error handling	Error Messages Multi-User FoxPro	*Developer's Guide* *Developer's Guide*

Installation		
Topic	**Chapter**	**Source**
Activation Key		*FoxPro Installation Guide*
Low density disks	"Order Entry Department" in Customer Support	*Developer's Guide*

Product Differences		
Topic	**Chapter**	**Source**
FoxPro 1.02/FoxPro 2.0	Compatibility	*Developer's Guide*
FoxBASE+/FoxPro 2.0	Compatibility	*Developer's Guide*

Printing		
Topic	**Chapter**	**Source**
CONFIG.FP (TIME=)	Customizing FoxPro	*Developer's Guide*
Printer drivers		*Late Breaking News*
Network printing	FoxPro/LAN	*Developer's Guide*

Source Code		
Topic	**Chapter**	**Source**
Debugging	Debugging Your Application	*Developer's Guide*
Optimizing	Optimizing Your Application	*Developer's Guide*
Documenting	Documenting Your Application with FoxDoc	*Developer's Guide*
Command and functions	The FoxPro Language	*Commands & Functions* On-Line Help

Report Writer		
Topic	**Chapter**	**Source**
Quick report	Sessions 3 and 4 Report Writer	*Getting Started* *Interface Guide*
UDFs	Report Writer	*Interface Guide*
Report variables	Report Writer Report Variable Hints	*Interface Guide* *Developer's Guide*
Removing blank lines		*Late Breaking News*
Database structures	Tables	*Developer's Guide*

Label Designer		
Topic	**Chapter**	**Source**
Removing blank lines	Label Designer	*Interface Guide*
Database structures	Tables	*Developer's Guide*

Menu Builder		
Topic	**Chapter**	**Source**
Creating	Menu Builder	*Interface Guide*
Debugging	Menus	*Developer's Guide*
Generating code	Menus	*Developer's Guide*
Database structures	Menu Builder	*Interface Guide*

Screen Builder		
Topic	**Chapter**	**Source**
Creating	Session 8 Screen Builder	*Getting Started* *Interface Guide*
Debugging	Screens	*Developer's Guide*
Generating Code	Session 8 Screen Builder Screens	*Getting Started* *Interface Guide* *Developer's Guide*
Database Structure	Tables	*Developer's Guide*

Project Manager		
Topic	**Chapter**	**Source**
General	Project Manager Project — The Main Organizing Tool	*Interface Guide* *Developer's Guide*

RQBE		
Topic	**Chapter**	**Source**
Creating	Sessions 2 and 4 RQBE	*Getting Started* *Interface Guide*
Quick report	Sessions 2 and 4 RQBE	*Getting Started* *Interface Guide*
Data grouping	Session 2 RQBE	*Getting Started* *Interface Guide*

SQL		
Topic	**Chapter**	**Source**
General	Session 5 SELECT — SQL SQL Quiz	*Getting Started* *Commands & Functions* *Developer's Guide*
Additional Features		*Late Breaking News*

Multi-User		
Topic	**Chapter**	**Source**
ON ERROR routines	Multi-User FoxPro	*Developer's Guide*
File and record locking	Multi-User FoxPro	*Developer's Guide*

Color		
Topic	**Chapter**	**Source**
Advanced Discussion	Customizing FoxPro	*Developer's Guide*
Color Picker	Window Menu	*Interface Guide*
SET COLOR commands		*Commands & Functions*

Software Support Form

Category of Problem: (check all that apply)

[ ] General [ ] Printing [ ] Installation
[ ] Source Code [ ] Report Writer [ ] Label Designer
[ ] Menu Builder [ ] Screen Builder [ ] Project Manager
[ ] RQBE/SQL [ ] LAN [ ] Other
[ ] Incompatibility with FoxBASE+ 2.10 [ ] Cannot work around
[ ] Incompatibility with FoxPro 1.02 [ ] Can work around
[ ] Request for a change or enhancement
[ ] Critical problem
[ ] Error — program gives incorrect results
Error number encountered, if applicable. _____
Line of code error encountered on, if applicable.
_____
_____

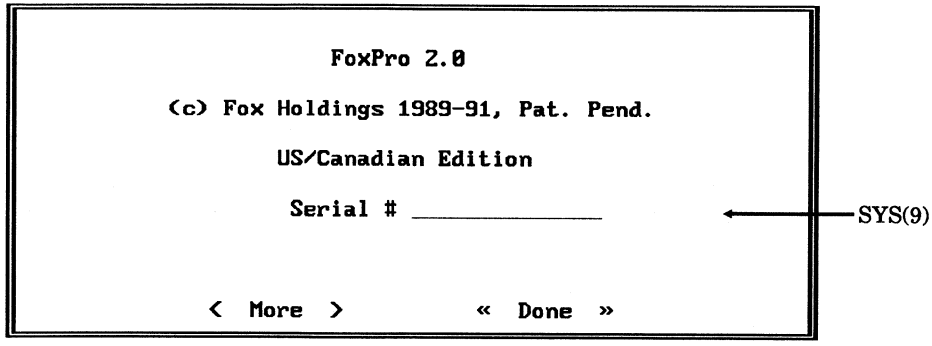
Enclosures:

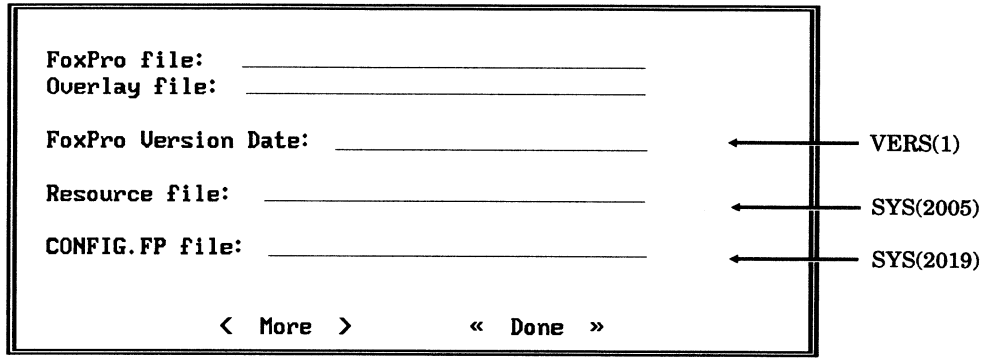
[ ] Floppy disk [ ] Additional notes
[ ] Console output [ ] Other_____
[ ] Program listing

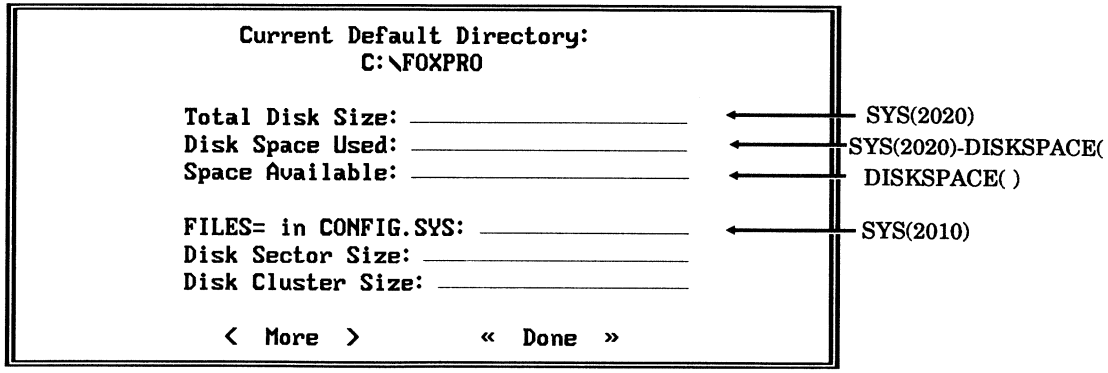
Configuration:

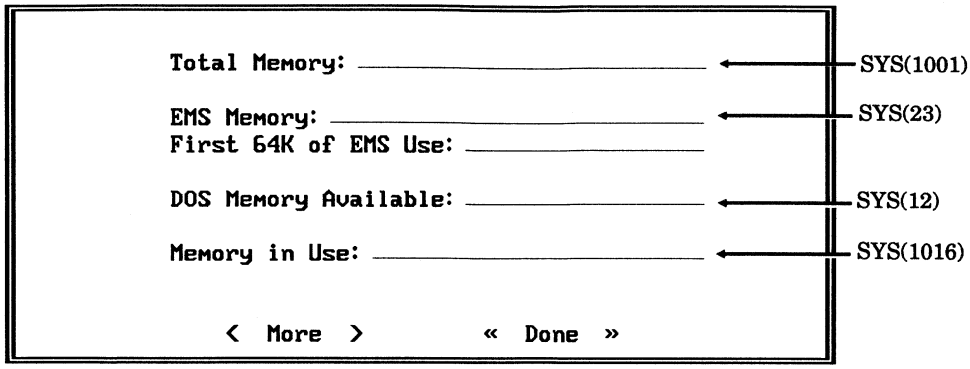
Operating System _____
Computer Make / Model _____
Available Memory Before Running FoxPro 2.0 _____
Expanded Memory Brand _____
Amount _____
Mouse [ ] Yes [ ] No Brand _____
Video Card Brand _____
Network _____ Network Revision _____
Setting of COMPATIBLE [ ] ON [ ] OFF

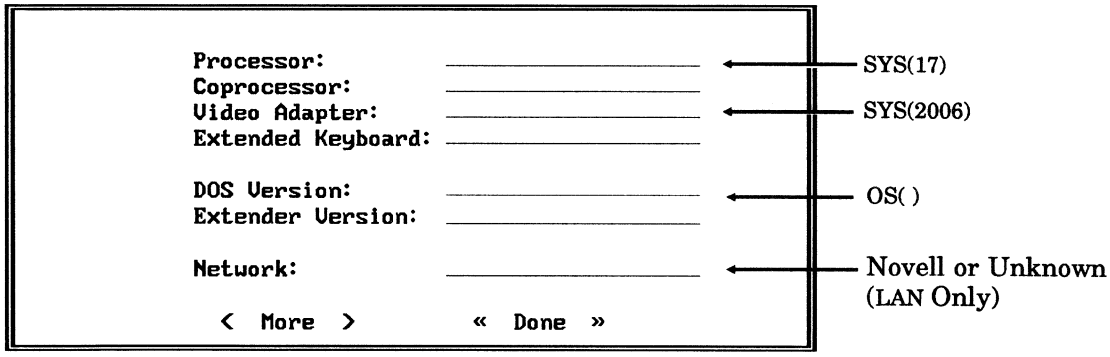
The pictures on the following two pages are the first five screens of the **About FoxPro...** box. Lines have been left for you to fill in your own configuration information. The accompanying functions for each configuration information are pointed out in red where applicable. To show the first screen, choose **About FoxPro...** from the **System** menu popup. Choose the **More** push button to go to each following screen and choose the **Done** push button to cause the **About FoxPro...** box to disappear.


FoxPro 2.0
(c) Fox Holdings 1989-91, Pat. Pend.
US/Canadian Edition
Serial # _____
< More >
« Done »
SYS(9)


FoxPro file: _____
Overlay file: _____
FoxPro Version Date: _____
Resource file: _____
CONFIG.FP file: _____
< More >
« Done »
VERS(1)
SYS(2005)
SYS(2019)


Current Default Directory:
C:\FOXPRO
Total Disk Size: _____
Disk Space Used: _____
Space Available: _____
FILES= in CONFIG.SYS: _____
Disk Sector Size: _____
Disk Cluster Size: _____
< More >
« Done »
SYS(2020)
SYS(2020)-DISKSPACE( )
DISKSPACE( )
SYS(2010)





AUTOEXEC.BAT file. Write all lines in the order they appear.

CONFIG.SYS file. Write all lines in the order they appear.

CONFIG.FP file. Write all lines in the order they appear.

Brief Problem Description:

Appendix B – Tables

This appendix contains the following tables:

- ASCII Chart
- File Types and Extensions
- System Capacities
- Control Key Shortcuts
- Database Structures
  - .PJX Database (Projects)
  - .SCX Database (Screens)
  - .FRX Database (Reports)
  - .MNX Database (Menus)
  - .LBX Database (Labels)
- Other File Structures
  - Database File (.DBF)
  - Memo File (.FPT)
  - Index File (.IDX)
  - Compact Index File (.IDX)
  - Compound Index File (.CDX)
  - FoxPro 1.x Report File (.FRX)
  - FoxPro 1.x Label File (.LBX)
  - FoxBASE+ Memo File (.DBT)
  - Macro File (.FKY)

ASCII Chart

dec	hex	scr
0	00	
1	01	☺
2	02	●
3	03	♥
4	04	♦
5	05	♣
6	06	\
7	07	•
8	08	■
9	09	○
10	0A	⊠
11	0B	♂
12	0C	♀
13	0D	♪
14	0E	♫
15	0F	☼
16	10	►
17	11	◄
18	12	↕
19	13	‼
20	14	¶
21	15	§
22	16	▬
23	17	↨
24	18	↑
25	19	↓
26	1A	→
27	1B	←
28	1C	∟
29	1D	↔
30	1E	▲
31	1F	▼

dec	hex	scr
32	20	
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

dec	hex	scr
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_

dec	hex	scr
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	⌂

dec	hex	scr
128	80	Ç
129	81	ü
130	82	é
131	83	â
132	84	ä
133	85	à
134	86	å
135	87	ç
136	88	ê
137	89	ë
138	8A	è
139	8B	ï
140	8C	î
141	8D	ì
142	8E	Ä
143	8F	Å
144	90	É
145	91	æ
146	92	Æ
147	93	ô
148	94	ö
149	95	ò
150	96	û
151	97	ù
152	98	ÿ
153	99	Ö
154	9A	Ü
155	9B	¢
156	9C	£
157	9D	¥
158	9E	₧
159	9F	ƒ

dec	hex	scr
160	A0	á
161	A1	í
162	A2	ó
163	A3	ú
164	A4	ñ
165	A5	Ñ
166	A6	ª
167	A7	º
168	A8	¿
169	A9	⌐
170	AA	¬
171	AB	½
172	AC	¼
173	AD	¡
174	AE	«
175	AF	»
176	B0	▒
177	B1	▓
178	B2	▓
179	B3	│
180	B4	┤
181	B5	╡
182	B6	╢
183	B7	╖
184	B8	╕
185	B9	╣
186	BA	║
187	BB	╗
188	BC	╝
189	BD	╜
190	BE	╛
191	BF	┐

dec	hex	scr
192	C0	└
193	C1	┴
194	C2	┬
195	C3	├
196	C4	─
197	C5	┼
198	C6	╞
199	C7	╟
200	C8	╚
201	C9	╔
202	CA	╩
203	CB	╦
204	CC	╠
205	CD	═
206	CE	╬
207	CF	╧
208	D0	╨
209	D1	╤
210	D2	╥
211	D3	╙
212	D4	╘
213	D5	╒
214	D6	╓
215	D7	╫
216	D8	╪
217	D9	┘
218	DA	┌
219	DB	▉
220	DC	▄
221	DD	▌
222	DE	▐
223	DF	▀

dec	hex	scr
224	E0	∝
225	E1	ß
226	E2	Γ
227	E3	π
228	E4	Σ
229	E5	σ
230	E6	µ
231	E7	τ
232	E8	⌽
233	E9	Θ
234	EA	Ω
235	EB	δ
236	EC	∞
237	ED	φ
238	EE	∈
239	EF	∩
240	F0	≡
241	F1	±
242	F2	≥
243	F3	≤
244	F4	⌠
245	F5	⌡
246	F6	÷
247	F7	≈
248	F8	°
249	F9	∙
250	FA	·
251	FB	√
252	FC	n
253	FD	²
254	FE	■
255	FF	

File Types and Extensions

File Type	Extension
Database	.DBF
Memo	.FPT
Memo Backup	.TBK
FoxBASE+ Style Memo	.DBT
Index	.IDX
Compound Index	.CDX
Program	.PRG
Compiled Program	.FXP
Format	.FMT
Compiled Format	.PRX
View	.VUE
Text	.TXT
File Backup	.BAK
Report	.FRX
Report Memo	.FRT
Label	.LBX
Label Memo	.LBT
Screen	.SCX
Screen Memo	.SCT
Generated Screen Program	.SPR
Compiled Screen Program	.SPX

File Type	Extension
Menu	.MNX
Menu Memo	.MNT
Generated Menu Program	.MPR
Compiled Menu Program	.MPX
Generated Query Program	.QPR
Compiled Query Program	.QPX
Project	.PJX
Project Memo	.PJT
Generated Application	.APP
Executable Program	.EXE
Compilation Error File	.ERR
Memory Variable Save	.MEM
Macro Files	.FKY
Window File	.WIN
Resource Files	FOXUSER.DBF FOXUSER.FPT FOXUSER.CDX
Help Files	FOXHELP.DBF FOXHELP.FPT
Configuration File	CONFIG.FP
Temporary File	.TMP
FoxDoc Reports	.DOC
FoxDoc Action Diagrams	.ACT
Libraries	.PLB

System Capacities

System Capacities		
	FoxPro	FoxPro Extended
Database and Index Files		
Maximum # of records per database file	1 billion[1]	1 billion[1]
Maximum # of characters per record	4,000	4,000
Maximum # of fields per record	255	255
Maximum # of databases open at one time	25	25
Maximum # of characters per database field	254	254
Maximum # of characters per index key (.IDX)	100	100
Maximum # of characters per index key (.CDX)	254	254
Maximum # of open index files per database	unlimited[2]	unlimited[2]
Maximum # of open indexes in all work areas	unlimited[2]	unlimited2
Field Characteristics		
Maximum size of character fields	254	254
Maximum size of numeric (and float) fields	20	20
Maximum number of characters in field names	10	10
Digits of precision in numeric computations	16	16
Memory Variables and Arrays		
Default # of memory variables	256	256
Maximum # of memory variables	3,600	65,000
Maximum # of arrays	3,600	65,000
Maximum # of elements per array	3,600	65,000
Program and Procedure Files		
Maximum # of lines in source program files	unlimited	unlimited
Maximum size of compiled program modules[3]	64K	64K
Maximum # of procedures per file	unlimited	unlimited

System Capacities		
	FoxPro	FoxPro Extended
Program and Procedure Files		
Maximum # of nested DO calls	32	32
Maximum # of READ nesting levels	4	4
Maximum # of structured programming commands	64	64
Report Writer Capacities		
Maximum # of objects in a report definition	unlimited	unlimited
Maximum # of lines in a report definition	255	255
Maximum # of grouping levels	20	20
Window Support		
Maximum # of open windows (all types)	unlimited	unlimited
Maximum # of open Browse windows	25	25
Miscellaneous Capacities		
Maximum # of characters per character string	64K	2 gigabytes
Maximum # of characters per command line	2,048	2,048
Maximum # of open files	99	DOS limit
Maximum keystrokes in keyboard macro	1024	1024
Maximum fields in a SQL SELECT statement	256	256
Color Support		
Number of color schemes per color set	24	24
Maximum # of color sets (in FOXUSER file)	unlimited	unlimited
Number of colors per color scheme	10	10
Schemes available for user definition	8	8

[1]The actual file size (in bytes) cannot exceed 2 gigabytes for single-user or exclusively opened multi-user .DBF files. Shared multi-user .DBF files with no indexes or .IDX indexes cannot exceed 1 gigabyte. Shared multi-user .DBF files with structural .CDX indexes cannot exceed 2 gigabytes.
[2]Limited by memory and available file handles. .CDX files use only one file handle.
[3]A program *module* is one procedure. A program or application can contain an unlimited number of program modules.

Control Key Shortcuts

Key	Action
Ctrl+A	Select All
Ctrl+C	Copy selected items to clipboard
Ctrl+D	Specify a program to execute
Ctrl+E	Replace text and find next occurrence
Ctrl+F	Find specified text
Ctrl+G	Find next occurrence of specified text
Ctrl+K	Locate next occurrence of specified record
Ctrl+M	Resume executing a suspended program
Ctrl+Q	Exit current edit without saving changes; Exit dialogs without taking action
Ctrl+R	Redo a text editing action you just undid
Ctrl+U	Undo a text editing action
Ctrl+V	Paste item from the clipboard
Ctrl+W	Exit current edit and save any modifications
Ctrl+X	Remove selected items and save on clipboard
Ctrl+F1	Cycle windows
Ctrl+F2	Display Command window
Ctrl+F7	Move frontmost window
Ctrl+F8	Size frontmost window
Ctrl+F9	Minimize frontmost window
Ctrl+F10	Zoom frontmost window
Esc	Exit current edit without saving changes; Exit dialogs without taking action

Data File Structures

The tables on the following pages contain the structures of the Project, Menu, Screen, Report and Label data files. These tables show the fields in the data files and the type of information stored in the fields. The structures of these data files are subject to change.

.PJX Database (Projects)

Fields		Object Type and Field Data				
	Type	Header	Screen Set	Screen	Program	Menu
NAME [1]	C	text	set name	name	name	name
TYPE	C	H	S	S	P	M
TIMESTAMP	N	time stamp	time stamp	time stamp	time stamp	time stamp
OUTFILE	M	location [2]	output file [1]			output file [1]
HOMEDIR	M	homedir	homedir			homedir
SETID	N	highest id	key number	key number		
EXCLUDE	L		exclude?		exclude?	exclude
MAINPROG	L		main?		main?	main?
ARRANGED	L			arranged?		
SAVECODE	L	saved?				
DEFNAME	L		defaulted?			
OPENFILES	L		open files?			
CLOSEFILE	L		close files?			
DEFWINDS	L		define winds?			
RELWINDS	L		release winds?			
READCYCLE	L		cycle?			
MULTREAD	L		multiple?			
MODAL	L		modal?			
ASSOCWINDS	M		window list			
DEBUG	L	debug?				
ENCRYPT	L	encrypt?				
SCRNORDER	N			order number		
SCRNROW	N			vpos		
SCRNCOL	N			hpos		
CMNTSTYLE	N	box/asterisk				
OBJREV	N		obj rev		obj rev	obj rev
COMMANDS	M		bitmap		bitmap	bitmap
DEVINFO	M	developer info				
SYMBOLS	M		symbol table		symbol table	symbol table
OBJECT	M		obj code		obj code	obj code
CKVAL	N					

[1]Includes normalized path.
[2]Location of the generated code (<Source>, <Project> or path)

Object Types and Field Data								
Query	**Report**	**Label**	**Library**	**Format**	**Database**	**Index**	**File**	**Application**
name	name	name	name	name	name	name	name	name
Q	R	B	L	F	D	I	x	Z
time stamp	time stamp	time stamp	time stamp	time stamp	time stamp	time stamp	time stamp	time stamp
exclude?	exclude?	exclude?	exclude?	exclude?	exclude?	exclude?	exclude?	always
main?								
obj rev				obj rev				
bitmap	bitmap	bitmap		bitmap				
symbol table	symbol table	symbol table	symbol table	symbol table				
obj code				obj code				

.SCX Database (Screens)

Fields	Object Types and Field Data							
OBJTYPE	N	SCREEN (1)	WORKAREA (2)	INDEX (3)	RELATION (4)	TEXT (5)	BOX (7)	GROUP (10)
OBJCODE	N	version (10)	1-25	1-25	1-25	SAY (8)	BOX/BOX... (3-6)	
NAME	M	window	file.dbf	file.idx				
EXPR	M		set skip	indexexpr	relexpr	thetext		
VPOS	N	vpos				vpos	vpos	objnumber
HPOS	N	hpos				hpos	hpos	objcount
HEIGHT	N	height				height	height	
WIDTH	N	width				width	width	
STYLE	N	USR/DLG..						
PICTURE	M							
ORDER	M		index name					
UNIQUE	L		current=.T.	unique?				
COMMENT	M							
ENVIRON	L	environ?						
BOXCHAR	C						boxchar	
FILLCHAR	C						fillchar	
TAG	M	title	alias	for expr	into alias			
TAG2	M	footer	tag name	to alias	from alias			
SCHEME	N	schemeno				schemeno	schemeno	
SCHEME2	N	schemeno						
COLORPAIR	N					col pair	col pair	
LOTYPE	N							
RANGELO	M							
HITYPE	N							
RANGEHI	M							
WHENTYPE	N	expr/code						
WHEN	M	when						
VALIDTYPE	N	expr/code						
VALID	M	valid						
ERRORTYPE	N							
ERROR	M							
MESSTYPE	N							
MESSAGE	M							
SHOWTYPE	N	expr/code						
SHOW	M	show						
ACTIVTYPE	N	expr/code						
ACTIVATE	M	activate						
DEACTTYPE	N	expr/code						
DEACTIVATE	M	deactivate						
PROCTYPE	N	expr/code						
PROCCODE	M	proc code						
SETUPTYPE	N	expr/code						
SETUPCODE	M	setup code						
FLOAT	L	float?						
CLOSE	L	close?						
MINIMIZE	L	minimize?						
BORDER	N	border						
SHADOW	L	shadow?						
CENTER	L	center?						
REFRESH	L					refresh?		
DISABLED	L							
SCROLLBAR	L							
ADDALIAS	L	add alias?						
TAB	L							
INITIALVAL	M							
INITIALNUM	N	init obj						
SPACING	N							

Object Types and Field Data						
LIST (11)	TEXTBUTN (12)	RADIOBUTN (13)	CHECKBOX (14)	GETFIELD (15)	TEXTRGN (15)	POPUP (16)
GET FROM (2)	GET (1)	GET (1)	GET (1)	GET/SAY (1)	GET (1)	GET (1)
varname	varname	varname	varname	varname	varname	varname
from expr				say expr		
vpos	vpos	vpos	vpos	vpos	vpos	vpos
hpos	hpos	hpos	hpos	hpos	hpos	hpos
height	height	height	height	height	height	height
width	width	width	width	width	width	width
type (0-4)						
	function	function	function	func/pict	picture	function
				type		
schemeno	schemeno	schemeno	schemeno	schemeno	schemeno	schemeno
				col pair		
expr/code				expr/code		expr/code
firstelem				lower lim		firstelem
expr/code				expr/code		expr/code
numelem				upper lim		numelem
expr/code	expr/code	expr/code	expr/code	expr/code	expr/code	expr/code
when	when	when	when	when	when	when
expr/code	expr/code	expr/code	expr/code	expr/code	expr/code	expr/code
valid	valid	valid	valid	valid	valid	valid
				expr/code		
				error		
expr/code	expr/code	expr/code	expr/code	expr/code	expr/code	expr/code
message	message	message	message	message	message	message
				refresh?		
disabled?	disabled?	disabled?	disabled?	disabled?	disabled?	disabled?
					scrollbar?	
					tab?	
init val		init val	init val	init val		init val
init numbr		init numbr	init numbr	init numbr		init numbr
	spacing	spacing				

.FRX Database (Reports)

Fields	Object Types and Field Data				
OBJTYPE	N	REPORT (1)	BANDINFO (9)	REPFIELD (8)	TEXT (5)
OBJCODE	N	version (0)	TYPE (0-8)	SAY (0)	SAY (0)
NAME	M	window			varname
EXPR	M		group expr	the text	say expr
VPOS	N			vpos	vpos
HPOS	N			hpos	hpos
HEIGHT	N	page len	height	height	height
WIDTH	N	page width		width	width
STYLE	M		style	style	
PICTURE	M			func/pict	
ORDER	M				
UNIQUE	L				
COMMENT	M				
ENVIRON	L	environ?			
BOXCHAR	C				
FILLCHAR	C				
TAG	M				
TAG2	M				
FLOAT	L			float?	float?
STRETCH	L			stretch?	
NOREPEAT	L			norepeat	
RESETRPT	N			reset repeat	
PAGEBREAK	L		pagebreak?		
RESETPAGE	L		reset page?		
SWAPHEADER	L		swap hdr?		
SWAPFOOTER	L		swap ftr?		
EJECTBEFOR	L	eject bef?			
EJECTAFTER	L	eject aft?			
PLAIN	L	plain?			
SUMMARY	L	summary?			
ADDALIAS	L	add alias?			
OFFSET	N	prt offset			
TOPMARGIN	N	top marg			
BOTMARGIN	N	bottom marg			
TOTALTYPE	N			total code	
RESETTOTAL	N			reset code	

Object Types and Field Data				
BOX (7)	WORKAREA (2)	INDEX (3)	RELATION	GROUP
BOX/BOXD...(3-6)	1-25	1-25	1-25	
	file.dbf	file.idx		
		index expr	rel expr	
vpos				obj number
hpos				obj count
height				
width				
style				
	index name			
		unique?		
box char				
fill char				
	alias	for expr	into alias	
	tag name	to alias	from alias	
float?				

.MNX Database (Menus)

Fields	Object Types and Field Data			
OBJTYPE	N	MENUSYSTEM (1)	SUBMENU (2)	ITEM (3)
OBJCODE	N	version		
NAME	M	pad name	menu name	
PROMPT	M			prompt
COMMAND	M			command
PROCTYPE	N			
PROCEDURE	M	global default	menu options	item
SETUPTYPE	N			
SETUP	M	procedure		
CLEANTYPE	N			
CLEANUP	M	procedure		
MARK	C	mark (global)	mark (menu)	mark (item)
KEYNAME	M			key name
KEYLABEL	M			key label
SKIPFOR	M			skip for
NAMECHANGE	L		changed name?	
NUMITEMS	N		number of items	
LEVELNAME	C	menu level name	menu level name	menu level name
ITEMNUM	C	item number	item number	item number
COMMENT	M			comment
LOCATION	N	location of menu		
SCHEME	N		scheme	

.LBX Database (Labels)

Fields	Type	Label Layout	Line Contents	Database Info	Index Info	Relations Info
OBJTYPE	N	LABEL(30)	(19)	WORKAREA(2)	INDEX(3)	RELATION
OBJCODE	N			1-25	1-25	1-25
NAME	M	remarks		data file name	index name	
EXPR	M		label contents		index expr	relational expr
STYLE	M		print style			
HEIGHT	N	label height				
WIDTH	N	label width				
LMARGIN	N	left margin				
NUMACROSS	N	number across				
SPACESBET	N	spaces between				
LINESBET	N	lines between				
ENVIRON	L	environment saved?				
ORDER	M			index name		
UNIQUE	L				unique?	
TAG	M			alias	for expr	into alias
TAG2	M			tag name	to alias	from alias
ADDALIAS	L	add alias?				

Database Structure (.DBF)

A database file is made up of a header record and data records. The header record defines the structure of the database and contains any other information related to the database. It starts at file position zero.

The data records[1] follow the header (in consecutive bytes) and contain the actual text of the fields. The length of a record (in bytes) is determined by summing the defined lengths of all fields. Numbers in this file are represented in reverse bytes.

Data File Header Record	
Bytes	**Description**
00	Type of data file: FoxBASE+/dBASE III PLUS, no memo — 0x03 FoxBASE+/dBASE III PLUS, with memo — 0x83 FoxPro/dBASE IV, no memo — 0x03 FoxPro with memo — 0xF5 dBASE IV with memo — 0x8B
01-03	Last update (YYMMDD)
04-07	Number of records in file
08-09	Position of first data record
10-11	Length of one data record (including delete flag)
12-31	Reserved
32-n	Field subrecords[2]
n + 1	Header record terminator (0x0D)

Field Subrecords[3]	
Bytes	**Description**
00-10	Field name (maximum of 10 characters — if less than 10 it is padded with null character (0x00))
11	Data Type: C – Character N – Numeric L – Logical M – Memo D – Date
12-15	Displacement of field in record
16	Length of field (in bytes)
17	Number of decimal places
18-32	Reserved

Notes to Data File Structure

[1]The data in the data file starts at the position indicated in bytes 08-09 of the header record. Data records begin with a delete flag byte. If this byte is an ASCII space (0x20) the record is not deleted; if the first byte is an asterisk (0x2A) the record is deleted. The data from the fields named in the field subrecords follows the delete flag.

[2]The number of fields determines the number of field subrecords There is one field subrecord for each field in the database.

[3]See the System Capacities table in this appendix for limitations on characters per record, maximum fields, etc.

Memo File Structure (.FPT)

Memo files contain one header record and any number of block structures. The header record contains a pointer to the next free block and the size of the block in bytes. The size is determined by the SET BLOCKSIZE command when the file is created. The header record starts at file position zero and occupies 512 bytes.

Following the header record are the blocks that contain a block header and the text of the memo. The database file contains block numbers that are used to reference the memo blocks. The position of the block in the memo file is determined by multiplying the block number by the block size (found in the memo file header record). All memo blocks start at even block boundary addresses. A memo block can occupy more than one consecutive block.

Memo Header Record	
Bytes	**Description**
00-03	Location of next free block[1]
04-05	Unused
06-07	Block size (bytes per block)[1]
08-511	Unused
Memo Block Header and Memo Text	
00-03	Block signature[1] (indicates type of data in block): a. 0 – picture (picture field type) b. 1 – text (memo field type)
04-07	Length[1] of memo (in bytes)
08-n	Memo text (n = length)

[1]Number represented in left-to-right order in hexadecimal.

Index File Structure (.IDX)

Index files contain one header record and one or many node records. The header record contains information about the root node, the current file size, the length of the key, index options and signature, and printable ASCII representations of the key[1] and FOR expressions. The header record starts at file position zero.

The remaining node records contain an attribute, number of keys present and pointers to nodes on the left and right (on the same level) of the current node. They also contain a group of characters encompassing the key value and either a pointer to a lower level node or an actual database record number. The size of each record that is ouput to file is 512 bytes.

An example of an ordered tree structure follows the tables.

Index Header Record	
Byte	**Description**
00-03	Pointer to root node
04-07	Pointer to free node list (-1 if not present)
08-11	Pointer to end of file (file size)
12-13	Length of key
14	Index options (any of the following numeric values or their sums): a. 1 – a unique index b. 8 – index has FOR clause
15	Index signature (for future use)
16-235	Key expression (uncompiled; up to 220 characters)[1,3]
236-455	FOR expression (uncompiled; up to 220 characters ending with null byte)
456-511	Unused

Index Node Record	
Byte	**Description**
00-01	Node attributes (any of the following numeric values or their sums): a. 0 – index node b. 1 – root node c. 2 – leaf node
02-03	Number of keys present (0, 1 or many)
04-07	Pointer to node directly to left of current node (on same level; -1 if not present)
08-11	Pointer to node directly to right of current node (on same level; -1 if not present)
12-511	Up to 500 characters containing the key value for the length of the key with a four-byte hexadecimal number (stored in normal left-to-right format): If the node is a leaf (attribute = 02 or 03) then the four bytes contain an actual database number in hexadecimal format — else the 4 bytes contain an intra-index pointer.[2] The key/four-byte hexadecimal number combinations will occur the number of times indicated in bytes 02-03.

[1]The type of the key is not stored in the index. It must be determined by the key expression.

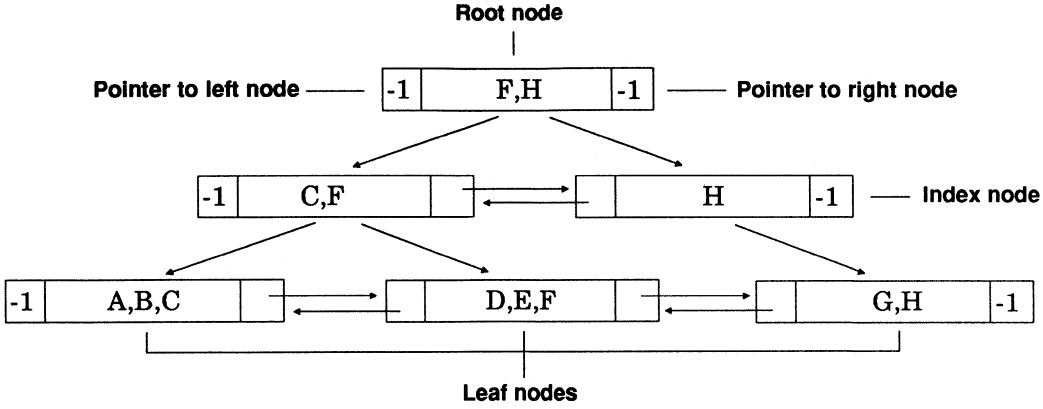
[2]Anything other than character strings, numbers used as key values, and the four-byte numbers in the leaf node are represented in reversed bytes (Intel 8086 format).

[3]Numbers are a special case when used as a key. They are converted through the following algorithm so they can be sorted using the same ASCII collating sequence as characters:

a. Convert the number to IEEE floating point format.
b. Swap the order of the bytes from Intel 8086 order to left-to-right order.
c. If the number was negative, take the logical complement of the number (swap all 64 bits, 1 to 0 and 0 to 1) else invert only the leftmost bit.

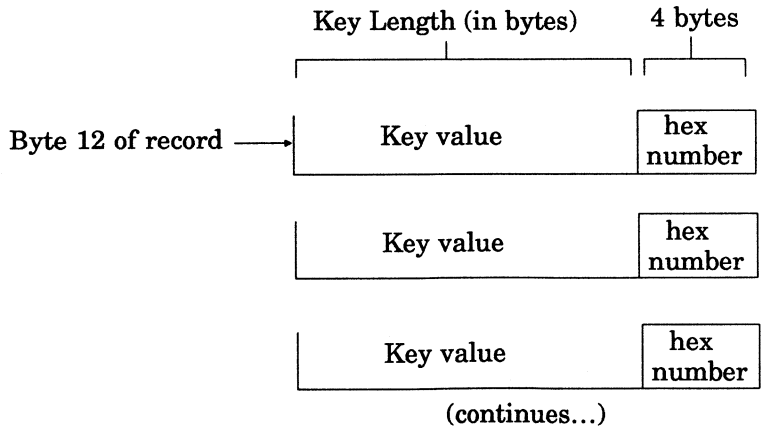
Example of an Ordered Tree Structure

Finding a key in the structure below requires searching a single path between the root and leaf nodes. Nodes at the lowest level are leaf nodes. Because the keys are sorted, all keys in the sub-tree are less than or equal to the parent node.



In the illustration above, the letters are used as the key values. Each key would also have a four-byte hexadecimal number. The numbers associated with the keys in the *leaf* nodes would be actual database numbers — all keys in other nodes would have intra-index pointers associated with them.

Bytes 12-511 in the index node records could be viewed as follows:



The key value/hexadecimal number combination occurs in bytes 12-511 n times where n is the number of keys present.

Compact Index File Structure (.IDX)

Compact Index Header Record	
Byte	**Description**
00-03	Pointer to root node
04-07	Pointer to free node list (-1 if not present)
08-11	Reserved for internal use
12-13	Length of key
14	Index options (any of the following numeric values or their sums): a. 1 – a unique index b. 8 – index has FOR clause c. 32 – compact index format d. 64 – compound index header
15	Index signature
16-19	Reserved for internal use
20-23	Reserved for internal use
24-27	Reserved for internal use
28-31	Reserved for internal use
32-35	Reserved for internal use
36-501	Reserved for internal use
502-503	Ascending or decending: a. 0=ascending b. 1=descending
504-505	Reserved for internal use
506-507	FOR expression pool length[1]
508-509	Reserved for internal use
510-511	Key expression pool length[1]
512-1023	Key expression pool (uncompiled)

[1]This information tracks the space used in the key expression pool.

Compact Index Interior Node Record	
Byte	**Description**
00-01	Node attributes (any of the following numeric values or their sums): a. 0 – index node b. 1 – root node c. 2 – leaf node
02-03	Number of keys present (0, 1 or many)
04-07	Pointer to node directly to left of current node (on same level; -1 if not present)
08-11	Pointer to node directly to right of current node (on same level; -1 if not present)
12-511	Up to 500 characters containing the key value for the length of the key with a four-byte hexadecimal number (stored in normal left-to-right format): If the node is a leaf (attribute = 02 or 03) then the four bytes contain an actual database number in hexadecimal format — else the 4 bytes contain an intra-index pointer.[2] The key/four-byte hexadecimal number combinations will occur the number of times indicated in bytes 02-03.

Compact Index Exterior Node Record	
Byte	**Description**
00-01	Node attributes (any of the following numeric values or their sums): a. 0 – index node b. 1 – root node c. 2 – leaf node
02-03	Number of keys present (0, 1 or many)
04-07	Pointer to node directly to left of current node (on same level; -1 if not present)
08-11	Pointer to node directly to right of current node (on same level; -1 if not present)
12-13	Available free space in node
14-17	Record number mask
18	Duplicate byte count mask
19	Trailing byte count mask
20	Number of bits used for record number
21	Number of bits used for duplicate count
22	Number of bits used for trail count
23	Number of bytes holding record number, duplicate count and trailing count
24-511	Index keys and information[2]

[2]Each entry consists of the record number, duplicate byte count and trailing byte count, all compacted. The key text is placed at the logical end of the node, working backwards, allowing for previous key entries.

Compound Index File Structure (.CDX)

All compound indexes are compact indexes.

One file structure exists to track all the tags in the .CDX file. This structure is identical to the compact index structure *with one exception* — the leaf nodes at the lowest level of this structure point to one of the tags in the compound index.

All tags in the index have their own complete structure that is identical to the compact index structure for an .IDX file.

FoxPro 1.x Report File Structure (.FRX)

NOTE: The .FRX file structure described here is for FoxPro 1.x .FRX files. The structure of FoxPro 2.0 .FRX files is documented earlier in this appendix.

All records in a report file consist of a fixed number of fields that are delimited by tabs (0x09). There is one record for each object type in the report. Each record terminates with a carriage return (0x0D) and a line feed character (0x0A). All information in the file is represented with ASCII characters.

	Object Type with Field Values				
Field #	**Screen (1)**	**Text (5)**	**Box (7)**	**Report Field (17)**	**Band Info (18)**
0[1]	Screen	Text	Box	Report Field	Band Info.
1[2]	Version[3]	SAY	Box/BoxD/ BoxC	SAY	Band Type[4]
2	Resource File Name	Actual Text	Null	Field Expr.	Group Expr.
3	Heading	Null	Null	Field Type[5]	Null
4	Vertical Position	Vertical Position	Vertical Position	Vertical Position	Page Break[6]
5	Horiz. Position	Horiz. Position	Horiz. Position	Horiz. Position	Reserved
6	Height	Height	Height	Height	Height
7	Width	Width (of actual text)	Width	Width	
8	Font Name	Null	Null	Null	Null
9	Printer Offset		0x00 or ?[7]		Swap Header[6]
10					Swap Footer[5,6]

Field #	Object Type with Field Values				
	Screen (1)	Text (5)	Box (7)	Report Field (17)	Band Info (18)
11					Null
12					Null
13					Null
14					Null
15					Null
16					Null
17	Null	Null	Null		Null
18	Null	Null		Once (0x30)/No-duplicate (0x31)	Null
19[8]	Null	Top/ Float	Top/ Float	Top/Stretch/ Float/Float and Stretch	Null
20	Null	Null	Null	Type Total[9]	Null
21	View File Info	Null	Null	Reset Total[10]	Null
22	Null	Null	Null	Null	Null
23	User Comment	User Comment	User Comment	User Comment	User Comment
24	User Data	User Data	User Data	User Data	User Data

NOTE: Shaded cells represent fields that are reserved for future use. When a field position is reserved, tab delimiters for that position are present but there are no characters in its placeholder. If a field is null, only a position holder is present. There are no characters between the tab delimiters.

Notes to Report File Structure

[1]Object types and their values in the file:

a. Screen = 0x31
b. Text = 0x35
c. Box = 0x37
e. Band Info = 0x31 and 0x38
d. Report Field = 0x31 and 0x37

[2]SAY/GET values:

a. Single-line box = 0x30 (Box)
b. Double-line box = 0x31 (BoxD)
c. GET (Input) = 0x32 (GET or BoxC)
   If the object type is Box, then 0x32 represents a character line box.
d. SAY (Output) = 0x34 (SAY)

[3]The current version for character-based reports is "1100".

[4]Band types and values when object type is Band Info:

a. Title = 0x30
b. Page Header = 0x31
c. Group Header = 0x33
d. Detail = 0x34
e. Group Footer = 0x35
f. Page Footer = 0x36
g. Summary = 0x37

[5]Field types:

N – Numeric
M – Memo
C – Character
D – Date
L – Logical

[6]When the object type is Band Info and the Band Type is Group Header, 0 = off and 1 = on. In all other cases, this field is null.

[7]Font size for a Box object type is 0x30 if it is either a single- or double-line box. If it is a character box, the binary value of the character used is shifted eight bits to the left and its value is placed in the field as base 10-ASCII characters.

[8]Object type and flag values:

a. Top = 0x30
b. Stretch = 0x31
c. Float = 0x33
d. Float and Stretch = 0x34

[9]Values for Totaling option (ASCII character of number in file):

a. No total = 0
b. Count = 1
c. Sum total = 2
d. Average = 3
e. Minimum = 4
f. Maximum = 5

[10]Values for Reset Total option (ASCII character of number in file):

a. End of report = 0
b. End of page = 1
c. End of column = 2
d. End of group = 3-22

FoxPro 1.x Label File Structure (.LBX)

NOTE: The .LBX file structure described here is for FoxPro 1.x .LBX files. The structure of FoxPro 2.0 .LBX files is documented earlier in this appendix.

Label files contain information for one label definition file as defined by the user. All bytes are stored in Intel 8086 format.

Byte	Description
00	Version: 03 = FoxPro label
01-60	Remarks (ASCII characters)
61-62	Height: Number of lines in label
63-64	Left Margin: Column number of the left margin
65-66	Width: Width of label
67-68	Number Across: Number of labels across row
69-70	Spaces Between: Number of spaces between labels
71-72	Lines Between: Number of lines between labels
73-74	Length of expression contents
75-n	Label Expression Contents: List of expressions in label separated by carriage returns (0x0D).

FoxBASE+ Memo File Structure (.DBT)

FoxBASE+ memo files don't have the versatility of FoxPro memo files. They can only contain ASCII text data.

Records output to this file, in blocks, are 512 bytes in size. The block at file position zero contains the block number of the first free file position. This block number is stored in the first two bytes in reverse order (Intel 8086 format). To find the address of the next free block, multiply the size of a single block (512 bytes) by the block number.

The blocks that follow the initial block information contain the text of the memos from the associated database. The memo field in the database file contains the number of the block in the memo file that contains the actual text. All memo blocks start on 512 byte boundary addresses.

FoxPro Macro File Format (.FKY)

File Header	
Byte	**Description**
01-03	Signature, Hex 79ff
04-15	Ignored
16-17	Number of macros (binary)
18-end	The macros
Individual Macros	
00-19	Macro name
20-21	Macro length (in keystrokes, binary)
22-23	Keystroke (two bytes, binary)
24-end	Macro keystrokes

Appendix C – 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 the error messages for FoxPro 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 on-line help under the topic ► Error Messages.

Alphabetical Listing of Error Messages

"<field>|<variable>" is not unique and must be qualified. (832)

The field or variable in the SQL SELECT command you have specified cannot be found.

["<name>"] must be a file variable. (1226)

A memory variable or an array variable was used where a file variable (field) is required.

["<name>"] must be a memory variable. (1225)

A file variable (field) was used where a memory or array variable was required.

["<name>"] must be an array. (1232)

A DIMENSION statement contains a variable declaration without the required subscript arguments.

**** or ^ domain error. (78)**

Exponentiation was attempted on a negative number.

ACOS( ): Out of Range. (293)

The expression you have used with the arccosine 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 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.

ASIN( ) : Out of Range. (291)

The expression you have used with the arcsine 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. (1631)

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 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.

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 build APP/EXE without a main program. (689)

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 ["<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 find screen/menu generation program. (693)

The GENSCRN or GENMENU programs that create menus and screens cannot be located.

Cannot GROUP by aggregate field. (846)

There has been an attempt to GROUP by one of the aggregate functions MIN( ), MAX( ), SUM( ), AVG( ), COUNT( ), NPV( ), STD( ) or VAR( ).

Cannot locate COMSPEC environment variable. (412)

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 DOS environment variables.

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 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/FORM allowed only with FROM clause. (1695)

You must include the COLUMN or FORM options when creating 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.

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 record is trashed. (1115)

The database header contains invalid information.

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).

Exclusive open of file is required. (110)

An attempt was made to perform an operation which requires exclusive use of the database.

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.

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.

File access denied. (1705)

An attempt was made to write to a file which is write protected by the 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 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.

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 value. (46)

An expression specified by SET MEMOWIDTH, BROWSE, or an @...TO command evaluated to an illegal value.

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.

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.

Insufficient memory. (43)

There was not enough memory for FoxPro to complete an operation.

Internal consistency error.

An internal FoxPro table has been corrupted. If this error occurs, call Fox Software Technical Support. No error number is returned and you are returned to 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 box dimensions. (277)

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 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 be imported has an invalid data type indicator.

Invalid DIF vector — DBF field mismatch. (116)

The DIF file you are attempting to be import has an internal conflict between its header and its data.

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 index must be between 1 and 100 bytes, inclusive for the Standard version and between 1 and 240 bytes, inclusive for the Extended version.

Invalid .LIB signature. (1190)

Library file used to contruct .EXE was 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 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 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.

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 or 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. (691)

FoxPro cannot use the library file you've specified.

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.

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. (607)

The maximum number of items that may be contained in a menu (128) was exceeded.

Maximum allowable menus (25) exceeded. (608)

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 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. (611)

A menu item or menu title was encountered which was defined with a datatype other than character.

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)

FoxPro Distribution Kit was not properly installed.

Must be a character, date or numeric key field. (1145)

The key field used to UPDATE ON must be character, date or numeric.

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 clause is present outside of the DO WHILE ENDDO, FOR ENDFOR or SCAN ENDSCAN commands.

No menu 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 ame. (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. (612)

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 an object file. (1309)

An attempt was made to load a compiled FoxPro program which does not have a proper header.

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. (56)

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.

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.

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 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.

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.

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. (337)

After encountering a PRINTJOB command, an ENDPRINTJOB command was not found before another PRINTJOB command was encountered.

Procedure not found. (1162)

The procedure specified with the IN clause of the DO command could not be located.

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.

PROMPTs for this popup have already been defined. (279)

An attempt has been made to use the BAR option to define the contents of the popup. The popup has already been defined with the PROMPT option.

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 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 DOS environment variable COMSPEC.

RUN/! command string too long. (411)

The command string included with the RUN command is too long. The maximum length allowed is approximately 240 characters.

Screen code too big for memory. (507)

The screen object you are trying to copy has a code snippet larger than 64K.

SELECT's are not UNION compatible. (851)

Data types or sizes of fields being projected in each SELECT are not identical.

SQL aggregate on non-numeric expression. (811)

There was an attempt to perform an average, sum, standard deviation or variance on a non-numeric field.

SQL cancelled operation. (839)

The user has pressed the Escape key.

SQL column "<field>|<variable>" not found. (806)

The field or variable you have specified cannot be found.

SQL could not locate database. (802)

The database specified could not be found.

SQL err building temporary index. (831)

A temporary index could not be built, possibly due to insufficient disk space.

SQL expression too complex. (845)

FoxPro ran out of memory when it tried to expand the SELECT statement to analyze it.

SQL illegal GROUP BY in subquery. (828)

When using one of the six operators +, -, *, /, < and >, there can only be one row of output.

SQL index not found. (830)

FoxPro could not find the existing index.

SQL internal Err <expN>. (800)

An internal error has occured. If this error occurs, call the Fox Software Technical Support.

SQL invalid * in SELECT. (820)

There was an attempt to use * in one of the aggregate functions of MAX( ), MIN( ), AVG( ), or SUM( ).

SQL invalid aggregate field. (822)

There has been an attempt to do aggregation on a memo field.

SQL invalid DISTINCT. (819)

There must only be one distinct used per level.

SQL invalid GROUP BY. (807)

There is an error in the GROUP BY clause.

SQL invalid HAVING statement. (805)

There is an error in HAVING clause.

SQL invalid ORDER BY. (808)

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 predicate. (836)

There is an invalid operator in the SQL statement.

SQL invalid SELECT statement. (804)

There is an error in the projection list.

SQL invalid SELECT. (826)

There has been an attempt to select more than one field in a subquery.

SQL invalid subquery. (825)

Something in the subquery is erroneous.

SQL invalid temporary file. (821)

The temporary file FoxPro created has become corrupted.

SQL invalid WHERE clause. (833)

There is an error in the WHERE clause.

SQL Multiple Row Subquery Result. (860)

Subqueries linked with the six normal operators +, - , *, /, < or > can only produce a one row result.

SQL no FROM clause. (818)

There must be a FROM clause in the SELECT command.

SQL open file failed. (823)

The file specified could not be opened.

SQL out of memory. (809)

FoxPro has run out of memory while trying to process your SELECT command.

SQL statement too long. (812)

The object code is too long to be compiled.

SQL subqueries nested too deep. (842)

FoxPro 2.0 does not support nested subqueries.

SQL too many columns referenced. (841)

There must be no more than 256 columns total referenced in any SELECT command.

SQRT domain error. (61)

The SQRT argument must not be negative.

Stack overflow — expression too complex. (1308)

In most cases, user-defined functions are nested too deep.

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.

Structural CDX file reference removed. (1107)

The database reference to an associated structural index file has been removed.

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.

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. (226)

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. (694)

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 READs in effect. (1249)

The maximum nesting level for READs (4) 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 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.

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.

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.

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 has not been defined. (214)

An attempt has been made to activate a window that has not been defined.

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
	Internal consistency error.
	Logical expression required.
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.
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).

Error Number	Text of Error Message
41	MEMO file is missing/invalid.
42	CONTINUE without LOCATE.
43	Insufficient memory.
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.
56	Not enough disk space.
58	LOG () : Zero or negative.
61	SQRT domain error.
62	Beyond string.
67	Expression evaluator fault.
78	** or ^ domain error.
91	File was not LOADed.
94	Wrong number of parameters.
95	Statement not allowed in interactive mode.
96	Nesting error.
101	Not suspended.
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.

Error Number	Text of Error Message
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 menu 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.
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 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.
226	Tab stops must be in ascending order.
231	Invalid SET expression.

Error Number	Text of Error Message
255	Not a valid RapidFile database.
256	Not a valid Framework II database/spreadsheet.
277	Invalid box dimensions.
279	PROMPTs for this popup have already been defined.
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.
337	Printjobs cannot be nested.
350	Field must be a Memo field.
355	Macro not defined.
356	Invalid keyboard macro file format.
392	Maximum record length exceeded in import file.
411	RUN/! command string too long.
412	Cannot locate COMSPEC environment variable.
507	Screen code too big for memory.
607	Maximum allowable menu items (128) exceeded.
608	Maximum allowable menus (25) exceeded.
611	Menu item/titles must be type CHARACTER.
612	No such menu/item is defined.
689	Cannot build APP/EXE without a main program.
691	Library file is invalid.
693	Cannot find screen/menu generation program.
694	Too many extensions specified.
800	SQL internal err <expN>.
802	SQL could not locate database.
804	SQL invalid SELECT statement.
805	SQL invalid HAVING statement.
806	SQL column "<field>|<variable>" not found.
807	SQL invalid GROUP BY.
808	SQL invalid ORDER BY.
809	SQL out of memory.

Error Number	Text of Error Message
811	SQL aggregate on non-numeric expression.
812	SQL statement too long.
818	SQL no FROM clause.
819	SQL invalid DISTINCT.
820	SQL invalid * in SELECT.
821	SQL invalid temporary file.
822	SQL invalid aggregate field.
823	SQL open file failed.
825	SQL invalid subquery.
826	SQL invalid SELECT.
828	SQL illegal GROUP BY in subquery.
830	SQL index not found.
831	SQL err building temporary index.
832	"<field>|<file>" is not unique and must be qualified.
833	SQL invalid WHERE clause.
836	SQL invalid predicate.
839	SQL cancelled operation.
841	SQL too many columns referenced.
841	SQL subqueries nested too deep.
845	SQL expression too complex.
846	Cannot GROUP by aggregate field.
851	SELECTs are not UNION compatible.
860	SQL multiple row subquery result.
1001	Feature not available.
1012	OS memory error.
1102	Cannot create file [<"file">].
1104	File read error.
1105	File write error.
1101	Cannot open file ["<file>"].
1107	Structural CDX file reference removed.
1108	Relational expression too long.
1112	File close error.
1113	File not open.
1115	Database record is trashed.

Error Number	Text of Error Message
1117	Wrong length key.
1124	Key too big.
1126	Record too long.
1127	For/while need logical expressions.
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.
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.
1156	Duplicate field names.
1157	Cannot update file.
1162	Procedure not found.
1163	Browse database closed.
1164	Browse structure changed.
1190	Invalid .LIB signature.
1191	Bad .LIB file.
1192	Internal .LIB undefined symbol error.
1193	Missing .RTT section.
1194	Link command failed.
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.

Error Number	Text of Error Message
1223	Invalid variable reference.
1225	"<name>" is not a memory variable.
1226	"<name>" is not a file variable.
1229	Too few arguments.
1230	Too many arguments.
1231	Missing operand.
1232	"<name>" is not an array.
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.
1252	Compiled code for this line too long.
1300	Missing)
1304	Missing (
1306	Missing ,
1307	Division by 0.
1308	Stack overflow – expression too complex.
1309	Not an object file.
1405	RUN/! command failed.
1410	Unable to create temporary work file(s).
1600	Not enough memory to USE database.
1604	No menu bar defined.
1605	No popup menu defined.
1621	No pads defined for this menu.
1631	Bad array dimensions.
1632	Invalid WINDOW file format.
1642	COLORSET resource not found.
1643	Printer driver is corrupted.
1644	Printer driver not found.
1645	Report nesting error.

Error Number	Text of Error Message
1646	Total field type must be numeric.
1647	Improper data type in field expression.
1649	No previous printjob to match this command.
1652	Attempt to use FoxPro function as an array.
1653	Label nesting error.
1659	Cannot convert Memo file for a read-only database file.
1695	COLUMN/FORM 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.
1707	Structural CDX file not found.
1708	File is open in another work area.
1903	String too long to fit.
1907	Bad drive specifier.
1908	Bad width or decimal place argument.
1999	Function not implemented.

Index

D

E

F

G

H

I

J

K

S

T

U

W

Contents

FoxPro Developer's Guide

Fox Software, Inc.
134 W. South Boundary
Perrysburg, OH 43551
Phone: 419-874-0162
Fax: 419-874-8678
Telex: 65030 40827 FOX

Fox Software International
Intech House, Cam Centre
Wilbury Way, Hitchin
Herts SG4 OAP
United Kingdom
Phone: 44-462-421-999
Fax: 44-462-421-318

Fox Software GmbH
Wendenstraße 4
2000 Hamburg 1
Federal Republic of Germany
Phone: 49-40-233-201
Fax: 49-40-232-566

Printed in U.S.A.