Table of Contents

µWelcome to FoxPro! 5
Using the Migration Kit 5

Section I: dBASE IV

Migrating dBASE applications: an overview 6

Steps to take 6
Using dBASE database, memo, and index files 6
Databases 6
Memo files 7
Indexes 7
Using dBASE queries 7
Running the Migration Tools 9
Converting screens, reports, and labels 10
The File Converter 10
Screen files 11
Converting FMTs and PRGs 11
Using dBASE reports 12
Labels 14
Addressing dBASE language compatibility issues 15
Overview 15
A note on dBASE IV Applications Generator applications 15
Major compatibility issues 15
Ways to fix possible problems 16
SET COMPATIBLE 16
Calling user-defined functions (UDFs) 17
Non-syntax issues 17
Parameter passing 17
Reading keystrokes 17
Hard-coded file extensions 18
Alphabetical list of potential dBASE IV issues 18

Section II: Clipper Summer '87

Migrating Clipper Summer '87 applications: an overview 57
Steps to take 57
A note on Clipper 5.x applications 57
Using Clipper database, memo, and index files 57
Databases 57
Memo files 57
Indexes 58
.NDX indexes 58
.NTX indexes 58
Choosing a database 59
Selecting an index type 59
Creating FoxPro screen files from .FMT and .PRG files 60
Reports and Labels 60
Visibility of functions and procedures 61
"Extracting" functions and procedures 61
Single procedure file or multiple .PRGs 62
Addressing Clipper language compatibility issues 63
Overview 63
Other compatibility issues 63
Function Calls 64
ACHOICE(), DBEDIT(), and MEMOEDIT() 64
Error handling 65
Error handling FoxPro-style 65
Simulating BEGIN SEQUENCE...[BREAK]...END in FoxPro 66
Third party libraries 68
Arrays 68
Color 68
Other compatibility issues 69
Windows Specific Problems 69
Printing 69
ASCII character set. 69

Binary functions 69

SET commands 69

Keystrokes 70

Alphabetical list of potential Clipper issues 70

Section III: Using the Program Analyzer

Using the Program Analyzer 103

What the Program Analyzer does 103

New Analysis 103

Processing files 104

Disk space requirements 104

Settings 104

Open Analysis 105

The Program Analyzer interface 105

Filtering and Sorting 106

Jumping to potential problem areas from the Program Analyzer 106

Using your own text editor 107

Help 107

Reports 108

Viewing the issues database 108

After addressing the issues in the Program Analyzer list 108

FoxPro Projects 109

Appendices 110

Appendix A: Effects of the SET COMPATIBLE command 110

Appendix B: dBASE file types and what to do with them 115

Appendix C: dBASE error numbers that represent different errors in FoxPro 117

Appendix D: Same errors with different numbers 120

Appendix E: Network and security libraries 121

Appendix F: Clipper 5.x incompatabilities 131

Appendix G: Key assignments 132

FoxPro ON KEY LABEL KEY assignments 132

INKEY() codes for FoxPro, dBASE and Clipper Summer '87 133

Appendix I: FoxPro Reserved Words 136

INDEX 140

Acknowledgments

Many thanks to the following people who contributed to this version of the FoxPro Migration Kit:

Menachem Bazian Flash Creative Management River Edge, NJ 201-489-4680	Ron Dennis HB Software & Services Redondo Beach, CA 310-372-4932	Miriam Liskin Berkeley, CA 510-644-3259	Marc Schnapp Primary Key Consulting El Sobrante, CA 510-223-1815
Randy Brown Sierra Systems Kensington, CA 510-526-0393	Russell Freeland Synergy Corp. Ft. Lauderdale, FL 305-792-1866	Dan Madoni Microsoft Redmond, WA	

Andrew Coupe	Bill French	Tina Newton
Microsoft	Global Technologies	Ph.Data
Washington, D.C.	Aurora, CO	New York, NY
	303-337-7758	212-473-1261

Welcome to FoxPro!

There are many reasons to move to Microsoft FoxPro: unsurpassed speed, better tools, reliability, and, because FoxPro 2.5 is available for both MS-DOS® and Microsoft Windows™, a powerful cross-platform capability. And finally, it preserves your current investment in Xbase knowledge, data, and applications while moving you forward to take advantage of today's technology.

This Migration Kit is designed to make your move to FoxPro as simple as possible. FoxPro 2.5 for MS-DOS and FoxPro 2.5 for Windows will run dBASE III Plus® applications *unchanged*. FoxPro is a full superset of dBASE III Plus commands and functions. dBASE IV applications might run in FoxPro unchanged as well. However, dBASE IV diverged from the dBASE III Plus standard. Applications that use dBASE IV-specific features might need to be modified. The same is true of Clipper Summer '87 applications, which is why we created this Migration Kit.

Together, the native capabilities of FoxPro and the Migration Kit allow you to migrate your files to FoxPro and enjoy the immediate benefits of FoxPro speed and reliability. And you'll soon discover the many benefits FoxPro provides such as better tools, full mouse support, and the ability to move between multiple design surfaces in a windowing environment.

Using the Migration Kit

The Migration Kit supports migration of dBASE IV (version 2.0 and earlier) and Clipper Summer '87 applications. Clipper 5.x differs greatly from FoxPro and is not supported. The Kit contains an application written in FoxPro 2.5 called the Migration Tools which converts many types of dBASE IV and Clipper Summer '87 files. The application also analyzes program files for code that might not run properly in FoxPro.

The Migration Tools run in both FoxPro 2.5 for MS-DOS and FoxPro 2.5 for Windows. The application will not run in FoxPro 2.0 or FoxPro 2.5 on platforms other than MS-DOS and Microsoft Windows.

The Migration Kit also includes this Migration Guide. The Guide explains the migration process, how to use the Migration Kit software, and how to modify your programs so they run correctly in FoxPro. The Guide is divided into three main sections:

I. Migrating dBASE applications

II. Migrating Clipper applications

III. Using the Program Analyzer

Whether you're migrating a dBASE or a Clipper application, you'll need to read through the "Using the Program Analyzer" section. You should first read the appropriate section (dBASE or Clipper) on migrating applications and other files.

Migrating dBASE applications: an overview

Your investment in dBASE consists of several different kinds of files: databases, queries, forms, reports, labels, and programs. All of these can be migrated to FoxPro.

FoxPro uses the same database file format as dBASE, so you can use your databases right away. You can run dBASE queries (.QBE files) in FoxPro after slight modification.

Using Migration Kit tools, you can quickly and easily convert dBASE screens, reports, and labels to their FoxPro equivalents. Format files (.FMTs) from dBASE work in FoxPro too, though some might need modifications in certain instances.

dBASE program files (.PRGs) may run unchanged in FoxPro but probably use at least a few commands and functions that depart from the dBASE III Plus standard. The Migration Kit's Program Analyzer helps identify these key words and suggests ways you can change your program so it runs correctly in FoxPro.

The time it takes to migrate an application will vary. In many cases it will take no time at all. In all cases, your patience will be greatly rewarded with faster program execution, greater reliability, and, if you have FoxPro 2.5 for Windows, the ability to turn your existing programs into true Windows-based applications.

Steps to take

The migration process boils down to the following steps:

1. First, create a backup copy of all your files. **Do not work on your original files!**
2. Bring databases into FoxPro.
3. Modify query files.
4. Using the File Converter, convert all screens, reports, and labels.
5. Use the Program Analyzer to find and address areas of potential incompatibility.
6. If used by your program, check to see if your .FMT files need to be modified.
7. Enjoy the speed and power of Microsoft FoxPro!

Not all your dBASE files are necessary to run an application in FoxPro, only databases, screens, reports, labels, and programs. Appendix B lists each type of dBASE file and how you should handle them in the migration process.

Using dBASE database, memo, and index files

DatabasesXE "Databases"§

FoxPro uses the same native file format (.DBFXE ".DBF"§) for databases as dBASE, so you can use your data right away, without any conversion. This is true of dBASE III PLUS databases as well. If files are encrypted, however, they must first be unencrypted before FoxPro can read them.

Simply type USE <database name> in the Command window or choose File...Open from the menus and you can begin browsing and analyzing your dBASE database.

Memo files

The XE "Memo files"§FoxPro file format for memo fields allows you to store an unlimited amount of data in a memo field. (You're limited by disk space, of course.) When you open a dBASE IV database that has an associated memo file in FoxPro, you are asked whether you want to convert the memo file. If you say yes, FoxPro converts the memo file to the FoxPro format (.FPT) and erases the original dBASE file (.DBTXE ".DBT"§). (Note that the database must be opened in exclusive mode in order to convert the files. In FoxPro, EXCLUSIVE is on by default.)

You must say yes, as FoxPro will not read a database with a dBASE memo field unless you allow FoxPro to convert the memo field to .FPT format.

After a memo field has been converted to FoxPro format, dBASE will not be able to read the .DBF file associated with the memo field. However, FoxPro can easily convert a .DBF and associated FoxPro memo field to a .DBF and .DBT memo field that dBASE IV can read. Use the command COPY TO <database name> TYPE FOXPLUS.

IndexesXE "Indexes"§

FoxPro uses a more efficient indexing scheme that results in better performance as well as index fields of one-half to one-third the size of dBASE indexes. FoxPro automatically converts .MDXXE ".MDX"§ index tags to create an equivalent structural .CDX index. The conversion leaves the dBASE indexes intact. The FoxPro structural .CDX is opened automatically just like an .MDX when a database is used.

FoxPro doesn't automatically convert .NDXXE ".NDX"§ indexes. However, a USE command that names .NDX indexes or a SET INDEX TO command that opens an .NDX index will cause FoxPro to convert the named indexes. .NDX indexes are converted to FoxPro .IDX indexes.

Using dBASE queriesXE "Queries"§

A dBASE query is a file with the .QBEXE ".QBE"§ extension that consists primarily of the dBASE IV commands required to execute the query. With minor changes, this file can be run in FoxPro as a program and return the same results as dBASE. However, you won't be able modify this query using the FoxPro query tool (Relational Query By Example, or RQBE). You would have to modify the commands in the .QBE file directly. Queries created in the FoxPro RQBE are stored as SQL commands, making them more succinct and faster to execute.

Before modifying a .QBE file, make a copy if you want to continue to run the query in dBASE IV.

To convert a .QBE into a program that can be run in FoxPro, take the following steps:

1. In FoxPro, choose File...Open from the menus.
2. Select Program as the type of file.
3. Select the Show all files check box.
4. Open a dBASE query (.QBE file).
5. Scroll down in the text editor to the first non-text or "garbage" character.
6. Delete all such characters.
7. Close the file, saving changes.

µ §

The highlighted "garbage" characters should be deleted.

Now you can run the query using the DO command. Be sure to include the .QBE file extension. dBASE IV queries issue a SET EXACT ON command but do not issue a SET EXACT OFF command at the end. When you close the database environment created by the query, return SET EXACT to its previous state.

Queries that open and link databases, specify a sort order, and specify selection criteria run without problem. Queries that compute summary statistics can contain several conditional tests and processing loops that might cause problems in FoxPro. You can recreate these queries very quickly in the FoxPro Relational Query by Example (RQBE) tool and be able to modify them later. Possible incompatibilities in .QBE files are:

- SET FIELDSXE "SET FIELDS"§ command
- TAGNOXE "TAGNO()"§ function
- NOSAVEXE "NOSAVE"§ option in USE command (version 1.5 only)
- SET CATALOGXE "SET CATALOG"§ TO
- SET("CATALOG")XE "SET(\"CATALOG\")"§ function

SET FIELDS behaves differently in FoxPro than in dBASE. For more information, see SET FIELDS in the section titled "Alphabetical list of potential dBASE IV issues."

Also, while FoxPro does not natively support the TAGNO function, this kit contains a user-defined function that provides the exact same functionality. As long as this UDF is available to the query program, the TAGNO function will not cause any problems. See the section titled "Calling user-defined functions (UDFs)."

FoxPro does not have a NOSAVE option in the USE command. To have the query delete files created during query execution, save the names of temporary files and delete them at the end of the query program, as in the example below:

```
temp_file=SYS(3)+".dbf"      &&Create unique filename with .DBF extension
<<execute code>>
delete file (temp_file)
```

Running the Migration Tools

The Migration Tools provided in this kit are comprised of the File Converter and Program Analyzer, both written in FoxPro. To install these tools, create a subdirectory named MIGRATE in your FoxPro directory. Copy all the files from the Migration Kit floppy disk to the MIGRATE subdirectory.

To run the Migration Tools, select the Do command from the Program menu. In the resulting file dialog box, change to the MIGRATE subdirectory and select MIGRATE.APP. Alternately, you can type DO MIGRATE.APP in the Command window. Make sure you set the default directory to the directory where MIGRATE.APP resides.

After you DO MIGRATE.APP, a new pad called Migration Tools is placed on the FoxPro menus.

µ §

The command DO MIGRATE.APP starts the Migration Tools application and adds a pad to the FoxPro menus.

All the Migration Tools, the File Converter and the Program Analyzer, can be accessed from the Migration Tools menu. While the Migration Tools are running, some other menus will be disabled. To remove the Migration Tools menu and reactivate these menus, choose Close Tools from the Migration Tools menu.

Converting screens, reports, and labels

The File ConverterXE "File Converter"§

The process for converting screens (.SCRXE ".SCR"§s), format files (as well as PRGs with @SAYs and GETs), reports (.FRMXE ".FRM"§s) and labels (.LBLXE ".LBL"§s) is the same for each type of file. From the Migration Tools menu, choose Convert Files. This brings up a dialog box that displays all the files in the current directory with .SCR, .FMT, .PRG, .FRM, and .LBL extensions. Files with the .NTX extension will also be displayed. These are Clipper index files and which do not concern us here.

µ §

Select the screens, reports, and labels to be converted from the File Converter dialog.

To select a file for conversion, either double-click on the filename, or highlight it and then press ENTER. Selected files will have an asterisk placed next to the filename on the left (or a check mark in FoxPro 2.5 for MS-DOS). You can select all the files in a directory for processing by clicking the Select All button, or you can start over by clicking the Clear All button. To cancel selection of a single file, double-click on the filename, or highlight it and press the ENTER key. The File Converter allows you to select a mix of .SCRs, .FRMs, and .LBLs for processing at the same time.

To convert files in another directory, click the Directory button and move to a new directory. To convert files in multiple directories, first select and convert the files in one directory, and then select and convert files in another directory.

When you have finished selecting files, click the Process button. The FoxPro File Converter goes to work. Screens and files with @SAYs and GETs (.SCRs, .FMTs, and .PRGs) are converted to the FoxPro .SCX format. Report files (.FRMs) are converted to FoxPro .FRX format, and labels (.LBLs) are converted to .LBX format. These new files are saved in the same directory as the original files (which are left intact).

To modify a new FoxPro screen, report, or label, simply choose File...Open from the menus. Select the file type and the file you wish to edit. FoxPro will then launch the appropriate tool. For more information about the FoxPro Power Tools, see the *User's Guide* included in the FoxPro 2.5 documentation.

ScreenXE "Screens"§ files

Converting SCRs

The File ConverterXE "File Converter"§ preserves @SAYS and GETS, PICTURE clauses, WHEN and VALID clauses, whether a field is editable or not editable, ERROR messages, PROMPT messages, RANGE, default values, multiple-choice lists, as well as boxes and lines. Individual field colors and other style attributes are not preserved, though all other color attributes are retained.

Screen files are written in FoxPro 2.5 for MS-DOS format. If the file is opened for modification in FoxPro for Windows, FoxPro will ask whether you want to transport the screen to Windows. In most cases, you will want to say yes so the screen looks like a Windows-based screen and not an MS-DOS-based screen. Refer to the *Developer's Guide* in the FoxPro 2.5 documentation for more information about FoxPro's powerful cross-platform capabilities.

Converting multiple-page SCRs

The File Converter will convert multiple-page dBASE screens. Multiple-page screens should be converted into separate screens. Each screen should be saved separately as part of a screen set. Pressing page down in one screen of a screen set takes the user to the next screen in the set. To create separate screens, begin with the converted file. Delete everything except for one page of the screen and save it as a new file. Repeat this process with the other pages of the screen.

Converting FMTs and PRGs

To take full advantage of FoxPro and its windowing environment, it is best to convert format files and PRGs with @SAYs and GETs to FoxPro screen files. The file converter will take program and format files and create SCX files. When converting @SAY commands, the picture, function, range, valid, when, color, and message clauses are preserved.

When the converter processes an FMT or PRG file, the following dialog appears:

µ §

The dialog says which file is being created (INVOICE.SCX in this case). The file name is the same as the original file with an .SCX extension. The dialog gives you the opportunity to associate a database with the SCX file that's going to be created. If you associate a database with the SCX file, FoxPro will automatically open and close that database for you when the screen is run or opened for modification--a great convenience.

To associate a database with an .SCX, click the Choose... button. This brings up a file dialog that allows you to select any database on your local hard drive or the network. Then click the OK button and the new file will be created. Note the OK button is disabled until a valid database file has been chosen. You may, however, click the Cancel button or press ESC to avoid converting a particular file.

If you don't want to associate a database with the new screen, click the None button and the new file will be created without any association.

As with SCRs, FMTs and PRGs are converted to FoxPro 2.5 for DOS files. If you open these files in FoxPro Windows you will be asked if you wish to transport the converted files. Ordinarily, you will want to press the "Transport and Open" button.

To convert multiple-screen format files, you need to break the file into multiple files because the converter will stop after it encounters the first READ statement.

Modifying format filesXE "Format files"§XE "format files"§ which you choose not to convert

FoxPro supports dBASE format files (.FMTXE ".FMT"§s) and related commands such as SET FORMAT. Some dBASE IV format files have setup and cleanup code. Format files with key words other than @SAYs and @GETs won't work in FoxPro. If you choose not to convert your format files to FoxPro screens, the setup and cleanup code needs to be moved outside the format file. For example, setup code could be placed before the SET FORMAT command, and the cleanup code after it.

Using dBASE reportsXE "Reports"§

There are two possible approaches to using dBASE reports in FoxPro. You can run an .FRG file or you can convert an .FRMXE ".FRM"§ file. If you will only be using FoxPro 2.5 for MS-DOS and you're not planning to modify the report, running the .FRG file is often the best idea.

If you have FoxPro 2.5 for Windows or you plan to alter the report, converting the .FRM is the recommended action.

Effects of the conversion process

All bands, fields, calculated fields, hidden fields, picture templates, and functions for fields are converted. So are style attributes such as underline, bold, italic, and colors. Fonts are not converted. However FoxPro for Windows supports all Windows-based fonts including True Type fonts, giving you even greater control over this area than before. FoxPro for MS-DOS fonts can be changed using report variables.

Word-wrap bands are converted to a series of one-line text fields in FoxPro reports. Band spacing and pitch information is not converted. FoxPro does not use a global ruler or a word-wrap paragraph ruler, so this information is not retained. Converted reports will have a right margin equal to the width of the report. The left margin is not converted but is settable in the FoxPro Report Writer.

Reports with tab characters may need to have their fields rearranged to achieve the desired appearance.

In dBASE you can print the page header in the report introduction. In FoxPro, converted reports of this type have the title band after the page header, and the summary band before the page footer. However, when printed, the title header will precede the page header and the page footer will come before the summary footer.

Empty report bands print in dBASE, so the File Converter adds null characters to these bands so they will also print in FoxPro.

dBASE reports do not store complete environment information, only database aliases. As a result, upon opening a converted report, you may get an error such as "Customer.dbf not found." FoxPro reports can save database names, relations, skips, and index information. Once in the Report Writer, set up the environment the way you want it and then save that information with that report. No more such errors will appear.

Note that vertical stretch bands are only supported in the detail band in FoxPro. If you're report relies on vertical stretching outside the detail band, you may wish to run the .FRG file directly.

Running report programs (.FRGXE ".FRG"§s)

If you're not planning to modify a report, simply running the .FRG file in FoxPro often makes the most sense. To run an .FRG in FoxPro 2.5 for MS-DOS, use the REPORT FORM command, for example, type REPORT FORM myreport.frg. FoxPro compiles the .FRG into an .F4X file and runs the report. In FoxPro for Windows, use the DO command: DO myreport.frg. The report is compiled into an .FXP file and then run.

When running the reports with a DO command, remember that the optional information supplied in the REPORT FORM command can be passed to the .FRG program as parameters. Label programs do not require (or accept) these parameters. The first three parameters are logical variables, which are .T. if the equivalent REPORT FORM command would contain the NOEJECT, PLAIN, or SUMMARY key word, respectively. The fourth is a character string that stores an additional title line, and the fifth is a character string not used by dBASE IV (which you can use for your own purposes within the report or in UDFs called from the report).

For example, the following are equivalent:

REPORT FORM Namelist TO PRINT NOEJECT SUMMARY HEADING "Total Receipts"

DO Namelist.frg WITH .T., .F., .T., "Total Receipts"

Also note that the REPORT FORM and LABEL FORM commands trigger a search equivalent to a LOCATE for records that satisfy any FOR clause or filter condition. Because there is a CONTINUE command in the main DO WHILE loop generated by dBASE, you *must* execute a LOCATE command before running a report or label program with a DO command.

Printer drivers

XE "Printer drivers"§If you encounter problems running dBASE report files (.FRGs), most likely the problems will involve printer drivers. In dBASE IV, a printer driver includes all the information needed to print on a particular printer and produce the standard styles--boldface, italic, underlined, superscript, and subscript--as well as five custom fonts defined in the CONFIG.DB file.

In dBASE IV, if an application requires more than five special fonts, it may define more than one printer driver for the same printer. Many aspects of a particular report run or report section are governed by assigning appropriate values to system memory variables, while using the same printer driver. The name of the current printer driver is stored in the system memory variable _PDRIVER. You can use a *print form* to store sets of values for the relevant system memory variables (_PADVANCE, _PAGENO, _PBPAGE, _PCOPIES, _PDRIVER, _PECODE, _PEJECT, _PEPAGE, _PLENGTH, _PLOFFSET, _PPITCH, _PQUALITY, _PSCODE, _PSPACING, and _PWAIT).

In FoxPro for MS-DOS, you use a *printer driver setup* to store information about the specific printer and some of the same aspects of the print job found in a dBASE IV print form. The _PDRIVER variable stores the name of the current printer control program, and the _PDSETUP variable stores the name of the current printer driver setup.

References to _PDRIVER must be removed from a dBASE IV program before you can run it in FoxPro, which will interpret the name of the dBASE IV printer driver as the name of a new printer interface control program. In most cases, you can substitute a command that stores an appropriate value to _PDSETUP to identify your printer and overall print parameters.

Once you assign the right printer driver setup, FoxPro will produce the five standard attributes. Some additional changes may be necessary to print the report exactly as it appeared in dBASE IV.

In FoxPro for Windows, printer control is handled by Windows, and reports are routed to the printer selected using the Print Setup option on the File menu. FoxPro for Windows ignores all references to _PDRIVER and _PDSETUP, so commands that assign new values to _PDRIVER will not generate errors. However, formatting changes implemented in dBASE IV by switching printer drivers--such as choosing landscape or portrait orientation for a report--will also be ignored.

Like the standard attributes, the special fonts are implemented through the STYLE clause in the ? and ?? commands that construct the report. Because FoxPro for MS-DOS ignores any unrecognized codes in the STYLE clause, the dBASE IV fonts will not generate errors but will not appear in the output. One way to reproduce any special fonts defined in the dBASE IV configuration file and referenced in report or label forms is to modify the printer driver program. However, this is much harder than adding custom code to the report or label program. The easiest way to implement these fonts is to store the escape sequences that initiate and terminate them (which you can find in the dBASE IV CONFIG.DB file) in memory variables and add ??? commands to the report or label program to send these codes to the printer before and after printing the affected data. To find all the style codes, you can edit the .FRG or .LBG program and search for the STYLE clause.

In FoxPro for Windows, you control the font and style of printed output by adding a FONT clause to a ? or ?? command. FoxPro for Windows ignores all STYLE clauses unless you also specify a font using the new FONT clause. If you only use a few simple attributes, you might simply use a global search and replace to add the necessary FONT clauses. You can also substitute FONT clauses to emulate the special fonts implemented in dBASE IV with custom font codes defined in the CONFIG.DB file. Some experimentation with fonts will be required in most cases to achieve correct pagination, because FoxPro for Windows respects the page length *in lines* established through the system memory variable _PLENGTH. However, unlike FoxPro for MS-DOS, it does not necessarily print six lines per inch--in Windows, changing the font size affects both the height and width of the characters. A report designed to print 66 lines per 11-inch page will usually print more than one "page" on each physical page.

Some of the settings established in dBASE IV through system memory variables--in particular, the print pitch (_PPITCH) and quality (_PQUALITY), also require different printer driver setups in FoxPro. If the dBASE IV program establishes these settings once for an entire report, you can define a FoxPro printer driver setup with the desired combination of settings. If the settings are changed within a report program, you must edit the program to switch printer driver setups or add ??? commands to initiate and terminate the print modes at the appropriate times. FoxPro also ignores the system memory variable _PWAIT, which determines whether to pause for a paper change between pages, and you need to write a UDF, which you can call from the report header, to accomplish the pause.

Labels

XE "Labels"§As with reports, you have the choice of running label programs (.LBGs) in FoxPro or converting the label file (.LBLXE ".LBL"§) to FoxPro format (.LBX) using the File Converter.

Effects of the conversion process

All fields and text are converted. Where multiple fields are placed on one line, an expression that concatenates these fields replaces them. Note that if you use the centering feature in dBASE, it centers the field by placing spaces to the left of the object.

The File Converter treats this as a text string and places quotation marks around it. Also, note that style attributes (underlining, bold, italics, superscript and subscript) are not preserved by the File Converter and will need to be recreated in FoxPro.

Running label programs (.LBGXE ".LBG"§s)

The same considerations for running .FRGs apply to running .LBG files. In general, these should run without problem. Importantly, programs generated by dBASE IV version 1.5 make heavy use of the ISBLANK function to suppress blank lines that result from empty fields. You can solve this problem by changing all the occurrences of this function to the roughly equivalent EMPTY function.

Addressing dBASE language compatibility issues

Overview

Of the many hundred commands and functions in dBASE IV, overwhelmingly, they work exactly the same way in FoxPro. The degree of compatibility is extremely high, even more so with dBASE IV 1.0 and 1.1. However, there are areas where dBASE and FoxPro differ.

To migrate program files to FoxPro, first use the Program Analyzer to find potential compatibility problems. See the section titled "Using the Program Analyzer." After the Program Analyzer has created a database of potential issues, you can turn to eliminating each in turn. After you have addressed these areas, you can then try running your application.

Each issue the Program Analyzer finds is documented in the section titled "Alphabetical list of potential dBASE IV issues." Each issue is described and an explanation offered on how programs can be modified so they perform the way you expect them to in FoxPro.

A note on dBASE IV Applications GeneratorXE "Applications Generator"§ applications

Applications generated by the dBASE IV Applications Generator consist of a variety of files that store the various application components. There is one file for each bar menu, menu popup, files list, batch process, and so on, each of which has a unique extension (.BAR, .POP, .FIL, .VAL, and so on). These files are required only to modify the application in dBASE IV. They do not need to be converted.

You should erase or separate from your other files the main application object, which has the extension .APP, otherwise FoxPro will attempt to run it instead of the .PRG file of the same name (unless you type the .PRG extension). The .APP extension in FoxPro is used for files similar to .DBO files in dBASE.

Applications Generator applications consist of two .PRG files: a main startup program, which starts up the application and establishes the working environment, and a larger program with the same name as the main menu, which contains the individual procedures that execute the menu options. The main startup program has the same name as the main application object defined using the Applications Generator. To run the application in FoxPro, use a DO command to execute the startup program.

Major compatibility issues

Although the Analyzer might find many potential problems in your application, most all can be resolved, often by changing a single line of code, so that your application will run in FoxPro. Also, you might find that few *types* of changes are required, although you might find many occurrences of the same few issues.

The only significant areas of incompatibility are security, network functions, and the more rarely used SQL and transaction processing functions. Most SQLXE " SQL"§ programs (.PRSXE ".PRS"§ files) will need to be rewritten in FoxPro. Also, FoxPro doesn't offer native support for security or transaction processing. If you have FoxPro

2.5 for MS-DOS and Novell Netware® 2.x or higher, you can take advantage of Novell Transaction Tracking Services®.

In addition, many software vendors offer extensive libraries of security and network functions, providing an even richer alternative to the set of functions supported in dBASE. See Appendix E for a list of products and the many functions each supports.

Ways to fix possible problems

After the Program Analyzer has identified potential issues in your programs, you can address these issues in any of six ways:

- Substitute an equivalent FoxPro command or function
- Modify syntax
- SET COMPATIBLE DB4 ON
- Call UDFs
- Write a procedure or function
- Use a third-party solution

The Program Analyzer and this document include a list of all the known compatibility issues. (See the section below titled "Alphabetical list of compatibility issues.") Each issue has a description along with what changes, if any, should be made.

Many if not most problems can be addressed by using an equivalent FoxPro command or function. Often, user-defined functions can be created that effectively perform this substitution for you, and many such UDFs are included in this kit. In some cases, code needs only an additional argument or other small modification in order to work as expected. A number of minor issues can be resolved by setting the dBASE IV compatibility mode on.

Where quick fixes aren't readily available, it's usually just a matter of adding a procedure or few lines of additional code. Rarely will you need to resort to third-party products, except possibly in the cases of security and transaction processing.

SET COMPATIBLE

You can maximize compatibility by adding a SET COMPATIBLE command to your main startup program, as follows:

SET COMPATIBLE DB4 ON

This command changes the way FoxPro executes certain commands and functions to match the behavior of dBASE. The Program Analyzer will find all key words affected by SET COMPATIBLE. The section titled "Alphabetical list of potential dBASE IV issues" explains the effects SET COMPATIBLE has on a particular key word.

Setting COMPATIBLE on is a great way to get your application up and running. Eventually, as you add FoxPro features, you might find it easier to move away from using the compatibility feature.

Appendix A has a key word-by-key word list of the effects of SET COMPATIBLE.

Calling user-defined functions (UDFs)

You can solve many problems that arise from different function names and dBASE IV functions that are unsupported in FoxPro by using UDFs that duplicate the behavior of dBASE IV functions.

For example, the dBASE IV PCOUNT function is identical to the FoxPro PARAMETERS function. By using the following code, you could avoid errors resulting from the use of the PCOUNT function in a dBASE IV procedure or UDF:

```
FUNCTION Pcount

RETURN PARAMETERS()
```

The Migration Kit disk includes this UDF and others in a file called FOXPROC.PRG. These are all the dBASE functions for which the Migration Kit provides a UDF:

FDATE()	ISBLANK()
FLDCOUNT()	PCOUNT()
FOR()	TAGCOUNT( )
FTIME()	TAGNO()
	WINDOW()

By adding the command SET PROCEDURE TO FOXPROC.PRG to the beginning of your application, all these functions will work exactly as they do in dBASE. If your application already has a procedure file, add these UDFs to that file.

Where FoxPro has a built-in function with the same name as dBASE but that operates slightly differently, another approach must be taken. A built-in function is always executed even if a UDF with the same name is available.

Non-syntax issues

Parameter passingXE "Parameter passing"§

In dBASE IV, parameters are passed by reference to both procedures and functions. In FoxPro, parameters are passed by reference to procedures and by *value* to functions.   To pass parameters by reference to functions, add the command SET UDFPARMS TO REFERENCE to your program.

Reading keystrokes

If a dBASE IV application uses the INKEY(), READKEY() XE "READKEY "§or LASTKEY() XE "LASTKEY "§functions to check the user's last action and respond appropriately, errors might result in FoxPro. FoxPro and dBASE have different key assignments for a number of key combinations. INKEY(), READKEY() and LASTKEY() are flagged by the Program Analyzer for this reason.

FoxPro reads key assignments from a macro file with the extension .FKY. Use the dBASE.FKY macro file, included on the Migration Kit disk, so FoxPro keystrokes are mapped to dBASE keystrokes. Then include the command RESTORE MACROS FROM dBASE.FKY in your program. Alternately, you could change your program code so it uses FoxPro's "native" key assignments. See the table in Appendix G for a list of these values.

FoxPro key labels differ slightly from dBASE. See the description of the ON KEY command in the section "Alphabetical list of potential dBASE IV issues."

Hard-coded file extensionsXE "File extensions"§

The Program Analyzer finds the most commonly used file extensions types not used in FoxPro. The following table shows FoxPro equivalent extensions for each type of dBASE file that a program might use.

dBASE File	FoxPro File
XE ".DBF "§.DBF	No change needed
XE ".DBO"§.DBO	.FXP
XE ".DBT"§.DBT	.FPT
XE ".FMO"§.FMO	.PRX
XE ".FMT "§.FMT	.FMT
XE ".FRG "§.FRG	No change needed
XE ".FRM "§.FRM	.FRX
XE ".FRO "§.FRO	.F4X
XE ".LBG "§.LBG	No change needed

XE ".LBL "§.LBL	.LBX
XE ".LBO "§.LBO	.L4X
.MDX	.CDX
XE ".NDX "§.NDX	.IDX
XE ".PRG"§.PRG	.PRG
XE ".PRS "§.PRS	.PRG
XE ".QBE "§.QBE	.PRG

Alphabetical list of potential dBASE IV issues

Below is a list of the known compatibility issues in running dBASE programs in FoxPro. Each issue describes the behavior of a function or command in dBASE and in FoxPro. The Action section tells you what you should do to your program, and an example is usually included. Any other relevant information is placed in the comment section.

This information is also displayed in the Program Analyzer, with the exception of examples which are not displayed in the Program Analyzer.

For the sake of brevity and conciseness, no attempt is made to reproduce the documentation on these commands and functions. You can consult the dBASE IV and FoxPro documentation on specific commands and functions for further details.

Each issue is assigned one of four levels:

- **Level XE "Level "§1** commands and functions are those not supported in FoxPro. These commands and functions must be removed or replaced with FoxPro equivalents.
- **Level 2** commands and functions will generate errors in FoxPro but often have very close FoxPro equivalents.
- **Level 3** commands and functions will not generate errors but behave differently enough to merit attention.
- **Level 4** commands and functions will not generate errors and will rarely cause a problem in your program. Searching for Level 4 issues flags many lines of code which will usually work fine.

Level assignments, while somewhat arbitrary, are designed to give you a sense of the importance of an issue and the effort required to address it.

@...GET MESSAGE
See MESSAGE.

@...GET XE "@...GET"§REQUIREDXE "@...GET REQUIRED"§XE "REQUIRED"§

Level: 1

dBASE IV behavior: When used with RANGE or VALID, the REQUIRED key word forces validation whether or not data has changed, even if you skip a field with the mouse.

FoxPro behavior: Generates an error.

Action: To require validation when data does not change but the user moves through the field, remove the REQUIRED key word and SET COMPATIBLE DB4 on.

To require validation in cases where the user skips a field with the mouse, remove the REQUIRED key word. Then add a VALID clause to the READ that calls a UDF. Place all the tests for each GET in the UDF called by READ VALID.

Comment: The mouse can also be turned off using the SET MOUSE command.

Example: **dBASE**

```
@ 2,2 SAY "Name:" GET custname VALID REQUIRED
my_UDF()
@ 6,2 SAY "Amount:" GET order_amt VALID REQUIRED ;
order_amt <> 0
READ

FUNCTION my_udf
IF LEN(custname) = 0
    <give user error message>
    RETURN .F.
ELSE
    RETURN .T.
ENDIF
```

FoxPro

```
@ 2,2 SAY "Customer name:"  GET custname
@ 6,2 SAY "Amount: " GET order_amt
READ VALID my_udf()

FUNCTION my_udf
ret_val=.T.
IF LEN(custname) = 0
    <give user error message>
```

```
ret_val = .F.
ENDIF
IF order_amt = 0
     <give user different error message>
ret_val = .F.
ENDIF
RETURN ret_val
```

@...GETXE "@...GET"§ XE "@...GET"§OPEN WINDOWXE "OPEN WINDOW"§

XE "@...GET OPEN WINDOW"§Level: 3

dBASE IV behavior:	dBASE opens a window, displays the memo text, and places the cursor in the predefined window when you move into the field. You must press Ctrl+Home to begin editing.
FoxPro behavior:	Same as dBASE, except FoxPro displays a memo marker behind the predefined window and places the cursor in this memo marker.
Comment:	If desired, you can change the coordinates of the window that is opened so that it does not cover the memo marker. Alternatively, you could use the EDIT or MODIFY MEMO commands in FoxPro, which gives you tremendous flexibility and control.
Action:	None required.

@...SAY

XE "@...SAY"§Level: Not flagged by the Program Analyzer.

dBASE IV behavior:	a) Output that extends beyond the lower right corner of the screen will be displayed, causing the screen to scroll upward.
	b) With SET STATUS 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.
	c) @SAY with the PICTURE key word rounds the rightmost digit.
FoxPro behavior:	a) Output that extends beyond the end of the screen is truncated.
	b) Output cannot overwrite the status bar. Text that extends beyond the end of the screen is truncated.
	c) When data is displayed using a PICTURE clause, the value is truncated, not rounded
Action:	SET COMPATIBLE DB4 ON.
Comment:	With COMPATIBLE set on, @SAY in FoxPro acts like @SAY in dBASE. These issues are mostly cosmetic so, the Program Analyzer does not flag the @SAY command.

ACCESS()

See Security.

ACTIVATE MENUXE "ACTIVATE MENU"§

Level:	2
dBASE IV behavior:	dBASE does not allow the user to click on disabled menu pads.
FoxPro behavior:	FoxPro allows the user to click on disabled menu pads. The effect is the same as if the user had pressed the escape key.
Action:	Place the ACTIVATE MENU command inside a DO...WHILE loop.
Comment:	The loop should run until the user clicks on the menu pad to exit the application.

Example: **dBASE**

```
ACTIVATE MENU my_menu
```

FoxPro

```
DO WHILE PAD() <>"Exit"          &&Use the exit
ACTIVATE MENU my_menu            &&string from your
ENDDO                            &&application
```

ACTIVATE POPUPXE "ACTIVATE POPUP"§

Level:	2
dBASE IV behavior:	dBASE does not allow the user to click on disabled popup bars.
FoxPro behavior:	FoxPro allows the user to click on disabled popup bars. The effect is the same as if the user had pressed the escape key.
Action:	Place the ACTIVATE POPUP command inside a DO...WHILE loop.

Example: **dBASE**

```
ACTIVATE POPUP my_popup
```

FoxPro

```
DO WHILE BAR() <> 5              &&Use the bar
ACTIVATE MENU my_menu            &&number from your
END DO                           &&application
```

ACTIVATE SCREENXE "ACTIVATE SCREEN"§, ACTIVATE WINDOWXE "ACTIVATE WINDOW"§

Level:	Not flagged by the Program Analyzer.
dBASE IV behavior:	By default, the cursor position is set to 0,0.
FoxPro behavior:	The cursor position retains the value it had prior to the ACTIVATE SCREEN or ACTIVATE WINDOW command.
Action:	SET COMPATIBLE DB4 ON.
Comment:	The cursor position will be set to 0,0 after issuing the ACTIVATE SCREEN or ACTIVATE WINDOW command. This issue is mostly cosmetic, so the Program Analyzer does not flag these

commands.

APPEND MEMOXE "APPEND MEMO"§

Level:	3
dBASE IV behavior:	The file extension .TXT is assumed if none is specified.
FoxPro behavior:	If no file extension is specified, an error is generated.
Action:	SET COMPATIBLE DB4 ON.

BARCOUNT()XE "BARCOUNT()"§ (version 2.0 only)

Level:	2
dBASE IV behavior:	Returns the number of bars in a specified popup or in the active popup if no popup name is given.
FoxPro behavior:	Generates an error.
Action:	Replace BARCOUNT() with the FoxPro function CNTBAR().
Comment:	CNTBAR() in FoxPro requires that you specify the name of the popup.

Example: **dBASE**

```
num_bars=BARCOUNT(pop_name)
```

FoxPro

```
num_bars=CNTBAR(pop_name)
```

BARPROMPT()XE "BARPROMPT()"§ (version 2.0 only)

Level:	2
dBASE IV behavior:	Returns the prompt text of the specified bar in a specified popup, or in the current popup if none is given.
FoxPro behavior:	Generates an error.
Action:	Replace BARPROMPT() with the FoxPro function PRMBAR().
Comment:	PRMBAR() in FoxPro requires that you specify the name of the popup.

Example: **dBASE**

```
prmpt_txt=BARPROMPT(1)
```

FoxPro

```
prmpt_txt=PRMBAR(pop_name,1)
```

BEGIN TRANSACTION

See Transaction processing.

BLANK (version 1.5 only)
XE "BLANK (version 1.5 only)"§Level: 2

dBASE IV behavior: Used to place spaces ("nulls") in all or selected fields in one or more records.

FoxPro behavior: Generates an error.

Action: Remove the BLANK command. Use the SCATTER MEMVAR BLANK and the GATHER MEMVAR command to "blank" a record. For individual fields, use the REPLACE command.

Comment: FoxPro does not have full support for null values. Null values are fields filled with spaces and FoxPro can do this with date and character fields, though not numeric or logical fields. For null support in numeric fields, create a logical field that keeps track of whether the numeric field contains a value or is null.

Example: **FoxPro**

For dates:

```
REPLACE inv_date WITH {  /  /  }
```

For character fields:

```
REPLACE cust_name WITH SPACE(LEN(cust_name))
```

For records (this blanks the current record; numeric fields are set to zero):

```
SCATTER MEMVAR BLANK
GATHER MEMVAR
```

BROWSE
XE "BROWSE"§Level: Not flagged by the Program Analyzer.

dBASE IV behavior: Data are committed when the user exits a row.

FoxPro behavior: Data are committed when the user exits a field.

Action: None required.

BROWSEXE "BROWSE"§ COMPRESSXE "COMPRESS"§
XE "BROWSE COMPRESS"§Level: 3

dBASE IV behavior: The COMPRESS key word compresses the header region to one line.

FoxPro behavior: Not supported. The COMPRESS key word is ignored.

Action: None required.

Comment: COMPRESS in dBASE permits up to 19 rows to fit on one screen instead of 17. FoxPro 2.5 for Windows supports scalable fonts. Using a smaller point size will decrease row height and increase the number of records visible on one screen. In the example, the number of records visible goes from 25 (default font and type size) to 31 (Times New Roman, 6 pt.).

Example: **dBASE**

```
BROWSE COMPRESS
```

FoxPro

```
BROWSE FONT 'Times New Roman', 6
```

<u>BROWSEXE "BROWSE"§ NOFOLLOWXE "NOFOLLOW"§</u>
XE "BROWSE NOFOLLOW "§Level: 3

dBASE IV behavior: The NOFOLLOW key word prevents the record pointer from following a record to its new position in the index after you edit the key fields.

FoxPro behavior: The NOFOLLOW key word is ignored. In FoxPro BROWSE always behaves as if NOFOLLOW has been specified. Neither the changed record nor the record pointer are moved after a change to a key field.

Action: None required.

Comment: In cases where you want to simulate a dBASE BROWSE without NOFOLLOW, you can use the SHOW WINDOW <window name> REFRESH command. This will "move" the updated record. The record pointer can be moved in the same routine.

<u>BROWSE XE "BROWSE "§NOINITXE "NOINIT"§</u>
XE "BROWSE NOINIT "§Level: 3

dBASE IV behavior: Redisplays the previous BROWSE window configuration.

FoxPro behavior: Not supported. The NOINIT key word is ignored.

Action: None required. Specifying the LAST key word is identical to using NOINIT.

Example: **dBASE**

```
BROWSE NOINIT
```

FoxPro

```
BROWSE LAST
```

<u>BROWSEXE "BROWSE"§ NOMENUXE "NOMENU"§</u>
XE "BROWSE NOMENU "§Level: 3

dBASE IV behavior: The NOMENU option suppresses the menu bar entirely.

FoxPro behavior: The NOMENU option suppresses only the Browse pad in the menu bar.

Action: To suppress the menus entirely, use the SET SYSMENU OFF command prior to the BROWSE command, and SET SYSMENU ON or SET SYSMENU AUTOMATIC afterward.

Example: **dBASE**

```
BROWSE NOMENU
```

FoxPro

```
SET SYSMENU OFF
BROWSE
SET SYSMENU ON
```

Alphabetical List of Potential dBASE IV Issues

BROWSEXE "BROWSE"§ NOORGANIZEXE "NOORGANIZE"§ XE "BROWSE NOORGANIZE "§

Level:	2
dBASE IV behavior:	The NOORGANIZE key word (version 1.5 only) suppresses the Organize menu pad in the menu bar.
FoxPro behavior:	Generates an error.
Action:	Remove this key word.
Comment:	FoxPro does not have a menu pad called Organize. Commands similar to those on the Organize menu are found under the Database and Browse pads in FoxPro. The FoxPro menus are easily customizable. Individual menu items as well as entire pads can be disabled or removed. Refer to the section on menus in the *User's Guide* (included the FoxPro 2.5 documentation).

CALLXE "CALL"§

Level:	1
dBASE IV behavior:	CALL accepts an expression list.
FoxPro behavior:	CALL accepts only one expression.
Action:	Redesign binary routines into several routines and break the single dBASE CALL into several FoxPro CALL commands.

CALL()

XE "CALL()"§Level:	1
dBASE IV behavior:	This function provides an alternative to the CALL command for calling binary programs loaded into memory with the LOAD command.
FoxPro behavior:	Generates an error.
Action:	Substitute the CALL command and (if necessary) modify the called routine.
Comment:	The CALL command in FoxPro works the same as in dBASE IV. You might have to change the number of parameters or modify the binary routine to take into account the fact that the routine can supply a return value to the CALL function. When you use the CALL command, the binary routine must return values by changing the values of the memory variables passed as parameters.

CATALOG()XE "CATALOG()"§

Level:	2
dBASE IV behavior:	Returns the name of the active catalog file.
FoxPro behavior:	Generates an error.

Action:	Remove this function.

CERROR()
XE "CALL()"§Level: 2

dBASE IV behavior:	Undocumented function that returns the number of the last compiler error.
FoxPro behavior:	Generates an error.
Action:	Remove this function.

CHANGE()
See Network functions.

Comparison operators
XE "Comparison operators"§Level: 2

dBASE IV behavior:	The >=XE " >="§ (greater than or equal to) and <=XE "<="§ (less than or equal to) operators can also be written as => and =<.
FoxPro behavior:	Placing the equal sign before a less than character or greater than character (for example, =>XE " =>"§ or =<XE "=<"§) yields the "Missing operand" error message.
Action:	Replace occurrences of => with >= and =< with <=.

COMPLETED()
See Transaction processing.

CONVERT
See Network functions.

COPY TO ARRAY
XE "Comparison operators"§Level: 4

dBASE IV behavior:	In the FIELDS clause, the same field can be included more than once.
FoxPro behavior:	Including a field more than once in the FIELDS clause generates an error.
Action:	Remove multiple references to a single field.

CTOD()
XE "CTOD()"§Level: 3

dBASE IV behavior:	If the input to the CTOD function is a character string in which the month is greater than 12 or the day is greater than the number of days in the specified month, dBASE IV carries out a date addition and returns the resulting legitimate date. For example, CTOD("13/32/93") yields the date 02/01/94.

FoxPro behavior:	FoxPro accepts only legitimate dates as arguments. CTOD returns an empty date, which is displayed as " / / " if the input is not a legitimate date.
Action:	None required.
Comment:	You may wish to check the value return by CTOD() using the EMPTY function to make sure a legitimate date was entered.

<u>DEFINE BAR</u>
See MESSAGE.

DEFINE MENU
XE "DEFINE MENU"§Level: 4

dBASE IV behavior: DEFINE MENU in dBASE IV version 1.0 adds an extra space to each pad (and to the highlight bar that indicates the selected pad). DEFINE PAD without coordinates places one space between pads, resulting in up to three spaces between pads on the screen. dBASE IV versions 1.1 and 1.5 do not add leading and trailing spaces but do add one space between pads.

MESSAGE expressions are output to the active window (see MESSAGE).

FoxPro behavior: Same as dBASE IV 1.0. An extra space is added to each pad.

MESSAGE expressions are output to the desktop (see MESSAGE).

Action: Optionally add the NOMARGIN key word to the DEFINE MENU command to suppress the extra spaces and match the appearance of the menu in dBASE IV 1.1 or 1.5.

Comment: Pads on bar menus written in dBASE IV version 1.1 or 1.5 might not all fit on one line with the extra spaces.

Example: **dBASE** (1.1 or 1.5 only)

```
DEFINE MENU Main
```

FoxPro

```
DEFINE MENU Main NOMARGIN
```

DEFINE PAD
See MESSAGE.

DEFINE POPUP
See MESSAGE.

DESCENDING()XE "DESCENDING()"§ (version 1.5 only)
Level: 1

dBASE IV behavior: The DESCENDING function evaluates to .T. if an index accesses records in descending order, or .F. otherwise.

FoxPro behavior: Generates an error.

Action: The DESCENDING function must be removed.

DISPLAY USERS
See LIST USERS

END TRANSACTION
See Transaction processing.

Alphabetical List of Potential dBASE IV Issues

<u>ERROR()</u>
XE "ERROR()"§Level: 3

dBASE IV behavior: ERROR() returns the number corresponding to the error message trapped by the ON ERROR command.

FoxPro behavior: Behaves like dBASE, but often returns error numbers different from dBASE error numbers.

Action: Use the appropriate FoxPro error number. Check Appendix D to see which dBASE error messages correspond to which FoxPro error numbers. Refer to the *FoxPro Developer's Guide* for a list of all FoxPro error messages.

Comment: You might be able to remove parts of your error handling code because some dBASE errors will never be generated in FoxPro. For example, FoxPro does not use error 76 (" -: Concatenated string too large") or 77 because FoxPro supports character strings of up to 65K characters.

Note that many error numbers used for the same error in FoxPro and dBASE have *different* error messages. Thus, an error-trapping routine that tests the MESSAGE function rather than the ERROR function may might to respond to many common errors. See Appendixes C and D.

<u>FCREATE()XE "FCREATE()"§ and FOPEN()XE "FOPEN()"§</u>
Level: 2

dBASE IV behavior: The second input to the FOPEN and FCREATE functions, which represents the file attributes, must be "R" (read-only), "W" (write-only), "A" (append-only), "RW" or "WR (read and write), or "RA" or "AR" (read and append).

FoxPro behavior: The second input to the FOPEN and FCREATE functions is a numeric code that represents the file attributes and also allows you to specify buffered or unbuffered access.

Action: Replace dBASE file attributes with FoxPro attribute number.

Example: **dBASE**

```
file_handle = FCREATE("example.txt", "R")
```

FoxPro

```
file_handle = FCREATE("example.txt", "1")
```

MS-DOS Attribute(s)	FoxPro Attribute Number	dBASE Attribute Code
Read/write (default)	0	RW or WR

Append only	0*	A
Read and append	0*	RA or AR
Read only	1	R
Hidden	2	(not available)
Read Only/Hidden	3	(not available)
System	4	(not available)
Read Only/System	5	(not available)
System/Hidden	6	(not available)
Read Only/Hidden/System	7	(not available)

*There is no direct equivalent in FoxPro to append. Both read and write modes are enabled.

FDATE()XE "FDATE()"§ (version 1.5 only)

Level:	2
dBASE IV behavior:	The FDATE function returns the date stamp on the disk file specified as input.
FoxPro behavior:	Generates an error.
Action:	Replace with the ADIR function, or use UDF in FOXPROC.PRG.

Comment:	One function in FoxPro, ADIR(), returns file date, as well as size, time and attributes. A file or file skeleton can be passed as arguments. The return data is automatically placed into an appropriately sized array. The third column stores file date information. A UDF called FDATE() in FOXPROC.UDF allows you to leave instances of the dBASE function FDATE() unchanged if you wish. See the section titled "Calling user-defined functions" above.

Example: **dBASE**

```
file_date = FDATE('customer.dbf')
```

FoxPro

```
temp = ADIR(dir_array,'c:\foxprow\employee.dbf')
file_date = dir_array(3)
```

<u>FGETS()</u>

XE "FGETS()"§Level: 2

dBASE IV behavior:	You can specify the end-of-line character in the third argument of the FGETS and FPUTS functions (which is by default, a carriage return and linefeed).
FoxPro behavior:	FGETS and FPUTS functions process only text files that use a carriage return and line feed as line-end characters.
Action:	If a file contains line-end characters other than a carriage return (ASCII 13) or line feed (ASCII 10), perform a global search and replace on the file. Alternatively, replace FGETS with an FREAD loop that reads and tests each character until it finds the end-of-line character.

<u>FLDCOUNT()XE "FLDCOUNT()"§ (version 1.5 only)</u>

Level:	2
dBASE IV behavior:	This function returns the number of fields in a database.
FoxPro behavior:	Generates an error.
Action:	Replace with the equivalent FoxPro function FCOUNT. Or use UDF in FOXPROC.PRG.
Comment:	FCOUNT works exactly like FLDCOUNT. A UDF called FLDCOUNT() in FOXPROC.UDF allows you to leave instances of the dBASE function FLDCOUNT() unchanged if you wish. See the section titled "Calling user-defined functions" above.

Example: **dBASE**

```
FLDCOUNT('customer.dbf')
```

FoxPro

FCOUNT('customer.dbf')

FLDLIST()XE "FLDLIST()"§ (version 2.0 only)

Level:	1
dBASE IV behavior:	Returns the fields of a SET FIELDS TO list, or an individual field if the optional numeric argument is included.
FoxPro behavior:	Generates an error.
Action:	Replace FLDLIST() with SET("FIELDS", 1).
Comment:	See SET FIELDS for more information on the behavior of a field list when the SET FIELDS list spans more than one work area.

FOPEN()

See FCREATE()

FOR()XE "FOR()"§ (version 1.5 only)

Level:	2
dBASE IV behavior:	The FOR function returns the FOR clause used to create a conditional index tag.
FoxPro behavior:	Generates an error.
Action:	Replace with the equivalent FoxPro function SYS(2021) or use UDF in FOXPROC.PRG.
Comment:	A UDF called FOR() in FOXPROC.UDF allows you to leave instances of the dBASE function FOR() unchanged if you wish. See the section titled "Calling user-defined functions" above.

Example: **dBASE**

for_clause = FOR('cust_no.mdx',1)

FoxPro

for_clause = SYS(2021, 1)

FPUTS()

XE "FPUTS()"§Level: 2

dBASE IV behavior:	You can specify the end-of-line character in the third argument of the FPUTS and FGETS functions (which is, by default, a carriage return and linefeed).
FoxPro behavior:	FPUTS and FGETS functions process only text files that use a carriage return and linefeed as line-end characters.
Action:	If a file contains line-end characters other than a carriage return (ASCII 13) or linefeed (ASCII 10), perform a global search and replace on the file.

Alternatively, to write a file with nonstandard end-of-line characters, use FWRITE to write the string that contains the line of text plus the end of line characters.

FSIZE()
XE "FSIZE() "§Level: 2

dBASE IV behavior:	The FSIZE function returns the size of the *file* specified as input.
FoxPro behavior:	The FSIZE function returns the size of the *field* specified as input.
Action:	Replace with the ADIR function.
Comment:	One function in FoxPro, ADIR(), returns file size, as well as date, time and attributes. A file or file skeleton can be passed as arguments. The return data is automatically placed into an appropriately sized array. The second column stores file size information.

Example: **dBASE**

```
file_size = FSIZE(customer.dbf)
```

FoxPro

```
temp = ADIR(dir_array,'c:\foxprow\employee.dbf')
file_size = dir_array(2)
```

FTIME() XE "FTIME() "§(version 1.5 only)

Level:	2
dBASE IV behavior:	The FTIME function returns the time stamp on the disk file specified as input.
FoxPro behavior:	Generates an error.
Action:	Replace with the ADIR function, or use UDF in FOXPROC.PRG.
Comment:	One function in FoxPro, ADIR(), returns file time, as well as date, size and attributes. A file or file skeleton can be passed as arguments. The return data is automatically placed into an appropriately sized array. The fourth column stores file time information. A UDF called FTIME() in FOXPROC.UDF allows you to leave instances of the dBASE function FTIME() unchanged if you wish. See the section titled "Calling user-defined functions" above.

Example: **dBASE**

```
file_time = FTIME('customer.dbf')
```

FoxPro

```
temp = ADIR(dir_array,'c:\foxprow\employee.dbf')
file_time = dir_array(4)
```

HOME()XE "HOME()"§

Level:	2
dBASE behavior:	Returns home directory of dBASE IV.
FoxPro behavior:	Generates and error.

Action:	Replace with the equivalent function SYS(2004)

ID()XE "ID()"§

Level:	1
dBASE behavior:	Returns name of current user on a multiuser system.
FoxPro behavior:	Generates an error.
Action:	Substitute a test based on the SYS(0) function.
Comment:	SYS(0) returns the network computer name and number when FoxPro is running on a network. A machine number and name must first be assigned by the network software and the network shell must be loaded. On Novell networks, add the following to the system login script:
	MACHINE="%USER_ID,%P_STATION,%LOGIN_NAME"
	If FoxPro is not running on a network or a machine number and name haven't been assigned by the network, SYS(0) returns a string of spaces (10 in FoxPro for MS-DOS or 15 in FoxPro for Windows), followed by a pound sign (#), space, and 0. When the single-user version of FoxPro is running, SYS(0) evaluates to 1.

INKEY()XE "INKEY()"§, LASTKEY()XE "LASTKEY()"§, READKEY()XE "READKEY()"§

Level:	3
dBASE behavior:	FoxPro and dBASE, in many cases, map keys to different values.
FoxPro behavior:	These functions work the same way as in dBASE, but FoxPro key values may differ, so unmodified dBASE programs may not behave the same way in FoxPro as they did before.
Action:	FoxPro reads key assignments from a macro file with the extension .FKY. Use the dBASE.FKY macro file, included on the Migration Kit disk, so FoxPro keystrokes are mapped to dBASE keystrokes. Then include the command RESTORE MACROS FROM dBASE.FKY in your program.
	Alternately, you could change your program code so it uses FoxPro's "native" key assignments. See the table in Appendix G for a list of these values.

ISBLANK()
XE "ISBLANK()"§Level: 2

dBASE IV behavior: The ISBLANK function returns the null status of any variable.

FoxPro behavior: Generates an error.

Action: Replace ISBLANK with the FoxPro EMPTY function or use UDF in FOXPROC.PRG

Comment: ISBLANK in dBASE and EMPTY in FoxPro are the same when dealing with date and character fields. With numeric fields, EMPTY returns .T. when the field has no value (is null) or when the value is 0. ISBLANK in dBASE would return .T., meaning the value is null.

For null support in numeric fields, create a logical field that keeps track of whether the numeric field contains a value or is null.

A UDF called ISBLANK() in FOXPROC.UDF allows you to leave instances of the dBASE function ISBLANK() unchanged if you wish. See the section titled "Calling user-defined functions" above.

Example: **dBASE**

```
null_state = ISBLANK(inv_date)
```

FoxPro

```
null_state = EMPTY(inv_date)
```

ISMARKED()
See Transaction processing.

ISMOUSE()XE "ISMOUSE()"§ (version 2.0 only)
Level: 2

dBASE IV behavior: Returns True (.T.) if a mouse driver is installed.

FoxPro behavior: Generates an error.

Action: Remove the function.

Comment: There is no equivalent in FoxPro.

KEYBOARDXE "KEYBOARD"§ CLEARXE "CLEAR"§
XE "KEYBOARD CLEAR "§Level: 2

dBASE IV behavior: The CLEAR key word clears the keyboard buffer before executing the KEYBOARD command.

FoxPro behavior: Generates an error.

Action: Remove CLEAR and place the CLEAR TYPEAHEAD command before the KEYBOARD command.

Example: **dBASE**

```
KEYBOARD cust_name + address + city + state CLEAR
```

FoxPro

```
CLEAR TYPEAHEAD
KEYBOARD cust_name + address + city + state
```

KEYMATCH()XE "KEYMATCH()"§ (version 2.0 only)

Level:	2
dBASE IV behavior:	Searches a specified index tag for a given key without changing the active index or moving the record pointer.
FoxPro behavior:	Generates an error.
Action:	Remove the function.
Comment:	There is no equivalent in FoxPro.

LIKE()

XE "LIKE()"§Level:	3
dBASE IV behavior:	Trailing blanks in both the pattern and target are trimmed before the comparison is made.
FoxPro behavior:	The pattern and target are both used as is and trailing blanks are significant.
Action:	Use the RTRIM function or SET COMPATIBLE DB4 on.

Example: **dBASE**

```
LIKE(var1,var2)
```

FoxPro

```
LIKE(RTRIM(var1),var2)
```

LIST USERSXE "LIST USERS"§

Level:	2
dBASE behavior:	Identifies the workstations currently logged into a dBASE networking environment.
FoxPro behavior:	Not supported. LIST USERS is ignored.
Action:	None required.
Comment:	See Appendix E for alternatives.

LKSYS

See Network functions.

LOCK()XE "LOCK()"§, RLOCK()XE "RLOCK()"§

Level:	2
dBASE IV behavior:	By default, you can lock more than one record at a time with these functions.
FoxPro behavior:	By default, FoxPro allows locking one record at a time.
Action:	To permit multiple locks, use the SET MULTILOCKS ON command. Add this command to the main startup program for network applications.

Comment: With SET MULTILOCKS ON, the ability to place multiple record locks is the same in both FoxPro and dBASE IV.

Example: **FoxPro**

```
SET MULTILOCKS ON
```

MEMORY()XE "MEMORY()"§ (version 2.0 only)

Level:	2
dBASE IV behavior:	Allows you to include a parameter of from 0 to 7. Each value represents a region of memory that MEMORY() will return.
FoxPro behavior:	Generates an error if you include a parameter.
Action:	Replace the MEMORY() command with the corresponding FoxPro commands as shown in the table below.

dBASE	**FoxPro equivalent**
MEMORY(0)	SYS(1001)+SYS(1016)
MEMORY(1)	No equivalent
MEMORY(2)	SYS(12)
MEMORY(3)	SYS(1001)
MEMORY(4)	SYS(23)
MEMORY(5)	SYS(23)
MEMORY(6)	SYS(1016)
MEMORY(7)	No equivalent

MESSAGEXE "MESSAGE"§XE "@...GET MESSAGE"§ (@GET, DEFINE BARXE "DEFINE BAR"§, DEFINE MENUXE "DEFINE MENU"§ DEFINE PADXE "DEFINE PAD"§, DEFINE POPUPXE "DEFINE POPUP"§)

Level:	2
dBASE IV behavior:	MESSAGE expressions are output to the active window.
FoxPro behavior:	MESSAGE expressions are output to the desktop.
Action:	There are three ways to resolve this issue:
	a) Use the SET MESSAGE WINDOW command to output the message expression to the window of your choice.
	b) Shorten windows by one line so the message on the desktop becomes visible.
	c) Use the desktop instead of a window (using the SAVE SCREEN and RESTORE SCREEN commands to simulate use of windows).
Comment:	This problem only occurs if a window is defined that covers the last line of the screen where the message is output.

NETWORK()
XE "NETWORK()"§Level: 2

dBASE IV behavior: This function returns .T. only if dBASE is currently running on a network.

FoxPro behavior: In FoxPro 2.5, NETWORK always returns .T.. In FoxPro 2.0, the single-user version will evaluate network to .F. while the multiuser version will return .T..

Action: Substitute a test based on the SYS(0) function.

Comment: SYS(0) returns the network computer name and number when FoxPro is running on a network. A machine number and name must first be assigned by the network software and the network shell must be loaded. On Novell networks, add the following to the system login script:

MACHINE="%USER_ID,%P_STATION,%LOGIN_NAME"

If FoxPro is not running on a network or a machine number and name haven't been assigned by the network, SYS(0) returns a string of 15 spaces, followed by a pound sign (#), space, and zero. When the single-user version of FoxPro is running, SYS(0) evaluates to 1.

Example: **dBASE**

```
on_network = NETWORK()
```

FoxPro

```
on_network = LEFT(SYS(0),10)<>SPACE(10)
```

Network functions
XE "Network functions"§Level: 1

dBASE IV behavior: dBASE IV supports a mechanism for detecting modifications to the current record on a network, using the CONVERT, CHANGE, LKSYS, and USER commands and functions.

FoxPro behavior: FoxPro ignores the CONVERT command. The CHANGE, LKSYS, and USER functions generate errors.

Action: Remove the CHANGE, LKSYS, and USER functions. Lock detection schemes that depend on these functions need to be written using other methods in FoxPro.

ON BARXE "ON BAR"§ (version 2.0 only)
Level: 2

dBASE IV behavior: Executes a command when a specified popup bar is *highlighted*.

FoxPro behavior: Allows execution of an ACTIVATE POPUP or ACTIVATE MENU statement when a specified popup bar is *selected*.

Action:	If it is necessary to perform an action other than ACTIVATE POPUP or ACTIVATE MENU, replace ON BAR with ON SELECTION BAR in FoxPro.

ON EXIT BARXE "ON EXIT BAR"§ (version 2.0 only)

Level:	2
dBASE IV behavior:	Executes a command when the cursor (highlight) leaves a specified bar.
FoxPro behavior:	Generates an error.
Action:	Remove the command.
Comment:	There is no equivalent in FoxPro.

ON EXIT MENUXE "ON EXIT MENU"§ (version 2.0 only)

Level:	2
dBASE IV behavior:	Executes a command when the cursor (highlight) leaves a specified menu.
FoxPro behavior:	Generates an error.
Action:	Remove the command.
Comment:	There is no equivalent in FoxPro.

ON EXIT PAD

Level:	2
dBASE IV behavior:	Executes a command when the cursor (highlight) leaves a specified bar.
FoxPro behavior:	Generates an error.
Action:	Remove the command.
Comment:	There is no equivalent in FoxPro.

ON EXIT POPUPXE "ON EXIT POPUP"§ (version 2.0 only)

Level:	2
dBASE IV behavior:	Executes a command when the cursor (highlight) leaves a specified popup.
FoxPro behavior:	Generates an error.
Action:	Remove the command.
Comment:	There is no equivalent in FoxPro.

ON KEY LABELXE "ON KEY LABEL"§

Level:	3
dBASE IV behavior:	Uses "-" character in multi-key labels.
FoxPro behavior:	Uses "+" character in mulit-key labels.
Action:	Replace any "-" (dash) characters in key labels and replace them with "+" (plus sign) characters.
Comment:	dBASE and FoxPro key labels are the same except for the concatenating character in combination keystrokes.

Example: **dBASE**

```
ON KEY LABEL ALT-A DO myprog.prg
```

FoxPro

```
ON KEY LABEL ALT+A DO myprog.prg  && change "-" to "+"
```

ON MENUXE "ON MENU"§ (version 2.0 only)

Level:	2
dBASE IV behavior:	Executes a command when any popup bar without an ON PAD handler is highlighted.
FoxPro behavior:	Generates an error.
Action:	Replace ON MENU with ON SELECTION MENU in FoxPro.
Comment:	ON SELECTION MENU in FoxPro is functionally identical to ON MENU in dBASE IV except that ON SELECTION MENU executes the specified command when a bar is *selected* whereas ON MENU in dBASE IV executes the specified command when a bar is *highlighted*.

Example: **dBASE**

```
ON MENU File DO my_prog
```

FoxPro

```
ON SELECTION MENU File DO my_prog
```

ON MOUSEXE "ON MOUSE"§ (version 2.0 only)

Level:	2
dBASE IV behavior:	Executes a given command when the left mouse button is clicked.
FoxPro behavior:	Generates an error.
Action:	Replace ON MOUSE with ON KEY LABEL LEFTMOUSE.
Comment:	ON MOUSE is most typically used in dBASE IV to create user-friendly controls such as check boxes and radio buttons. Although the dBASE IV approach will work in FoxPro, you might consider substituting the code that supports these controls with FoxPro @..GET push buttons, radio buttons, check boxes, lists, etc.

Example: **dBASE**

```
ON MOUSE DO mouse_proc
```

FoxPro

```
ON KEY LABEL LEFTMOUSE DO mouse_proc
```

ON PADXE "ON PAD"§ (version 2.0 only)

Level:	2
dBASE IV behavior:	Executes any command when a specified menu pad is highlighted.
FoxPro behavior:	Allows execution of an ACTIVATE POPUP or ACTIVATE MENU statement when a specified menu pad is highlighted.
Action:	If it is necessary to perform an action other than ACTIVATE POPUP or ACTIVATE MENU, replace ON PAD with ON SELECTION PAD in FoxPro.
Comment:	ON SELECTION PAD in FoxPro is functionally identical to ON PAD in dBASE IV except that ON SELECTION PAD executes the specified command when a pad is *selected* whereas ON PAD in dBASE IV executes the specified command when a pad is *highlighted*.

Example: **dBASE**

```
ON PAD bachelor OF type DO bach_pad
```

FoxPro

```
ON SELECTION PAD bachelor OF type DO bach_pad
```

ON POPUPXE "ON POPUP"§ (version 2.0 only)

Level:	2
dBASE IV behavior:	Executes a command when any popup bar without an ON BAR handler is highlighted.
FoxPro behavior:	Generates an error.
Action:	Replace ON POPUP with ON SELECTION POPUP in FoxPro.
Comment:	ON SELECTION POPUP in FoxPro is functionally identical to ON POPUP in dBASE IV except that ON SELECTION POPUP executes the specified command when a bar is *selected* whereas ON POPUP in dBASE IV executes the specified command when a bar is *highlighted*.

Example: **dBASE**

```
ON POPUP just DO it
```

FoxPro

```
ON SELECTION POPUP just DO it
```

ON SELECTION POPUP
XE "ON SELECTION POPUP"§Level: 3

dBASE IV behavior: Any data displayed or windows activated by the ON SELECTION POPUP command may cover the popup. After this command or procedure terminates, the popup will reappear on the screen.

FoxPro behavior: A popup remains on top of windows or data displayed when FoxPro executes the procedure named in the ON SELECTION POPUP command.

Action: None required.

Comment: If desired, add a HIDE POPUP command to the procedure called by ON SELECTION POPUP. The popup reappears automatically when the command or procedure terminates and the popup regains control.

Setting COMPATIBLE on will also result in the same behavior as dBASE.

Example: **dBASE**

```
ON SELECTION POPUP edit_pop DO edit_proc

PROCEDURE edit_proc
<procedure code>
```

FoxPro

```
ON SELECTION POPUP edit_pop DO edit_proc

PROCEDURE edit_proc
HIDE POPUP edit_pop
<procedure code>
SHOW POPUP edit_pop
```

ON SELECTION POPUP BLANK
XE "ON SELECTION POPUP BLANK"§Level: 2

dBASE IV behavior: The optional BLANK key word clears the pop-up menu from the screen before executing any commands. The pop-up menu will be redrawn upon return from the executed command.

FoxPro behavior: Generates an error.

Action: Remove the BLANK key word.

Comment: See ON SELECTION POPUP.

PADPROMPT()XE "PADPROMPT()"§ (version 2.0 only)
Level: 2

dBASE IV behavior: Returns the prompt text of the specified pad in a specified menu, or in the current menu if none is given.

FoxPro behavior: Generates an error.

Action: Replace PADPROMPT() with PRMPAD().

Comment: PRMPAD() in FoxPro requires that you specify the name of the menu.

Example: **dBASE**

```
pr_txt=PADPROMPT("File")
```

FoxPro

```
pr_txt=PRMAD("menu_1","File")
```

PCOUNT()XE "PCOUNT()"§ (version 1.5 only)

Level: 2

dBASE IV behavior: The PCOUNT function evaluates to the number of parameters passed to a user-defined function.

FoxPro behavior: Generates an error.

Action: Replace PCOUNT with the PARAMETERS function or use UDF in FOXPROC.PRG.

Comment: A UDF called PCOUNT() in FOXPROC.UDF allows you to leave instances of the dBASE function PCOUNT() unchanged if you wish. See the section titled "Calling user-defined functions" above.

Example: **dBASE**

```
param_no = PCOUNT()
```

FoxPro

```
param_no = PARAMETERS()
```

PROTECT

See Security

REPLACE FROM ARRAY

XE "REPLACE FROM ARRAY"§Level: 2

dBASE IV behavior: Replaces the contents of one or more records with the corresponding fields in a two-dimensional array.

FoxPro behavior: Generates an error.

Action: Replace with the commands SCATTER and GATHER.

Comment: The SCATTER command creates memory variables from fields. GATHER writes those memory variables back to the database. These commands are faster than COPY TO and REPLACE FROM, plus, the variables created are named based on the field names, rather than becoming a numbered element in an array.

If this command is used to replace multiple records, create a loop using the GATHER command.

Example: **dBASE**

```
COPY TO ARRAY rec_array
<<@ SAYs and GETS>>
READ
REPLACE FROM ARRAY rec_array
```

FoxPro

```
SCATTER MEMVAR
<<@ SAYs and GETS>>
READ
GATHER MEMVAR
```

<u>RESET</u>

See Transaction processing.

RESTORE FROMXE "RESTORE FROM"§

Level:	3
dBASE IV behavior:	A .MEM extension is assumed if none is supplied.
FoxPro behavior:	If no extension is specified, FoxPro will search for the filename without an extension.
Action:	Add an explicit extension.

RESTORE SCREENXE "RESTORE SCREEN"§

Level:	2
dBASE IV behavior:	The SAVE SCREEN command saves a screen image in memory but not in a memory variable. RESTORE SCREEN can be executed from anywhere in the application to restore the screen image.
FoxPro behavior:	The SAVE SCREEN command stores the screen image to a memory variable. The memory variable must be available in the procedure or function where the RESTORE SCREEN command is issued.
Action:	If SAVE SCREEN and RESTORE screen are in the same procedure or function, no change is necessary. If not, declare the FoxPro memory variable public before executing the SAVE SCREEN command.
Comment:	If the screen is saved to a private variable, RESTORE SCREEN will generate a "Variable not found" error message if issued from a procedure or function different from where SAVE SCREEN was issued.

Example: **dBASE**

```
SAVE_SCREEN TO screen_var
```

FoxPro

```
PUBLIC screen_var
SAVE SCREEN to screen_var
```

RLOCK()
See LOCK()

ROLLBACK
See Transaction processing.

ROLLBACK ()
See Transaction processing.

RUN()XE "RUN()"§

Level:	2

dBASE behavior:	Runs an external program.
FoxPro behavior:	Generates an error.
Action:	Substitute the RUN command.

Security
XE "Security"§Level: 1

dBASE IV behavior:	Security in dBASE IV is implemented with the PROTECTXE "PROTECT"§, ACCESSXE "ACCESS"§, USERXE "USER"§, SET ENCRYPTIONXE "SET ENCRYPTION"§, and SET("ENCRYPTION")XE "SET(\"ENCRYPTION\")"§ commands and functions.
FoxPro behavior:	FoxPro ignores the SET ENCRYPTION command. The ACCESS function always returns 0. The USER function and PROTECT command generate an errors.
Action:	Remove the USER function. Copy any encrypted databases to unencrypted files in dBASE before you begin using FoxPro.
Comment:	If the application requires a detailed security system or file encryption, you can design your own or you can use third-party products.

SELECT()
XE "SELECT()"§Level: 2

dBASE IV behavior:	In version 1.1, this function takes no inputs and returns the highest numbered available work area. In version 1.5, when evaluated with no inputs, it returns the lowest numbered available work area. With an alias specified as input, it returns the number of the work area in which a database with the specified alias is open. Also, if the current work area is changed by a UDF, it changes back automatically after the UDF is executed.
FoxPro behavior:	SELECT() and SELECT(0) return the number of the selected work area. SELECT(1) returns the highest numbered available work area. In FoxPro, if the current work area is changed in a UDF, that will become the current work area after the UDF has executed unless another is explicitly selected.
Action:	When using SELECT to open a database in an available work area, change the syntax to SELECT(1) to avoid closing the database already open in the current work area. To select the work area of a particular database, use the SELECT command. Or SET COMPATIBLE on and SELECT() will behave as it does in dBASE.

Comment: If your program tests the value of SELECT and depends on numbered work areas, you might need to change the program's logic to take account of the FoxPro SELECT function returning the highest available work area rather than the lowest.

Example: **dBASE**

```
SELECT()
```

FoxPro

```
SELECT(1)
```

SET("ATTRIBUTES")XE "SET(\"ATTRIBUTES\")"§

Level:	2
dBASE IV behavior:	SET("ATTRIBUTES") returns a string consisting of the seven color pairs established with the SET COLOR OF command.
FoxPro behavior:	Generates an error.
Action:	In applications generated by the dBASE Applications Generator, this code can usually be commented out.
	If a program relies on finding out colors set using the SET COLOR TO command, you should write code that saves these colors to memory variables after the SET COLOR TO command is issued. These memory variables can then be interrogated instead of using the SET("ATTRIBUTES") function.
Comment:	SET COLOR TO acts the same in dBASE and FoxPro, so programs that manipulate color usually behave the same way as well.
	FoxPro will return dBASE II-style color information from the SYS(2001,'COLOR') or SET("COLOR") functions.

SET("BORDER")XE "SET(\"BORDER\")"§

Level:	3
dBASE IV behavior:	Returns border type key word.
FoxPro behavior:	Returns a string of ten characters which compose the border.
Action:	Parse the return string and use the ASC() function, which returns the ASCII number. Check to see whether the ASCII code is of single, double, or panel type.

SET CATALOG

XE "SET CATALOG"§Level:	1
dBASE behavior:	The catalog is a database file with the extension .CAT and contains a record for each file in an application. The following catalog-related commands and functions are supported in dBASE:
	SET CATALOG TO <catalog> establishes the current catalog.
	SET CATALOG ON | OFF activates and deactivates the catalog.
FoxPro behavior:	Not supported. Catalog commands are ignored.
Action:	Use ADIR() to find files matching a particular file skeleton.

SET("CATALOG")XE "SET(\"CATALOG\")"§

Level:	2
dBASE IV behavior:	Returns ON or OFF.

FoxPro behavior:	Generates an error.
Action:	Remove this function. It is not supported in FoxPro.

SET COLOR TO
XE "SET COLOR TO"§Level: 3

dBASE IV behavior: SET COLOR TO (with no color pairs listed) resets the screen colors to black and white.

FoxPro behavior: SET COLOR TO leaves the current screen colors unchanged.

Action: None required. To get a monochrome color scheme in FoxPro, use the code in the example below.

Example: **dBASE**

```
SET COLOR TO
```

FoxPro

```
SET COLOR TO W/N, N/W, N
or
SET COLOR SCHEM TO monochrome
```

SET DBTRAPXE "SET DBTRAP"§ (versions 1.1 and 1.5 only)
Level: 1

dBASE IV behavior: When DBTRAP is set on, UDFs and interrupt routines are prevented from executing certain commands and functions.

FoxPro behavior: Generates an error.

Action: Remove this function.

Comment: You may add commands to UDFs to save the environment at the beginning and restore it at the end. FoxPro relies on the programmer not to execute commands such as PACK or MODIFY STRUCTURE in UDFs or interrupt routines that might disrupt the function or procedure that called the UDF or interrupt routine.

SET DESIGNXE "SET DESIGN"§
Level: 2

dBASE IV behavior: This command disables all design modes and prevents the user from creating or editing databases, reports, screens, queries, or applications.

FoxPro behavior: Not supported. SET DESIGN is ignored.

Action: None required.

Comment: By modifying the FoxPro system menu, you can easily remove or disable the New and Open options from the File menu to prevent users from creating or modifying files.

SET("DESIGN")XE "SET(\"DESIGN\")"§
Level: 2

dBASE IV behavior: Returns ON or OFF.

FoxPro behavior:	Generates an error.
Action:	Remove this function.
Comment:	See the SET DESIGN command, above.

SET DIRECTORY
XE "SET DIRECTORY"§Level: 2

dBASE IV behavior: This command establishes the full path of the default directory.

FoxPro behavior: Generates an error.

Action: Substitute the SET DEFAULT command.

Comment: SET DEFAULT in FoxPro acts exactly like SET DIRECTORY in dBASE. In FoxPro, SET DEFAULT accepts a full subdirectory path. In dBASE IV, it accepts only a disk drive letter.

Example: **dBASE**

```
SET DIRECTORY TO c:\data
```

FoxPro

```
SET DEFAULT TO c:\data
```

SET("DIRECTORY")XE "SET(\"DIRECTORY\")"§
Level: 2

dBASE IV behavior: Returns the full path of the default directory.

FoxPro behavior: Generates an error.

Action: Substitute the SET("DEFAULT") and SYS(2003) functions.

Comment: SET("DEFAULT") in FoxPro acts exactly like SET ("DIRECTORY") in dBASE. In FoxPro, SET DEFAULT accepts a full subdirectory path. In dBASE IV, it accepts only a disk drive letter.

Example: **dBASE**

```
curr_dir = SET("DIRECTORY")
```

FoxPro

```
curr_dir = SET("DEFAULT")+ SYS(2003)
```

SET("DISPLAY")XE "SET(\"DISPLAY\")"§
Level: 2

dBASE IV behavior: Returns the current video display mode.

FoxPro behavior: Generates an error.

Action: Store the previous video display mode in a variable before switching modes.

Example: **FoxPro**

To store the video mode:

```
vmd=LEFT(SYS(2006),AT("/",SYS(2006))-1)+STR(SROWS(),2)
SET DISPLAY TO VGA50
```

To reset the video mode:

```
SET DISPLAY TO &vmd
```

SET ENCRYPTION and SET("ENCRYPTION")
See Security.

SET FIELDS
XE "SET FIELDS"§Level: 1

dBASE IV behavior: a) SET FIELDS can specify fields from multiple databases.

b) SET FIELDS TO without inputs changes SET FIELDS to off.

c) SET FIELDS /r makes a field read only.

FoxPro behavior: a) Fields in the SET FIELDS command come from one database, but each work area can have its own field list.

b) SET FIELDS TO without inputs changes the field list to the null string.

c) The fields in a BROWSE can be made read only, but not individual fields in a SET FIELDS command.

Action: a) Create a memory variable that stores a field list and macro substitute it into list-type commands.

or

Convert SET FIELDS commands with fields from more than one database to separate SET FIELDs commands for each database.

b) SET DB4 COMPATIBLE on and SET FIELDS without inputs will SET FIELDS to off.

Example: **dBASE**

```
SET FIELDS TO customer->cust_id, orders->order_amt
LIST
```

FoxPro

```
fld_strng = "customer.cust_id, orders.order_amt"
LIST &fld_strng
```

or

```
SET FIELDS TO customer.order_amt
LIST
SET FIELDS TO orders.cust_id
LIST
```

SET FORMAT
XE "SET FORMAT"§Level: 2

dBASE IV behavior: dBASE format files (.FMTs) can contain setup and cleanup code in addition to @SAYs and @GETs.

FoxPro behavior: Format files with commands and functions other than @SAYs and @GETs generate an error.

Action: Move cleanup and setup code outside the format file.

Comment: If you convert .SCR files to .SCX, replace the commands that open the format file and initiate editing (EDIT, CHANGE, APPEND, or

READ) and that close the format file afterwards with a DO command that calls the generated screen file.

<u>SET IBLOCK TO (version 2.0 only)</u>

Level:	2
dBASE IV behavior:	Changes the default size of the indexing block to enhance performance.
FoxPro behavior:	Generates an error.
Action:	Remove the SET IBLOCK command. It is not necessary.
Comment:	Because Rushmore optimization in FoxPro is not affected by the size of the indexing block, this command is not necessary.

<u>SET INSTRUCT</u>

XE "SET INSTRUCT"§Level:	3
dBASE IV behavior:	Determines the level of prompting in the Control Center.
FoxPro behavior:	Not supported. SET INSTRUCT is ignored.
Action:	Remove this command.

<u>SET("INSTRUCT")XE "SET(\"INSTRUCT\")"§</u>

Level:	2
dBASE IV behavior:	Returns ON or OFF.
FoxPro behavior:	Generates an error.
Action:	Remove this function.

<u>SET KEYXE "SET KEY"§ (version 1.5 only)</u>

Level:	2
dBASE IV behavior:	Displays only records whose ordering index matches a specified condition. This command uses an index key rather than searching the database sequentially from the top.
FoxPro behavior:	Generates an error.
Action:	Remove this command and use the SET FILTER command.
Comment:	SET FILTER also uses an index key. If the filter expression is optimizable, Rushmore™ will speed execution still further.

Example: **dBASE**

```
SET ORDER TO TAG ZIP
SET KEY TO "94000","94999"
```

FoxPro

```
SET FILTER TO zip >= "94000" AND zip <= "94999"
```

SET LDCHECKXE "SET LDCHECK"§ (version 2.0 only)

Level:	2
dBASE IV behavior:	Enables or disables language driver ID checking.
FoxPro behavior:	Generates an error.
Action:	Remove this command.
Comment:	There is no equivalent command in FoxPro. If you need international support, make sure you have FoxPro 2.5a which offers state-of-the-art support for code page translation and multiple collate sequences.

SET LIBRARYXE "SET LIBRARY"§ (version 1.5 only)

Level:	1
dBASE IV behavior:	This command establishes a special procedure file that remains open in addition to any others opened with SET PROCEDURE TO.
FoxPro behavior:	SET LIBRARY TO opens a FoxPro Application Programming Interface library. If the command does not include an extension, it generates the "File does not exist" error message. If it does, non-API library files generate the "Library file is invalid" error message. The FoxPro SET PROCEDURE TO command allows only one procedure file to be open at a time.
Action:	Change SET LIBRARY TO to SET PROCEDURE TO. (Note that SET PROCEDURE will close any procedure file that is open.) To make the routines in the dBASE IV procedure library available throughout an application, add them to the procedure file that you open with SET PROCEDURE TO or place them in the main startup program for the application.

SET("LIBRARY")XE "SET(\"LIBRARY\")"§ (version 1.5 only)

Level:	2
dBASE IV behavior:	Returns a filename.
FoxPro behavior:	Not supported. SET ("LIBRARY") is ignored.
Action:	Change SET ("LIBRARY") to SET ("PROCEDURE").
Comment:	See also the SET LIBRARY command, above.

Example: **dBASE**

```
curr_lib = SET("LIBRARY")
```

FoxPro

```
curr_lib = SET("PROCEDURE")
```

Alphabetical List of Potential dBASE IV Issues

SET MBLOCKXE "SET MBLOCK"§ (version 2.0 only)

Level:	2
dBASE IV behavior:	Changes the default size of blocks that are allocated to new memo field files.
FoxPro behavior:	Generates an error.
Action:	Replace SET MBLOCK with SET BLOCKSIZE or remove the SET MBLOCK command.
Comment:	SET MBLOCK in dBASE IV is similar to SET BLOCKSIZE, found in both FoxPro and dBASE IV. The difference is that SET MBLOCK can be used independently with SET IBLOCK whereas SET BLOCKSIZE in dBASE IV affects block size for both indexes and memo files. Since Rushmore optimization in FoxPro is not affected by the size of the indexing block and since SET BLOCKSIZE in FoxPro does not affect the index block size, this command is not necessary.

SET MESSAGEXE "SET MESSAGE"§

Level:	3
dBASE IV behavior:	dBASE supports the optional AT key word for specifying message location.
FoxPro behavior:	FoxPro ignores the AT key word.
Action:	None required.
Comment:	You can replace AT with one of the FoxPro alignment key words LEFT, CENTER, or RIGHT.

SET PAUSEXE "SET PAUSE"§

Level:	3
dBASE IV behavior:	Causes SQL Select output to pause after each full screen.
FoxPro behavior:	SET PAUSE is ignored. FoxPro behaves as if PAUSE is on by default.
Action:	None required.

SET("PAUSE")XE "SET(\"PAUSE\")"§

Level:	2
dBASE IV behavior:	Causes SQL Select output to pause after each full screen.
FoxPro behavior:	Generates an error. FoxPro behaves as if PAUSE is on by default.
Action:	Remove this function.

SET PRECISIONXE "SET PRECISION"§

Level:	3

dBASE IV behavior:	Determines the number of digits between 10 and 20 used in math calculations. The default is 16.
FoxPro behavior:	Not supported. SET PRECISION is ignored.
Action:	Remove this command.
Comment:	Precision in FoxPro is 16 digits and is not settable.

SET("PRECISION")XE "SET(\"PRECISION\")"§

Level:	2
dBASE IV behavior:	Returns the number of digits used in math calculations.
FoxPro behavior:	Generates an error.
Action:	Remove this function.
Comment:	Precision in FoxPro is 16 digits and not settable.

SET PRINTER TO FILEXE "SET PRINTER TO FILE"§

Level:	3
dBASE IV behavior:	If no file extension is specified, dBASE writes a file with a .PRT extension.
FoxPro behavior:	If no extension is supplied, FoxPro writes a file without an extension.
Action:	None required. SET DB4 COMPATIBLE on and .PRT will be the extension assigned by default to print files.

SET("SQL")XE "SET(\"SQL\")"§

Level:	2
dBASE IV behavior:	Returns ON or OFF.
FoxPro behavior:	Generates an error.
Action:	Remove this function.
Comment:	SQL is integrated in FoxPro and does not need to be turned on or off. See SQL.

SET TITLEXE "SET TITLE"§

Level:	3
dBASE IV behavior:	Turns the catalog file title prompt on and off.
FoxPro behavior:	Not supported. SET TITLE is ignored.
Action:	None necessary.
Comment:	Catalogs are not supported in FoxPro.

SET("TITLE")XE "SET(\"TITLE\")"§

Level:	3

dBASE IV behavior:	Returns ON or OFF.
FoxPro behavior:	Always returns false.
Action:	None necessary.
Comment:	Catalogs are not supported in FoxPro.

SET TRAP
XE "SET TRAP"§Level: 3

dBASE IV behavior: When on, the debugger is invoked when you press the ESC key.

FoxPro behavior: Not supported. SET TRAP is ignored.

Action: None required. Run program with trace and debug window open.

Comment: FoxPro is a windowing application, so you can see both your application and the debug and trace windows simultaneously. There is no need to toggle between your application and your debugging environment.

To imitate SET TRAP, use the SET ECHO command. This will bring up the Trace window when an error occurs.

Example: **FoxPro**

```
ON ERROR SET ECHO ON
```

SET("TRAP")XE "SET(\"TRAP\")"§
Level: 2

dBASE IV behavior: Returns ON or OFF.

FoxPro behavior: Generates an error.

Action: Run program with trace and debug window open.

SET VIEWXE "SET VIEW"§
XE "SET (\"TRAP\")"§Level: 2

dBASE IV behavior: Performs a query or restores a view from a dBASE III Plus view (.VUE) file.

FoxPro behavior: Generates an error if a query is specified.

Action: If SET VIEW is running a query, replace the command with the DO command. If SET VIEW specifies a .VUE file, no action is necessary.

Comment: See the section titled "Using dBASE queries" as some queries may need to be modified before running properly in FoxPro.

Example: **dBASE**

```
SET VIEW TO myquery.qbe
```

FoxPro

```
DO myquery.qbe
```

SET("VIEW")XE "SET(\"VIEW\")"§
XE "SET (\"TRAP\")"§Level: 2

dBASE IV behavior: Returns ON or OFF.

FoxPro behavior: Generates an error.

Action:	Save the VIEW to a memory variable after using the SET VIEW command, and interrogate this variable instead of calling the SET("VIEW") function.

SET("WINDOW")XE "SET(\"WINDOW\")"§

Level:	2
dBASE IV behavior:	Returns the name of the default window for memo fields.
FoxPro behavior:	Generates an error.
Action:	Save the name of the window created by the SET WINDOW OF MEMO command in a variable and interrogate this variable.

SQLXE "SQL"§

Level:	1
dBASE IV behavior:	To work with dBASE data in SQL, you must create an SQL database and convert the dBASE databases to SQL tables (which are stored in a group of .DBF files). To activate and deactivate SQL mode, you must use SET SQL ON | OFF.
FoxPro behavior:	FoxPro supports CREATE CURSOR, CREATE TABLE, INSERT, and SELECT. These commands can be interspersed with standard Xbase commands and do not require a semicolon at the end. SET SQL ON and SET SQL OFF are not required and are ignored by FoxPro.
Action:	Complex dBASE IV SQL programs will not run in FoxPro and will have to be rewritten in FoxPro.

STOREXE "STORE"§XE "Arrays"§

Level:	4
dBASE IV behavior:	If you use the name of an array in a STORE command with no reference to array elements, dBASE IV releases the array from memory and creates a single memory variable with the same name.
FoxPro behavior:	If you use the name of an array in a STORE command with no reference to array elements, FoxPro assigns the specified value to every element in the array.
Action:	No action is required unless your program relies on STORE to transform an array to a memory variable.
	To prevent STORE in FoxPro from initializing all elements of an array with one value, SET COMPATIBLE DB4 on.

SUM()

XE "SUM()"§Level:	3
dBASE IV behavior:	The number specified in SET DECIMALS is the number of decimal places that are output by SUM.
FoxPro behavior:	The number of decimal places in the database structure for the field being summed determines the number of decimal places that are output.

Action:	SET COMPATIBLE DB4 on to use the number of decimal places specified in SET DECIMALS.

TAG()XE "TAG()"§

Level:	2
dBASE IV behavior:	dBASE allows the use of TAG() which returns active tag name or a null string if no tag is active.
FoxPro behavior:	Generates an error.
Action:	Replace with SYS(22).
Example:	**dBASE**
	TAG()
	FoxPro
	SYS(22)

TAGCOUNT() XE "TAGCOUNT() "§(version 1.5 only)

Level:	2
dBASE IV behavior:	The TAGCOUNT function returns the number of tags in an .MDX file.
FoxPro behavior:	Generates an error.
Action:	Use user-defined function.
Comment:	The Migration Kit includes a procedure library with a user-defined function called TAGCOUNT() that behaves exactly like the dBASE function TAGCOUNT. See the section titled "Calling user-defined functions" above.

TAGNO()XE "TAGNO()"§ (version 1.5 only)

Level:	2
dBASE IV behavior:	The TAGNO function returns the number of the tag specified as input.
FoxPro behavior:	Generates an error.
Action:	Use user-defined function.
Comment:	The Migration Kit includes a procedure library with a user-defined function called TAGNO() that behaves exactly like the dBASE function TAGNO. See the section titled "Calling user-defined functions" above.

Transaction processingXE "Transaction processing"§

Level:	1
dBASE IV behavior:	Although infrequently used, dBASE IV supports a transaction processing system. The transaction processing facilities include: BEGIN TRANSACTIONXE "BEGIN TRANSACTION"§ ... END TRANSACTIONXE "END TRANSACTION"§ command ROLLBACKXE "ROLLBACK"§ command RESETXE "RESET"§ command NOLOGXE "NOLOG"§ key word in the USE command COMPLETEDXE "COMPLETED"§ function ROLLBACK function ISMARKEDXE "ISMARKED"§ function
FoxPro behavior:	These commands and functions are not supported and generate errors.
Action:	Replace these functions and commands with equivalents supported by Novell Netware's Transaction Tracking System or with third-party software. FoxPro 2.5 for MS-DOS includes a library enabling applications to call the Netware TTS. See Appendix E for a list of products that offer transaction processing and a wealth of other network and security features.

UNIQUE()XE "UNIQUE()"§ (version 1.5 only)

Level:	2
dBASE IV behavior:	The UNIQUE function evaluates to .T. if an index was created with UNIQUE set ON or with the UNIQUE key word, or .F. otherwise.
FoxPro behavior:	Generates an error.
Action:	This function must be removed.

USEXE "USE"§ NOSAVEXE "NOSAVE"§, NOLOGXE "NOLOG"§, EXCLUSIVE, AGAINXE "EXCLUSIVE"§

Level: 2

dBASE IV behavior: a) NOSAVE causes dBASE IV to erase the database when it is closed.

b) The NOLOG key word suppresses the recording of changes in the transaction log if one is currently open.

c) By default SET EXCLUSIVE is off.

d) In dBASE IV 1.1, AGAIN assigns an alias that's the same as the letter of the work area. In dBASE IV 1.5, the assigned alias is an underscore character plus the number of the work area (e.g. _2).

FoxPro behavior: a) The NOSAVE key word is not supported and generates an error.

b) The NOLOG key word is not supported and generates an error.

c) By default, SET EXCLUSIVE is on in FoxPro.

d) AGAIN behaves like dBASE IV 1.1--the assigned alias is the letter of the work area into which the second copy of the database is opened.

Action: a) Use CREATE CURSOR to define a temporary database that is automatically removed from memory when it is closed. Or add a DELETE FILE command to erase the database after you close it.

b) The NOLOG key word is not supported and should be removed.

c) If your application needs shared use of a database, set EXCLUSIVE off.

d) Assign the same alias dBASE would, namely, an underscore character plus the number of the work area into which the database is opened.

USER

See Network functions.

WINDOW()

XE "WINDOW()"§Level: 2

dBASE IV behavior: Returns the name of the active window.

FoxPro behavior: Generates an error.

Action: Substitute either WONTOP (which returns the topmost window) or WOUTPUT (which returns the current output window).

Comment: FoxPro has a more advanced windowing model where it is possible for a window to be "active" in two senses--it can be on top of one or more windows, and/or it can be receiving output.

A UDF called WINDOW() in FOXPROC.UDF allows you to leave instances of the dBASE function WINDOW() unchanged if you wish. The UDF returns the value of the FoxPro function WONTOP(). See the section titled "Calling user-defined functions" above.

Example: **dBASE**

```
wind_name = WINDOW()
```

FoxPro

```
wind_name = WONTOP()
or
wind_name = WOUTPUT()
```

Migrating Clipper Summer '87 applications: an overview

Clipper Summer '87 applications consist of a number of files. Together, the Migration Kit, along with the native conversion capabilities of FoxPro, will make your data, indexes, format and program files work smoothly in FoxPro. dBASE III Plus reports (.FRMs) and labels (.LBLs) will often need to be recreated in FoxPro.

Steps to takeXE "Steps to Take"§

1. Create a backup copy of all your files. **Do not work on your original files!**
2. Bring your Clipper databases into FoxPro.
3. Convert .NTX indexes to FoxPro indexes.
4. If desired, use the File Converter to convert .FMTs and .PRGs to FoxPro screens.
5. Recreate reports and labels in FoxPro format.
6. "Extract" procedures and functions which aren't visible to FoxPro
7. Use the Program Analyzer to find and address areas of potential incompatibility.
8. Enjoy the speed and power of FoxPro!

A note on Clipper 5.x applications

The Migration Kit only supports Clipper Summer '87. However, unless Clipper 5.x applications fully embrace 5.x-specific, the Migration Kit may be of assistance. Clipper applications which make use of 5.x-specific features such as code blocks, user-defined commands, classes, new operators, replaceable database drivers, etc. will be very difficult to convert. Microsoft cannot offer support for this conversion process.

A list of Clipper 5.x syntax which is not supported by FoxPro can be found in Appendix F.

Using Clipper database, memo, and index files

Databases

Clipper Summer '87 and FoxPro use the same native file format (.DBF) so you can use your databases right away without any conversion. Type USE <database name> in the Command window or choose File...Open from the FoxPro menus.

Memo files

FoxPro and Clipper use different formats for storing memo fields. The XE "Memo files"§FoxPro format allows you to store an unlimited amount of data, and any kind of binary data, in a memo field. (You're limited by disk space, of course.) If you wish, FoxPro will maintain and write Clipper .DBT files, however you won't benefit from the advantages of FoxPro memo fields.

There are two ways to convert .DBT files to FoxPro style memo fields. The first way is to make a structure change in the data file. When Fox saves the new structure it creates the FoxPro style memo field. The second way is to copy the structure of the file to a new file and then append the records from the old file into the new file. Once the file is converted you won't be able to read it under Clipper, so make sure you do this on backup copies of the data files.

FoxPro can easily convert a memo field to back to .DBT format using the command COPY TO <database name> TYPE FOXPLUS.

Indexes

.NDX indexes

If you used the .NDX driver with your Clipper Summer '87 application, FoxPro can (natively, without the Migration Kit) recreate these indexes in .IDX format. FoxPro uses a more efficient indexing scheme that results in better performance as well as index sizes of one-half to one-third the size of .NDX indexes.

FoxPro will convert .NDXXE ".NDX"§ indexes when a USE command names .NDX indexes or a SET INDEX TO command that opens an .NDX index is issued.

.NTX indexes

If you have .NTX indexes, the Migration Kit will allow you to convert them to FoxPro indexes. To convert these indexes, you need to start the Migration Tools. See the section titled "Running the Migration Tools" on page 9. When the Migration Tools application is running, choose "Convert files..." from the Migration Tools menu. This will bring up the File Converter dialog.

µ §

The Convert Files dialog allows you to choose several .NTX files to convert at once.

To select an .NTX for conversion, either double-click on the filename, or highlight it and then press ENTER. Selected files will have an asterisk placed next to the filename on the left (or a check mark in FoxPro 2.5 for MS-DOS). You can select all the files in a directory for processing by clicking the Select All button, or you can start over by clicking the Clear All button. To cancel selection of a single file, double-click on the filename, or highlight it and press the ENTER key. The File Converter allows you to select a mix of files--some are dBASE IV files like. SCRs, .FRMs, and .LBLs. Others are .FMTs and .PRGs which can be converted to FoxPro screens. This is discussed in the section titled "Converting FMTs and PRGs."

To convert files in another directory, click the Directory button and move to a new directory. To convert files in multiple directories, first select and convert the files in one directory, and then select and convert files in another directory.

When you have finished selecting files, click the Process button. This brings up the following dialog:

µ §

Choose a database to associate with the converted index and an index type.

Choosing a database

To convert an .NTX index, you must associate it with the database on which it is based. Click the "Choose..." button. This brings up a file dialog box which allows you to browse your files to find the database.

Selecting an index type

The dialog shows which NTX is being converted. You will be presented with this dialog for each .NTX you selected to convert. The dialog offers you the option to either create .CDX or .IDX indexes. Indexes of the .IDX are functionally identical to .NTX indexes--both are single-entry indexes. If you wish to make the fewest changes to your program, you should convert .NTX indexes to .IDX indexes.

However, the .CDX format offers a number of advantages. A .CDX can contain multiple entries or "tags." In addition, .CDX indexes are automatically opened by FoxPro when you open the database associated with that index.

Creating .IDX indexes

To create an .IDX index, click on the .IDX radio button. Note that the Migration Tools automatically reads the index expression from the .NTX file into the field titled "Index expression". Once you have selected a database, you can name the .IDX file. By default, it will have the same name as the .NTX file, only with a .IDX extension.

When you have made these choices, click OK. The .NTX will be converted to an .IDX. The new file will be written to the same directory as the original .NTX.

Creating .CDX indexes

To create a .CDX index, click on the .CDX radio button. Whether you create a .CDX or .IDX, the Migration Tools reads the index expression from the .NTX file into the field titled "Index expression." After you choose a database, you can name the index "tag" anything you please as long as it conforms to the same rules as MS-DOS file names. The default will be the first ten characters of the index expression. Note that in some cases, this will not be a valid tag name and you will need to change it.

When you're satisfied with these settings, click OK. The .NTX will be converted to an .CDX. The new file will be written to the same directory as the associated database, unlike IDXs

If you convert additional .NTXs and you use the name of an existing .CDX, the Migration Tools will add new tags to the existing .CDX.

Functions in index expressions.

If your .NTX index expression includes a function, that function must be available to FoxPro. User-defined functions must be rewritten (at least temporarily) as .PRG files. These should then be placed in the same directory as MIGRATE.APP and the other Migration Kit files. If a function is not available, you will get an error and be instructed to make the file available to the converter.

If the function is a Clipper function not supported by FoxPro, it must be rewritten as a UDF in FoxPro or the index needs to be rebuilt in Clipper without the function and then converted using the Migration Tools. (The index expression is not editable in the .NTX converter dialog.) In some cases it may be easier to simply create the index from scratch in FoxPro.

Creating FoxPro screen files from .FMT and .PRG files

Files with @SAY commands can be converted to FoxPro screen files (.SCX). See the section titled "Converting FMTs and PRGs" on page 11.

Reports and Labels

FoxPro 2.5 for MS-DOS will run or, at your option, convert dBASE III Plus-style reports (.FRMs) and labels (.LBLs).

To run or modify dBASE III Plus-style reports (.FRMs) and labels (.LBLs), in FoxPro 2.5 for Windows, you will need to replace the Transporter program in your FoxPro for Windows directory. The Transporter in FoxPro 2.5 for Windows will not handle dBASE III Plus-style reports and labels. (If you have FoxPro 2.5a, the Transporter *will* handle these files.)

Note that by replacing your current Transporter, you will lose any changes you may have made to it. (To change this file, you would have had to have opened it like any other program file, make changes to the code, and save those changes. Merely using the Transporter will not change it.) If you are unsure whether to replace the old file, rename it or move it to a new location.

On the Migration Kit disk, there is a directory called NEWTPORT. In that directory is a file called TRANSPRT.PRG. Copy this file into the same directory as FOXPROW.EXE. Usually this will be a directory called \FOXPROW. You will now be able to convert and then run or modify dBASE III Plus .FRMs and .LBLs.

Unlike FoxPro 2.5 for MS-DOS, you will need to include the file extensions (.FRM or .LBL) when referring to these files in FoxPro for Windows.

If you hard-coded your reports (or labels) they should run like any other program, though you should use the Program Analyzer to check for any language incompatibilities. If you used a third party product such as R&R Reportwriter from Concentric, you can continue to use that product, as long as it supports FoxPro indexes, (which R&R does).

Note that the Migration Kit will convert dBASE IV-style reports (.FRMs) and labels

(.LBLs). However, the file converter will return an error if it encounters dBASE III Plus-style files.

Visibility of functions and procedures

In Clipper Summer '87, functions and procedures can be located in any .PRG and called from any .PRG. In FoxPro, called functions and procedures must either be in the same PRG from which they're called, or in the calling stack above the .PRG calling the function or procedure.

The Migration Tools provide a simple way to modify your program files to accommodate this difference. The Migration Tools can copy functions and procedures from a set of .PRGs to either a single procedure file or individual .PRGs. It can also make a copy of the original .PRGs minus the functions and procedures that were in them.

"Extracting" functions and procedures

To address the visibility issue, choose "Extract procs/funcs..." from the Migration Tools menu. You will then be presented with a dialog similar to those used elsewhere in the Migration Tools. Note that **your original program files are NOT changed by this process.** Procedures and functions are actually ***copied*** ("extracted") out of the original file and place into a new file.

Also note that if you have used the Program Analyzer on these files, you have two different copies of the files. One copy on disk, and the other in the analysis file (.EXP) memo field. Make sure you are "extracting" from the right files.

µ §

Choose the .PRG files from which to "extract" procedures and functions.

To select a file, either double-click on the filename, or highlight it and then press ENTER. Selected files will have an asterisk placed next to the filename on the left (or a check mark in FoxPro 2.5 for MS-DOS). You can select all the files in a directory for processing by clicking the Select All button, or you can start over by clicking the Clear All button. To cancel selection of a single file, double-click on the filename, or highlight it and press the ENTER key.

To convert files in another directory, click the Directory button and move to a new directory. To convert files in multiple directories, first select and convert the files in one directory, and then select and convert files in another directory.

When you have finished selecting files, click the Process button. This brings up the dialog pictured below.

µ §

This dialog allows you to place procedures and functions in a single procedure file or individual .PRG files.

Single procedure file or multiple .PRGs

Functions and procedures can be extracted to a single procedure file (a program file with the .PRG extension). Then, by adding the command SET PROCEDURE TO <procedure file name> to your startup program, all the procedures and functions in the procedure file will be available throughout your application.

Alternatively, functions and procedures can be extracted to individual .PRG files--one for each function and procedure and named like the original function or procedure (unless the name is greater than eight characters, in which case it is truncated).

By default, a single procedure file is created. Click on the radio button which reflects your choice.

Creating a single procedure file

If you choose to create a single procedure file, you need to choose a name and directory for that file. Click the "Choose..." button. Change to the directory you want the file to be written to and type in the name of the file. Then click the "Create" button. It is best to choose a directory other than the directory of the source files. The Migration Kit will always prompt you before overwriting any files, but choose a different directory anyway to avoid accidentally saying "yes" to an overwrite prompt.

Creating multiple .PRGs

If you want to create individual .PRG files, you need to select a destination directory for those files. Click the "Choose" button and navigate to the desired directory. Then click the "Select" button.

Creating new versions of the original files with the procedures and functions removed

If you would like a copy of the source .PRG files (the ones that are searched for procedures and functions) that has all procedures and functions removed, check the check box at the bottom of the screen. This will create a file of the same name as the original. The file will be written to the same directory as the procedure file if you've chosen that option, or the directory of the multiple .PRGs.

Again, the Migration Kit will always prompt you before overwriting any files. Don't be too quick to say "yes."

Addressing Clipper language compatibility issues

Overview

Although there are some incompatibilities between Clipper Summer '87 and FoxPro, most Clipper commands and functions work exactly the same way in FoxPro.

To take care of those commands and functions that don't work in FoxPro, first use the Program Analyzer to find potential compatibility problems. See the section titled "Using the Program Analyzer." After the Program Analyzer has created a database of potential issues, you can begin eliminating each in turn. After you have addressed these areas, you can then try running your application.

Each issue the Program Analyzer finds is documented in the section titled "Alphabetical list of potential Clipper issues." Each issue is described and an explanation offered on how programs can be modified so they perform the way you expect them to in FoxPro.

Main compatibility issues

Although the Program Analyzer might find many potential problems in your application, most all can be resolved, often by changing a single line of code. Also, you might find that few *types* of changes are required, although you might find many occurrences of the same few issues.

The amount of code you will have to change will depend upon how much you utilized features unique to Clipper. If your code is plain vanilla Xbase, you will only need to make minimal changes. In a number of cases, you may be removing code because FoxPro automatically provides some functionality that required coding or a third-party library in Clipper.

There are six areas where you will need to focus most of your attention. These are:

1. Function call syntax
2. ACHOICE(), DBEDIT(), and MEMOEDIT()
3. Error handling
4. Third party libraries
5. Arrays
6. Colors

Other compatibility issues

Besides the main compatibility issues listed above, there are a few more categories of issues that need to be addressed in migrating a Clipper Summer '87 application.

1. Windows-specific issues
2. Binary functions
3. SET commands
3. Keystrokes
4. Hard-coded file extensions

Function CallsXE "Function Calls"§

The bulk of the coding changes you will have to make will be the result of the different syntax used by Clipper and FoxPro to perform function calls.

Clipper allows functions to be called without assigning the return result to a variable. FoxPro requires that results be assigned to a variable or that an equal sign be placed in front of the function. The following example shows how a Clipper function made in this way would need to be modified.

Clipper:

```
FSEEK(handle,15,0)
```

FoxPro:

```
= FSEEK(handle,15,0)
or
dummy = FSEEK(handle,15,0)
or
? FSEEK(handle,15,0)
```

The Program Analyzer will flag instances of functions, including user-defined functions, made in the Clipper manner. The issue listed will be "Bad function call." As shown in the example, modify your code by placing an equal sign in front of the function, assigning the return value to a variable, or using the ? command. If FoxPro supports the function, it will then work properly.

ACHOICE(), DBEDIT(), and MEMOEDIT()

These three frequently used functions are not supported in FoxPro. However, FoxPro does offer a number of alternatives for ACHOICE(), and some almost direct equivalents exist for DBEDIT() and MEMOEDIT(). This information along with examples can be found in the section titled "Alphabetical list of potential Clipper issues".

Error handlingXE "Error handling"§

Error handling is one area where recoding will be necessary. Clipper error handling is based on the BEGIN SEQUENCE...[BREAK]...END structure. This allows the programmer to encapsulate a function call in a BEGIN SEQUENCE...END structure and then, from a function called, possibly nested several levels below, to return to the END statement via the BREAK keyword. This is a bit like a Longjump in C or a GoTo in Basic.

Example: **Clipper**

```
OldScreen=SAVESCREEN( 0, 0, 24, 79,)
CLEAR
@ 2, 12 SAY "Please standby while Printing!"
BEGIN SEQUENCE
    SET PRINT ON
    Foo()
END
SET PRINT OFF
CLEAR
@ 2, 12 SAY "Cannot print report at this time!"
@ 3, 12 SAY "Please try again later!"
RESTSCREEN( 0, 0, 24, 79, OldScreen )

FUNCTION Foo
    USE Test Exclusive
    IF NETERR()
        BREAK               && If you can't get exclusive return
                        && to END statement
    ENDIF
PrintIt()               && is a hard-coded report that
                        && requires exclusive file use
                        && It never executes if Break hit.
RETURN .T.
```

This is type of program flow is not possible in FoxPro unless you write some fairly elaborate code. It is recommended that instead of trying to simulate SEQUENCE...[BREAK]...END that you reengineer the error handling code using FoxPro's ON ERROR command.

Error handling FoxPro-style

Errors that can be fatal errors in Clipper are easily handled in FoxPro. Both languages provide some error handling internally. In Clipper Summer '87, error recovery was provided in EXTEND.LIB with source code provided in ERRORSYS.PRG. Many programmers added their own error handlers to provide for more graceful and trapable exits when errors occurred. FoxPro provides much more extensive error handling internally and traps for more types of errors than Clipper.

FoxPro error handling is based on the ON ERROR command which specifies a procedure to be executed when an error occurs. The error handling procedure is usually a lengthy case statement which traps for specific errors. If those errors are fatal it should cancel program execution (returning to FoxPro or to the operating system). If they are non-fatal, the procedure would report the error and return to the calling program.

In FoxPro, you might add the following code in the calling part of your program:

Example: **FoxPro**

```
ON ERROR DO Err_Hand WITH ERROR(), MESSAGE(), ;
MESSAGE(1), SYS(16), LINENO(), SYS(102), SYS(100), ;
SYS(101), LASTKEY(), ALIAS(), SYS(18), SYS(5), ;
SYS(12), SYS(6),      SYS(2003), WONTOP(), ;
SYS(2011), SYS(2018), SET("CURSOR")
```

In the case statement of the error handler you would have sections to handle specific errors. To use the same case as the Clipper error above, you would trap for File is in Use, or File is in use by another.

Assuming you defined a Message Window at the top of the Error handler, you could call it to tell the user the problem. For example:

Example: **FoxPro**

```
PROCEDURE Err_Hand
PARAMETERS m_error, m_message, m_message2, ;
m_progname, m_lineno, m_prtset, m_console, m_device, ;
m_lastkey, m_alias, m_curget, m_defdriv, m_mem, ;
m_print, m_curdir, m_wontop, m_lockstat, ;
m_winname, m_cursor

DO CASE
    CASE m_error= <error you want to trap>
        <handle the error>
    CASE m_error = 3 or ERROR() = 108
        Activate Window Message
        YN=" "
        @ 1,1 Say "File is already in use! Retry Y/N ? "
        Get YN
        Read
        If Upper(YN) $ "Y"
                Retry
        Else
                Set Print Off
                Return
        Endif
    CASE m_error= <some other error>
        <handle the error>
    ...
```

```
ENDCASE
```

FoxPro has many more built-in error messages than Clipper. You can easily handle them all with a well designed error handler. That error handler can also be a powerful debugging tool, providing you with information on the location and type of error that occurs.

Simulating BEGIN SEQUENCE...[BREAK]...END in FoxPro

When nesting of calls isn't too deep, BEGIN SEQUENCE...[BREAK]...ENDXE "BEGIN SEQUENCE...[BREAK]...END"§ can be simulated using DO WHILE loops. Remember, however, that the deeper the nesting, the more advisable reengineering using ON ERROR becomes.

In brief, this method consists of finding all occurrences of BREAK statements nested at lower levels in the program code and changing these routines and all routines that call them, all the way up to the original call in the BEGIN SEQUENCE block.

Example: **FoxPro**

```
DO WHILE .T.
    <statements>...
    IF break_cond
        EXIT
    ENDIF
<statements>...
ENDDO
 <recovery statements>...
```

If the routine is a procedure you must change it to a function call.

Example **Clipper**

```
DO MyRoutine [ WITH parm1, parm2, ... ]
```

FoxPro

```
IF !MyRoutine ( [ parm1, parm2 ...] )
     EXIT                                  &&If call is in original block
     * or RETURN .F.  &&If call is at lower level
ENDIF
```

If your routine is already a function, there are other changes that need to be made.

Example **Clipper**

```
MyRoutine ( [ parm1, parm2 , ...] )
```

FoxPro

```
PRIVATE RVAL
RVAL = .F.
IF !MyRoutine( @RVAL [ parm1, parm2, ... ] )
     EXIT                                  &&If ref is in BEGIN SEQ block
     * or RETURN .F.  &&If ref is at lower level
ENDIF
RETURN RVAL
```

Or

```
PRIVATE RVAL
RVAL  = .F.
x = MyRoutine( parm1, parm2,@RVAL)
IF !rval
     EXIT
     * or RETURN .F.
ENDIF
```

Then change the function as follows:

Example: **Clipper**

```
FUNCTION MyRoutine
PARAMETERS parm1, parm2,...
DO Whatever
BREAK
DO Whatever
RETURN whatever
```

FoxPro

```
FUNCTION MyRoutine
PARAMETERS RVAL, parm1, parm2,...
DO Whatever
```

```
RETURN .F.
DO Whatever
RVAL=whatever
RETURN .T.
```

Third party libraries

A large number of third party libraries and utilities were available for Clipper and if you have used one of them in your program, you will have to replace or remove those functions. Some libraries such as NetLib, CommTools, dGE are available for FoxPro as well. (dGE works with Clipper and FoxPro). One very commonly used library, OVERLAY(), can easily be replaced with an FoxPro utility, Foxswap.

Other libraries, such as Funcky, Grumpfish, Artful have no analog in FoxPro. Some of their functions can be replaced by writing your own UDFs. Getit, a popular library with Clipper Summer '87, allowed for nested reads and gets. That functionality is built into FoxPro. You can utilize up to five read levels. Also, a number of shareware offerings are available on the Foxforum on CompuServe. If you have Funcky II you can use the "C" functions and RUN them from within FoxPro.

Note: **the Program Analyzer only searches for Clipper Summer '87 syntax** that is incompatible with FoxPro. It will not find calls to third party libraries. These need to be found and removed (or reworked) manually, or they will generate compile or runtime errors.

Arrays

Clipper Summer '87 arrays are one-dimensional while FoxPro's arrays can be two-dimensional. Array handling functions are more extensive in FoxPro. (See page L2-27 in the FoxPro 2.5 Language Reference for a list.) Because FoxPro supports multi-dimension arrays, Clipper functions such as ADIR() and AFIELDS(), which take several arrays as arguments, require only one array in FoxPro.

Note that FoxPro does not support the AFILL() function, and that the TYPE() function will not operate on arrays like it does in Clipper. A full explanation of how to modify these four functions is in the section titled "Alphabetical list of potential Clipper incompatibilities."

Color

FoxPro handles colors very differently than Clipper. You will find it easier to use the FoxPro method of color management rather than trying to make your Clipper-style colors work in FoxPro.

FoxPro uses color schemes which are sets of ten color pairs. The color pairs in a scheme are used to color all the elements of the FoxPro user interface and the interface of any applications created in FoxPro. (Note that in FoxPro for Windows, the Windows Control Panel determines a number of colors in all Windows applications, including FoxPro.)

For instance, FoxPro uses a particular color pair of a scheme to show the difference between a push button that is enabled versus one that is disabled. No code needs to be written to affect the change in color. Similarly, the borders of a window change color depending on whether it is active or not. FoxPro handles that color change automatically.

FoxPro provides a number of predefined schemes. You can modify these or create your own from scratch. Color schemes are chosen with the SET COLOR SET command. To modify a color scheme, use the SET COLOR OF SCHEME or the CREATE COLOR SCHEME commands.

Example: **FoxPro:**

```
* Modify a color scheme and then load it
SET COLOR OF SCHEME 24 TO B/BG,W+/N, ;
BG+/BG,N/BG,N/BG,W+/GR,GR+/RB,N+/N,GR+/B,R+/B
SET COLOR SET TO SCHEME 24
```

For backwards compatibility, FoxPro also supports the SET('COLOR') function which returns current color settings and the SET COLOR TO command to define a new color setting. Together, these can offer the same functionality as the Clipper SETCOLOR() function.

Note that FoxPro 2.5 for MS-DOS comes with a very useful application called PROCOLOR which shows how to use FoxPro color schemes to your advantage.

Other compatibility issues

Windows Specific Problems

Printing

When printing hard-coded reports, some people have reported problems with the Windows printer drivers. The solution, if you are using ?/?? to print is to use the FoxPro commands, PRINTJOB/ENDPRINTJOB around your printing code. You also need to use the MS-DOS printer drivers in Windows. If you are printing with @ SAY, printing works correctly with SET DEVICE TO PRINTER, and even works with fonts. However, be careful with fonts where you must watch out for proper spacing.

ASCII character set.

If you used ASCII codes to print items to the screen, these calls will not work properly in the Windows product. The Windows version of FoxPro uses the ANSI character set, rather than the ASCII equivalent. There is a function in FoxPro for Windows, OEMTOANSI, which converts ASCII characters to their ANSI equivalents, but this will not help if the characters are line drawing characters. They have no equivalent in the ANSI character set.

Binary functions

Clipper supports five binary conversion functions not found in FoxPro: BIN2I, BIN2W, BIN2L, I2BIN, L2BIN. However, on the Migration Kit disk in the \BINFUNC directory, there is a file called BINFUNC.PRG with user-defined functions for each of these Clipper functions.

By using the SET PROCEDURE TO command to refer to this file or by placing the UDFs where they will be available in your program, you will not need to change these functions in your programs. They will execute as they did in Clipper.

SET commands

Many Clipper SET commands will accept a logical value. This will generate an error in FoxPro. Since SET commands will accept memory variables as arguments, there is no way for the Program Analyzer to always check if a logical value is being used with a SET command.

Unfortunately, you will need to find such instances manually or run the application in FoxPro and find them by encountering runtime errors. The SET commands which accept logical values are:

SET CENTURYXE "SET CENTURY"§	SET EXCLUSIVEXE "SET EXCLUSIVE"§
SET CONFIRMXE "SET CONFIRM"§	SET FIXEDXE "SET FIXED"§
SET CONSOLEXE "SET CONSOLE"§	SET INTENSITYXE "SET INTENSITY"§
SET CURSORXE "SET CURSOR"§	SET PRINTXE "SET PRINT"§
SET DATEXE "SET DATE"§	SET SCOREBOARDXE "SET SCOREBOARD"§
SET DELETEDXE "SET DELETED"§	SET SOFTSEEKXE "SET SOFTSEEK"§

SET DELIMITERSXE "SET DELIMITERS"§	SET UNIQUEXE "SET UNIQUE"§
SET ESCAPEXE "SET ESCAPE"§	SET WRAPXE "SET WRAP"§
SET EXACTXE "SET EXACT"§	

Keystrokes

FoxPro and Clipper Summer '87, in many cases, map keys to different values. The commands SET FUNCTION and SET KEY (an unsupported command), as well as the functions INKEY()XE "INKEY()"§, LASTKEY()XE "LASTKEY()"§ and NEXTKEY()XE "NEXTKEY()"§ (an unsupported function) could be affected by these differences. Consult Appendix G for a table comparing Clipper key return values with those of FoxPro and change your code accordingly.

The Program Analyzer flags all these functions and commands so you can identify the areas of your program that may need to be modified.

Alphabetical list of potential Clipper issues

Below is a list of the known compatibility issues in running Clipper Summer '87 programs in FoxPro. Each issue describes the behavior of a function or command in Clipper and in FoxPro. The Action section tells you what you should do to your program, and an example is usually included. Any other relevant information is placed in the comment section.

This information is also displayed in the Program Analyzer, with the exception of examples which are not displayed in the Program Analyzer.

For the sake of brevity and conciseness, no attempt is made to reproduce the documentation on these commands and functions. You can consult the Clipper and FoxPro documentation on specific commands and functions for further details.

Each issue is assigned one of four levels:

- **Level XE "Level "§1** commands and functions are those not supported in FoxPro. These commands and functions must be removed or replaced with FoxPro equivalents.
- **Level 2** commands and functions will generate errors in FoxPro but often have very close FoxPro equivalents.
- **Level 3** commands and functions will not generate errors but behave differently enough to merit attention.
- **Level 4** commands and functions will not generate errors and will rarely cause a problem in your program. Searching for Level 4 issues flags many lines of code which will usually work fine.

Level assignments, while somewhat arbitrary, are designed to give you a sense of the importance of an issue and the effort required to address it.

<u>== (comparison operator)XE "== (comparison operator)"§</u>

Level: 3

Clipper behavior: For character strings, returns TRUE if the strings are of exactly of the same length. Trailing spaces are ignored.

FoxPro behavior: Character strings must contain exactly the same characters, including spaces, for the operation to return TRUE.

Action: Use the RTRIM(), LTRIM() or ALLTRIM() functions to remove spaces from strings before comparing them.

Example: **Clipper**

```
IF string1==string2
    <execute code>
ENDIF
```

FoxPro

```
IF ALLTRIM(string1)==ALLTRIM(string2)
    <execute code>
ENDIF
```

<u>ACHOICE()XE "ACHOICE()"§</u>

Level: 1

Clipper behavior: ACHOICE() creates a menu based on two arrays. One of the arrays stores prompts, the other stores Logical values to dim or not dim the prompt array.

FoxPro behavior: Generates an error.

Action: Remove ACHOICE(). If ACHOICE() was used as a menu, recreate it using the FoxPro Menu Builder. If it was used as a picklist, replace it with a popup. For large lists, use a list array. For short lists, a popup with prompt fields should work fine.

Example: **Example 1: ACHOICE() used to create a menu**

Clipper:

```
DO WHILE .T.
menu_size=6

DECLARE cues[ menu_size ], msgs[ menu_size ]

cues[1] = " Orders "
cues[2] = " Styles "
cues[3] = " Tables "
cues[4] = " Totals "
cues[5] = " Reports "
cues[6] = " Get Totals "

msgs[1] = "    Enter, Edit, Delete and View orders."
msgs[2] = "    Enter, Edit, Delete and View Styles table.;     Add or
change items."
msgs[3] = "    Manage Tables."
msgs[4] = "    View Totals."
msgs[5] = "    Run any of the reports."
msgs[6] = "    Create Totals."

mchoice=ACHOICE( 1, 1, 2, 80, cues, msgs)

DO CASE
    CASE mchoice = 1
        THE_USUAL('ORD', .f., .f., .t.)
  CASE mchoice = 2
        THE_USUAL('STY')
  CASE mchoice = 3
        TABL_MENU()
  CASE mchoice = 4
        TOTL_MENU()
  CASE mchoice = 5
        RPT_MENU()
  CASE mchoice = 6
        GET_MENU()
  OTHERWISE
        Exit
ENDCASE
ENDDO
QUIT
```

FoxPro: replace an ACHOICE() menu with FoxPro menu

The above Clipper example would appear like this if created via the FoxPro Menu Builder. FoxPro would then generate the code for this menu for you.

µ §

To hand code these menus, the program would look like this:

```
SET SYSMENU TO
SET SYSMENU AUTOMATIC
DEFINE PAD Orders OF _MSYSMENU PROMPT "Orders"
DEFINE PAD Styles OF _MSYSMENU PROMPT "Styles"
DEFINE PAD Tables OF _MSYSMENU PROMPT "Tables"
DEFINE PAD Reports OF _MSYSMENU PROMPT "Reports"
DEFINE PAD GetTotal OF _MSYSMENU PROMPT "Get
Totals"
DEFINE PAD mQuit OF _MSYSMENU PROMPT "Quit"
ON SELECTION PAD Orders OF _MSYSMENU do ordmenu
ON SELECTION PAD Styles OF _MSYSMENU do stymenu
ON SELECTION PAD Tables OF _MSYSMENU do tablmenu
ON SELECTION PAD Reports OF _MSYSMENU do rptmenu
ON SELECTION PAD GetTotal OF _MSYSMENU do totmenu
ON SELECTION PAD mQuit OF _MSYSMENU QUIT
```

Example 2:   Clipper: ACHOICE() used to create a large picklist

```
select 0
use Vendor
DECLARE raTemp[RECCOUNT()]
FOR i = 1 TO RECCOUNT()
    raTemp[ i ] = Accoun_nam
    SKIP
NEXT
nChoice= ACHOICE(nTrow+1, nTcol+1, ;
    nBrow-1, nBcol-1, raTemp)
```

FoxPro popup with list array

```
DIMENSION Myarray[RECCOUNT()]
Select Vendor.Accoun_nam, recno() ;
from Profile into array Myarray
=Asort(Myarray)

@ 0,0 GET Vendor.Accoun_nam ;
     PICTURE "@&T" ;
     FROM Myarray ;
     SIZE 16,33 ;
     DEFAULT 1 ;
     WHEN check_prompt() ;
     COLOR SCHEME 2
```

Example 3: Clipper: ACHOICE() used to create a short picklist

See Example Above

FoxPro popup with prompt fields

```
Select Selcodes
Set Filter to Popname="ContType"
Go Top
m.type="Telephone"

DEFINE WINDOW ContactTy FROM 8, 2 TO 16,21 ;
      TITLE " Contact Type "  NOFLOAT ;
     NOCLOSE  NOMINIMIZE COLOR SCHEME 1

DEFINE POPUP Contact PROMPT FIELD pcue ;
SCROLL MARGIN MARK ""

ACTIVATE WINDOW ContactTy SAME
@ 0,0 GET m.type PICTURE "@&T" POPUP Contact ;
SIZE 7,18 DEFAULT " " COLOR SCHEME 2
READ CYCLE MODAL ;
     VALID Finishit()

RELEASE WINDOW ContactTy
RELEASE POPUPS Contact

m.cont_type=pcue
SELECT Calls
SHOW GET m.cont_type ENABLE

FUNCTION FinishIt     && Read level valid
```

ADIR()XE "ADIR()"§

Level: 2

Clipper behavior: ADIR() takes up to five arrays as arguments.

FoxPro behavior: ADIR() requires and accepts only one array.

Action: Change the syntax of the ADIR() function.

Comment: FoxPro supports multiple dimensions in arrays, obviating the need for several arrays.

Example: **Clipper**

```
DECLARE aName[10], aSize[10], aDate[10], aTime[10]
ADIR(aName,aSize,aDate,aTime)
```

FoxPro

```
=ADIR(Arin,"*.dbf")
```

AFIELDS()XE "AFIELDS()"§

Level: 2

Clipper behavior: AFIELDS() accepts up to four arrays as arguments.

FoxPro behavior: AFIELDS() requires and accepts only one array.

Action: Change the syntax of the AFIELDS() function.

Comment: FoxPro supports multiple dimensions in arrays, obviating the need for several arrays.

Example: **Clipper**

```
DECLARE aNames[FCOUNT()],aTypes[FCOUNT()] ;
aWidths[FCOUNT()], aDec[FCOUNT()]
=AFIELDS(aNames, aTypes, aWidths, aDec)
fld_name=aNames
fld_type=aTypes
fld_width=aWidths
fld_dec=aDec
```

FoxPro

```
=AFIELDS(Myarray)
fld_name=Myarray(1,1)
fld_type=Myarray(2,1)
fld_width=Myarray(3,1)
fld_dec=Myarray(4,1)
```

AFILL()XE "AFILL()"§

Level: 2

Clipper behavior: AFILL() fills an array with a chosen value.

FoxPro behavior: Generates an error.

Action: Remove AFILL(). Simply set the array name (without any subscript reference or brackets) equal to the desired value. Each array element will be set to that value.

Example: **Clipper**

```
AFILL(Myarray,"")
```

FoxPro

```
Myarray = ""
```

ALTD()XE "ALTD()"§

Level: 2

Clipper behavior: ALTD() invokes the Clipper debugger.

FoxPro behavior: Generates an error.

Action: Remove ALTD(). Replace with code to suspend program execution and invoke the Trace and/or Debug window.

Example: **Clipper**

```
Do WHILE !EOF()
    ALTD()
ENDDO
```

FoxPro

```
Do WHILE !EOF()
    ACTIVATE WINDOW Trace
    ACTIVATE WINDOW Debug
    SUSPEND
ENDDO
```

APPEND FROM FIELDSXE "APPEND FROM FIELDS"§

Level: 2

Clipper Behavior: The FIELDS clause precedes the FROM clause. In addition, a WHILE clause is supported.

FoxPro Behavior: Generates an error if the FIELDS clause precedes the FROM clause.

Action: Place the FROM clause before the FIELDS clause.

Example: **Clipper**

```
APPEND FIELDS Id, Firstname, Lastname FROM Myfile ;
FOR Id > 10 WHILE Lastname = "Smith"
```

FoxPro

```
APPEND FROM Myfile Fields Id, Firstname, Lastname ;
FOR Id > 10 AND Lastname = "Smith"
```

APPEND FROM WHILEXE "APPEND FROM WHILE"§

Level: 2

Clipper Behavior: Supports a WHILE clause.

FoxPro Behavior: Generates an error if a WHILE clause is included.

Action: Use a FOR clause instead of WHILE.

Example: **Clipper**

```
APPEND FIELDS Id, Firstname, Lastname FROM Myfile ;
        FOR Id > 10 WHILE Lastname = "Smith"
```

FoxPro

```
APPEND From Myfile Fields Id, Firstname, Lastname ;
FOR Id > 10 and LASTNAME = "Smith"
```

BEGIN SEQUENCE...[BREAK]...ENDXE "BEGIN SEQUENCE.. [BREAK]...END"§

Level 1

Clipper behavior: Defines a code sequence of statements used for error handling.

FoxPro behavior: Generates an error.

Action: There are two approaches to adapting Clipper error handling code. The recommended approach is to reengineer the error handling code using FoxPro's ON ERROR command. A second, potentially much more complicated approach, is to mimic BEGIN SEQUENCE...END with DO WHILE loops.

Example: See the section titled "Error handling"

BIN2I()XE "BIN2I()"§

Level: 1

Clipper behavior: Converts a character string formatted as a 16-bit signed integer to a Clipper numeric value.

FoxPro behavior: Generates an error.

Action: Use the UDF of the same name in the procedure file BINFUNC.PRG found in the \BINFUNC directory on the Migration Kit disk. This file contains UDFs for each of the five Clipper binary functions. Make these UDFs available to your program and there will be no need to change the Clipper function or code which relies on it.

BIN2L()XE "BIN2L()"§

Level: 1

Clipper behavior: Converts a character string formatted as a 32-bit signed integer to a Clipper numeric value.

FoxPro behavior: Generates an error.

Action: Use the UDF of the same name in the procedure file BINFUNC.PRG found in the \BINFUNC directory on the Migration Kit disk. This file contains UDFs for each of the five Clipper binary functions. Make these UDFs available to your program and there will be no need to change the Clipper function or code which relies on it.

BIN2W()XE "BIN2W()"§

Level: 1

Clipper behavior: Converts a character string formatted as a 16-bit unsigned integer to a Clipper numeric value.

FoxPro behavior:	Generates an error.
Action:	Use the UDF of the same name in the procedure file BINFUNC.PRG found in the \BINFUNC directory on the Migration Kit disk. This file contains UDFs for each of the five Clipper binary functions. Make these UDFs available to your program and there will be no need to change the Clipper function or code which relies on it.

CALLXE "CALL"§

Level:	1
Clipper behavior:	Executes separately compiled routines and programs. Allows passing of a list of expressions.
FoxPro behavior:	Executes a binary file that has been placed in memory with the LOAD command. CALL allows passing only a single character string. Clipper Summer '87 programs with the CALL command will generate an error in FoxPro.
Action:	Remove the function or recompile the routine to make it usable by FoxPro.
Comment:	Separately compiled and assembled routines for Clipper cannot be run in FoxPro as is. They would have to recompiled and turned into .BIN files or, using the FoxPro Library Construction Kit (LCK), into a .PLB or .FLL. For more information about the LCK, call 1-800-221-4679.

CANCELXE "CANCEL"§

Level:	3
Clipper behavior:	Returns to the operating system.
FoxPro behavior:	Returns to the last calling program or to FoxPro.
Action:	To return to the operating system, replace CANCEL with QUIT.

CLEARXE "CLEAR"§

Level:	2
Clipper behavior:	CLEAR supports a SCREEN clause which suppresses the automatic clearing of GETs.
FoxPro behavior:	A SCREEN clause generates an error.
Action:	Remove the SCREEN clause. By default, CLEAR in FoxPro will not clear GETs. To clear GETs, add the CLEAR GETS command.
Comment:	In Clipper, Clear Screen is used to clear the screen, place the cursor in the home position and clear any pending GETs. In FoxPro, the CLEAR GETS or CLEAR ALL command is required to clear the pending GETS. CLEAR alone, in FoxPro, clears the screen. If you have used the SCREEN option to clear GETs in your Clipper program, you must add a CLEAR GETS to your program to get the same effect in FoxPro.

Example: **Clipper**

```
CLEAR
```

FoxPro

Alphabetical List of Potential Clipper Issues

```
CLEAR
CLEAR GETS              && If your program expects GETs
to
                    && be cleared by the CLEAR command
```

COMMITXE "COMMIT"§

Level:	2
Clipper behavior:	COMMIT flushes all Clipper buffers to DOS and then performs a solid-disk write.
FoxPro behavior:	Generates an error.
Action:	Replace with the equivalent FoxPro command FLUSH.

Example: **Clipper**

```
COMMIT
```

FoxPro

```
FLUSH
```

DBEDIT()XE "DBEDIT()"§

Level:	1
Clipper behavior:	DBEDIT() displays and edits records from one or more work areas using a browse-style table layout that executes within a defined window area. DBEDIT() takes a custom keyboard handling routine which modifies its behavior.
FoxPro behavior:	Generates an error.
Action:	Remove the DBEDIT function and replace it with a BROWSE command.
Comment:	Customize the browse to match the previous behavior of DBEDIT(). You will not need to translate your keyboard handling routine because FoxPro handles the keystrokes for you. You use command line parameters to modify BROWSE behavior rather than writing a function call.

BROWSE includes a lot of built-in functionality that you had to write for yourself in Clipper. For example, in DBEDIT, to present read-only data, it required creating a key handling function that never allowed editing. In FoxPro, you simply issue the command line argument NOMODIFY after the BROWSE statement.

Example: **Clipper**

```
USE Customer
DECLARE field_list[3]
field_list[1] = "Branch"
field_list[2] = "Salesman"
field_list[3] = "Amount"
DBEDIT(4, 0, 22, 79, FIELD_LIST, "UserFunc")
```

FoxPro

```
DEFINE WINDOW Tbrowse FROM 4, 0 to 22, 79
USE Customer
BROWSE FIELDS Branch, Salesman, Amount ;
```

```
LAST NORMAL WINDOW TBROWSE;
TITLE "Customer"
```

<u>DBFILTER()XE "DBFILTER()"§</u>

Level:	2
Clipper behavior:	DBFILTER() returns as a character string the filter condition defined in the current work area.
FoxPro behavior:	Generates an error.
Action:	Replace DBFILTER() with the identically behaved FoxPro function FILTER().

Example: **Clipper**

```
filt_string=DBFILTER()
```

FoxPro

```
filt_string=FILTER()
```

<u>DBRELATION()XE "DBRELATION()"§</u>

Level:	2
Clipper behavior:	DBRELATION() returns a character string containing a relation expression.
FoxPro behavior:	Generates an error.
Action:	Replace DBRELATION() with its FoxPro equivalent, RELATION().

Example: **Clipper**

```
rel_string=DBRELATION()
```

FoxPro

```
rel_string=RELATION()
```

DBRSELECT()XE "DBRSELECT()"§

Level: 2

Clipper behavior: DBRSELECT() returns the work area number of a specified relation.

FoxPro behavior: Generates an error.

Action: Create a user-defined function called DBRSELECT() that mimics the behavior of its Clipper namesake. Include this function in your program or place it in a procedure library.

The UDF should use the FoxPro function TARGET() to return the proper value. Note you must be in the workarea in which you wish to find the relation..

Example: **Clipper**

```
MyArea=DBRSELECT(1)
```

FoxPro

```
MyArea=DBRSELECT(1)

FUNCTION DBRSELECT
PARAMETERS relnum
PRIVATE i, malias, relnum
mAlias=TARGET(relnum)
i=0
If !EMPTY(malias)
FOR i= 1 TO 255
    IF ALIAS(i) = malias
        RETURN i
    ENDIF
NEXT
    IF i = 226
        i =0
    ENDIF
ENDIF
RETURN i
```

DESCEND()XE "DESCEND()"§

Level: 2

Clipper behavior: DESCEND() allows creation of descending order indexes and perform seeks on descending indexes.

FoxPro behavior: Generates an error.

Action: Replace with the equivalent FoxPro command ASCENDING | DESCENDING in the INDEX ON expression. ASCENDING and DESCENDING can be toggled with the SET ORDER TO command in FoxPro.

Example: **Clipper**

```
INDEX ON DESCEND(Lastname) TO Lastname
SET INDEX TO Lastname
SEEK "Jones"
```

FoxPro

```
INDEX ON Lastname TO Lastname DESCENDING
SET INDEX TO Lastname
SET ORDER TO Tag Lastname Descend
SEEK "Jones"
```

DOSERROR()XE "DOSERROR()"§

Level: 2

Clipper behavior: DOSERROR() returns the number of the last MS-DOS error.

FoxPro behavior: Generates an error.

Action: Modify error handling routines to use the ON ERROR command.

Comment: See the section titled "Error handling."

Because Clipper applications are compiled, most errors that occur at runtime are critical errors requiring the user to be returned to the operating system. DOSERROR() in Clipper is used to determine what caused RUN commands to fail.

Example: **Clipper**

```
IF NETERR() .AND. model == "USE"
     RETURN .F.
ENDIF
BREAK_SEQ()

RETURN ERR_EXIT( ERR_LINE() + M->info + ", "  ;
+ M->_1 + IF( DOSERROR() > 0, ", DOS error "  ;
+ STR( DOSERROR()) , "" ))
```

FoxPro

```
ON ERROR DO FP25EROR WITH ERROR(),; MESSAGE(), ;
MESSAGE(1), SYS(16), LINENO(), SYS(102), ;
SYS(100), SYS(101), LASTKEY(), ;
ALIAS(), SYS(18), SYS(5), SYS(12), SYS(6),;
SYS(2003), WONTOP(), ;
SYS(2011), SYS(2018), SET("CURSOR")
```

ERRORLEVEL()XE "ERRORLEVEL()"§

Level:	2
Clipper behavior:	ERRORLEVEL() returns the current MS-DOS error level setting.
FoxPro behavior:	Generates an error.
Action:	ERRORLEVEL() was primarily used with the Clipper utility, SWITCH.EXE in situations where the programmer needed to run multiple executables. You do not need to check the MS-DOS error level when running the equivalent FoxPro command, FOXSWAP. Once you replace uses of SWITCH.EXE with FOXSWAP, this function can be removed.

EXTERNALXE "EXTERNAL"§

Level:	2
Clipper behavior:	EXTERNAL declares a symbol to the Clipper compiler.
FoxPro behavior:	Generates an error.
Comment:	In FoxPro, EXTERNAL is used to include files and to resolve undefined references in a FoxPro project. EXTERNAL is used only by the Project Manager and is ignored during program execution. EXTERNAL in Clipper is a command to the compiler that it will find references to the functions listed after EXTERNAL at link time, rather than at compile time.
Action:	Remove EXTERNAL references. FoxPro does not require the programmer to create compiler directives.
Comment:	The Project Manager handles resolving references in FoxPro. When an applications is built using the Project Manager, it will search for programs, functions and procedures which are called by the application. If it can find them, it will add them to the application. If it can't, it asks the developer to locate them.

FREAD()XE "FREAD()"§

Level: 2

Clipper behavior: FREAD() returns the number of bytes read, and reads those bytes into a memory variable which must be included as an argument.

FoxPro behavior: FREAD() returns the data actually read and accepts as an argument the number of bytes to read.

Action: Change the syntax of FREAD().

Example: **Clipper**

```
block = 128
buffer = SPACE(512)
handle = FOPEN("Test.txt")

IF FERROR() <> 0
    bytes = FREAD(handle, @buffer, block)
    IF bytes <> block
        ? "Error reading Test.txt"
    ENDIF
ENDIF
```

FoxPro

```
*Note: TEST.TXT must exist in this example
STORE FOPEN('test.txt') TO file_handle
STORE FSEEK(file_handle, 0, 2) TO ifp_size
IF ifp_size <= 0                        && Is File empty?
    WAIT WINDOW 'This file is empty!' NOWAIT
ELSE
    l_string = FREAD(file_handle, ifp_size)
ENDIF
```

FREADSTR()XE "FREADSTR()"§

Level: 2

Clipper behavior: FREADSTR() reads characters from a DOS file.

FoxPro behavior: Generates an error.

Action: Replace FREADSTR() with the equivalent FoxPro function FREAD().

Example: **Clipper**

```
buffer = FREADSTR(handle,16)
```

FoxPro

```
buffer = FREAD(handle,16)
```

HARDCR()XE "HARDCR()"§

Level:	2
Clipper behavior:	HARDCR() replaces all soft carriage returns with hard carriage returns.
FoxPro behavior:	Generates an error.
Action:	Remove the function HARDCR(). FoxPro supports word wrapping in memo fields, so the function should not be necessary.
Comment:	By default, word wrap is on in FoxPro. However, if you wish to strictly emulate HARDCR(), use the STRTAN() function.

Example: **Clipper**

```
memo_var=HARDCR(memo_var)
```

FoxPro

*Note this code should not be necessary!

```
memo_var=STRTRAN(memo_var,CHR(141),CHR(13)+CHR(10))
```

I2BIN()XE "I2BIN()"§

Level:	2
Clipper behavior:	I2BIN() converts an integer numeric data type to a two-byte character containing a 16-bit binary integer.
FoxPro behavior:	Generates an error.
Action:	Use the UDF of the same name in the procedure file BINFUNC.PRG found in the \BINFUNC directory on the Migration Kit disk. This file contains UDFs for each of the five Clipper binary functions. Make these UDFs available to your program and there will be no need to change the Clipper function or code which relies on it.

IFXE "IF"§

Level:	2
Clipper behavior:	An ELSEIF clause is supported.
FoxPro behavior:	Inclusion of ELSEIF generates an error.
Action:	Change the IF..ENDIF to a nested IF or CASE statement.
Comment:	For long ELSEIF constructs, a CASE statement works best.

Example: **Clipper**

```
IF A
    <do something>
ELSEIF B
    <do something else>
ELSE
    <do nothing>
ENDIF
```

FoxPro: nested if example

```
IF A
    <do something>
ELSE
    IF B
        <do something else>
    ELSE
        <do nothing>
    ENDIF
ENDIF
```

FoxPro: Case statement example

```
DO CASE
    CASE A
        <do something>
    CASE B
        <do something else>
    CASE <N>
        <do still another thing>
    OTHERWISE
        <do nothing>
ENDCASE
```

IF()XE "IF()"§

Level:	2
Clipper behavior:	Performs an "immediate if."
FoxPro behavior:	Generates an error.
Action:	Replace IF() with the equivalent FoxPro function IIF().

Example: **Clipper**

```
IF(Paid, SPACE(0), "Go get 'em")
```

FoxPro

```
IIF(Paid, SPACE(0), "Go get 'em")
```

INDEXEXT()XE "INDEXEXT()"§

Level:	2
Clipper behavior:	INDEXEXT() returns the type of index used in an application.
FoxPro behavior:	Generates an error.
Action:	Remove the function. Since FoxPro only uses its own index file formats, there is no need to check for index type.
Comment:	Note FoxPro *does* support .IDX indexes (a single-entry index similar to .NTXs and .NDXs) as well as .CDX indexes which can contain multiple tags. In addition, .CDXs are automatically opened and closed by FoxPro when the database with which they are associated is opened and closed.

INDEXKEY()XE "INDEXKEY()"§

Level:	2
Clipper behavior:	INDEXKEY() returns the key expression of an index.
FoxPro behavior:	Generates an error.
Action:	Replace INDEXKEY() with equivalent FoxPro function SYS(14) or use the KEY() function.

Example: **Clipper**

```
MyKey=INDEXKEY(1)
```

FoxPro

```
MyKey=KEY(1)
or
MyKey=SYS(14,1)
```

INDEXORD()XE "INDEXORD()"§

Level:	2
Clipper behavior:	INDEXORD() returns the index position number of the controlling index in the list of open index files.
FoxPro behavior:	Generates an error.
Action:	Replace INDEXORD() with the FoxPro function, SYS(21).
Comment:	Note that SYS(21) returns a string.

Example: **Clipper**

```
pos_num=INDEXORD(1)
```

FoxPro

```
pos_num=VAL(SYS(21))
```

INKEY()XE "INKEY()"§, LASTKEY()XE "LASTKEY()"§, SET FUNCTIONXE "SET FUNCTION"§

Level: 3

Clipper behavior: FoxPro and Clipper Summer '87, in many cases, map keys to different values.

FoxPro behavior: These functions and commands work the same way as in Clipper, but FoxPro key values may differ so unmodified Clipper programs may not behave as expected in FoxPro.

Action: Consult Appendix G for a table comparing Clipper key values with those of FoxPro and change your code accordingly.

ISPRINTER()XE "ISPRINTER()"§

Level: 2

Clipper behavior: ISPRINTER() determines if LPT1 is ready.

FoxPro behavior: Generates an error.

Action: Substitute the FoxPro function PRINTSTATUS().

Comment: It is not specific to LPT1, and will return the status of the active print device.

Example: **Clipper**

```
? ISPRINTER()
```

FoxPro

```
? PRINTSTATUS()
```

L2BIN()XE "L2BIN()"§

Level: 1

Clipper behavior: L2BIN() converts a Clipper numeric data type to a four-byte character string formatted as a 32-bit signed integer.

FoxPro behavior: Generates an error.

Action: Use the UDF of the same name in the procedure file BINFUNC.PRG found in the \BINFUNC directory on the Migration Kit disk. This file contains UDFs for each of the five Clipper binary functions. Make these UDFs available to your program and there will be no need to change the Clipper function or code which relies on it.

LASTKEY()

See INKEY().

LASTREC()XE "LASTREC()"§

Level: 2

Clipper behavior: LASTREC() returns the number of physical records in the active database.

FoxPro behavior: Generates an error.

Action: Replace this function with RECCOUNT().

Example: **Clipper**

```
num_recs=LASTREC()
```

FoxPro

```
num_recs=RECCOUNT()
```

MEMOEDIT()XE "MEMOEDIT()"§

Level: 2

Clipper behavior: MEMOEDIT() allows editing of memo fields.

FoxPro behavior: Generates an error.

Action: Replace MEMOEDIT() with MODIFY MEMO.

Comment: FoxPro's built-in memo editor handles editing, so you don't have to do all the programming that was necessary in Clipper. You can simply replace instances of MEMOEDIT with MODIFY MEMO, or mimic your Clipper program's behavior exactly, as in the example.

You can do some fancy things with memo fields and browses, that were hard if not impossible to do in Clipper. See example 2.

Also, memo fields can also be scattered with the FoxPro command SCATTER MEMVAR MEMO, making indirect reads easier to implement.

Example: **Clipper**

```
mNotes = MEMOEDIT(Notes,12,6,18,73,.T.)
```

FoxPro

```
DEFINE WINDOW TNotes FROM 12, 6 TO 18, 73
SET WINDOW OF MEMO TO Tnotes
MODIFY MEMO Notes
```

Example 2: **FoxPro**

```
*This sets up a BROWSE with a scrolling memo window
*side by side on the screen

DEFINE WINDOW Tbrow FROM 3,3 TO 15,15 ;
TITLE "Recent Calls"
DEFINE WINDOW Tcomment FROM 3,17 TO 15,78 ;
TITLE "Comments"
SELECT CALLS
SET FILTER TO Analyst_id=Anal_id AND ;
Contact_id=Rolodex->Contact_id
SET WINDOW OF MEMO TO TCOMMENTS
SET ORDER TO TAG CALLDATE
MODI MEMO COMMENTS NOWAIT
```

```
*Activate Window Tbrow
GO TOP
BROWSE FIELDS CALLDate LAST NORMAL ;
NOMENU WINDOW TBROW
DEACTIVATE WINDOW Tbrow
DEACTIVATE WINDOW TComments
RELEASE WINDOWS TBROW, TComments
SET WINDOW OF MEMO TO Close Memo Comments
SET FILTER TO ANALYST_ID=Anal_id
SET ORDER TO TAG ACCOUN_NAM
SELECT (cAlias)
```

MEMOLINE()XE "MEMOLINE()"§

Level: 2

Clipper behavior: MEMOLINE() extracts a formatted line of text from a character expression or memo field.

FoxPro behavior: Generates an error.

Action: In FoxPro you would do this in two steps. You would use ATLINE() or ATCLINE() to determine the line number, and then use MLINE() to return to actual line.

Example: **Clipper**

```
mNotes=MEMOLINE(Notes,79,3)
*Notes is the Memo Field
*79 is the line length
*3 is the line no
```

FoxPro

```
SET MEMOWIDTH TO 79
mNotes=MLINE(Notes,3)
```

MEMOREAD()XE "MEMOREAD()"§

Level: 2

Clipper behavior: MEMOREAD() returns the contents of a text file as a character string.

FoxPro behavior: Generates an error.

Action: Remove the command and replace with MODIFY FILE or MODIFY MEMO.

Example: **Clipper**

```
MEMOWRIT("Afile.txt.,Memoedit(Memoread("Afile.txt")))
```

FoxPro

```
MODIFY FILE Afile.txt
```

Example 2: **FoxPro**

```
* Use MODIFY FILE to edit a temporary file
COPY FILE MyFile.txt TO MyTemp.txt
MODIFY FILE MyTemp.txt
IF NOT LASTKEY()==27
    SET SAFETY OFF
    COPY FILE MyTemp.txt TO MyFile.txt
    SET SAFETY ON
ENDIF
ERASE MyTemp.txt
```

MEMORY()XE "MEMORY()"§

Level:	2
Clipper behavior:	MEMORY(0) returns the amount of available memory.
FoxPro behavior:	Returns an error.
Action:	Replace with either MEMORY(), SYS(12) or SYS(1001).
Comment:	In FoxPro for MS-DOS, MEMORY() and SYS(12) return the amount of memory below 640KB which is available to execute an external program. SYS(1001) returns the amount of memory available to the FoxPro memory manager including high memory between 640K and 1MB that has been made available to DOS. In FoxPro for Windows, MEMORY() always returns 640 and SYS(12) returns 655,360.

Example: **Clipper**

```
mem_avail=MEMORY(0)
```

FoxPro

```
mem_avail=MEMORY()
or
mem_avail=SYS(12)
or
mem_avail=SYS(1001)
```

MEMOTRAN()XE "MEMOTRAN()"§

Level:	2
Clipper behavior:	MEMOTRAN() replaces carriage return/line feed pairs.
FoxPro behavior:	Generates an error.
Action:	Remove the function. If desired, replace with STRTRAN().
Comment:	By default, word wrap is on in FoxPro. However, if you wish to strictly emulate MEMOTRAN(), use the STRTRAN() function.

Example: **Clipper**

```
*By default, MEMOTRAN replaces all hard carriage
*returns with semicolons, soft returns with spaces,
*and eliminates all line feeds
memo_var=MEMOTRAN(memo_var, "", "")
```

FoxPro

```
*Replace hard returns with semicolons
memo_var=STRTRAN(memo_var,CHR(13),";")
*Replace soft returns with spaces
memo_var=STRTRAN(memo_var,CHR(141)," ")
*Eliminate line feeds
memo_var=STRTRAN(memo_var,CHR(10))
```

Alphabetical List of Potential Clipper Issues

MEMOWRIT()XE "MEMOWRIT()"§

Level: 2

Clipper behavior: Writes a character string to a specified disk file.

FoxPro behavior: Generates an error.

Action: Replace with COPY FILE or COPY MEMO.

Example: **Clipper**

```
Memowrit(MyVar)
```

FoxPro

```
REPLACE Test.notes WITH MyVar
COPY MEMO Test.notes TO Myfile.txt
```

MLCOUNT()XE "MLCOUNT()"§

Level: 2

Clipper behavior: MLCOUNT() counts the number of word-wrapped lines in a character string or a memo field.

FoxPro behavior: Generates an error.

Action: Remove this function and replace it with the FoxPro equivalent MEMLINES()..

Example: **Clipper**

```
mNO=MLCOUNT(NOTES)
```

FoxPro

```
mNO=MEMLINES(NOTES)
```

MLPOS()XE "MLPOS()"§

Level: 2

Clipper behavior: MLPOS() determines the position of a specified line number in a character string or memo field.

FoxPro behavior: Generates an error.

Action: Replace this function with the FoxPro MLINE() function.

Comment: The system memory variable _MLINE stores the memo field offset for the MLINE() function.

Example **Clipper**

```
string = MEMOREAD("Temp.txt")
? MLPOS(string, 40, 5)
```

Foxpro

```
SET MEMOWIDTH to 40
string = Mydbf.Notes
= MLINE(string,5,_mline)
? _MLINE
```

Alphabetical List of Potential Clipper Issues

NETERR()XE "NETERR()"§

Level:	1
Clipper behavior:	NETERR() determines if a USE, USE...EXCLUSIVE, or APPEND BLANK has failed in a network environment.
FoxPro behavior:	Generates an error.
Action:	Remove this function call. You can trap errors resulting from USE and USE... EXCLUSIVE or an APPEND BLANK with an ON ERROR routine. See the section titled "Error handling."

NETNAME()XE "NETNAME()"§

Level:	1
Clipper behavior:	NETNAME() returns the workstation identification.
FoxPro behavior:	Generates an error.
Action:	Replace NETNAME() with SYS(0).
Comment:	SYS(0) returns the network computer name and number when FoxPro is running on a network. A machine number and name must first be assigned by the network software and the network shell must be loaded. On Novell networks, add the following to the system login script: MACHINE="%USER_ID,%P_STATION,%LOGIN_NAME" If FoxPro is not running on a network or a machine number and name haven't been assigned by the network, SYS(0) returns a string of 15 spaces, followed by a pound sign (#), space, and zero.

NEXTKEY()XE "NEXTKEY()"§

Level:	2
Clipper behavior:	NEXTKEY() reads the next keystroke without removing it from the keyboard buffer and returns an INKEY() value (or a 0 if the buffer is empty).
FoxPro behavior:	Generates an error.
Action:	Remove this function and replace it with LASTKEY().
Comment:	Note that key assignment values in Clipper and FoxPro may differ. See Appendix G for a list of these assignments and change your code if necessary. Another FoxPro function that may be useful in this context is CHRSAW() which returns TRUE if a character is present in the keyboard buffer without affecting the buffer's contents.

Example: **Clipper**

```
IF NEXTKEY()=27
```

```
    RETURN .F.
ENDIF
```

FoxPro

```
IF LASTKEY()=27
    RETURN .F.
ENDIF
```

PCOUNT()XE "PCOUNT()"§

Level: 2

Clipper behavior: PCOUNT() returns the number of actual parameters that have been passed to a procedure or user-defined function.

FoxPro behavior: Generates an error.

Action: Replace PCOUNT with the equivalent function PARAMETERS().

Example: **Clipper**

```
prm_count=PCOUNT()
```

FoxPro

```
prm_count=PARAMETERS()
```

PROCLINE()XE "PROCLINE()"§

Level: 2

Clipper behavior: PROCLINE() returns the source code line number from the beginning of the current program file.

FoxPro behavior: Generates an error.

Action: Remove the function and replace it with the FoxPro equivalent LINENO().

Example: **Clipper**

```
line_num=PROCLINE()
```

FoxPro

```
line_num=LINENO()
```

PROCNAME()XE "PROCNAME()"§

Level: 2

Clipper behavior: PROCNAME() returns the name of the current program or procedure.

FoxPro behavior: Generates an error.

Action: Replace with the FoxPro equivalent PROGRAM().

Example: **Clipper**

```
prog_name=PROCNAME()
```

FoxPro

```
prog_name=PROGRAM()
```

READEXIT()XE "READEXIT()"§

Level: 2

Clipper behavior: Toggles the up arrow and down arrow keys as READ exit keys.

FoxPro behavior: Generates an error.

Action: This command is unnecessary in FoxPro. If you want the up and down arrow keys to serve as exit keys in a read, create ON KEY LABEL commands to exit the read.

Example: **Clipper**

```
READEXIT(.T.)
```

FoxPro

```
ON KEY LABEL UPARROW CLEAR READ
ON KEY LABEL DNARROW CLEAR READ
```

READINSERT()XE "READINSERT()"§

Level: 2

Clipper behavior: Reports the current insert mode setting for READ and MEMOEDIT().

FoxPro behavior: Generates an error.

Action: Replace with INSMODE().

Example: **Clipper**

```
READINSERT( .T. )          &&Turns on insert
```

FoxPro

```
= INSMODE(.T.)             &&turns on insert
```

READVAR()XE "READVAR()"§

Level:	3
Clipper behavior:	READVAR() returns the name of the current GET or MENU variable.
FoxPro behavior:	Generates an error.
Action:	For GETs, replace READVAR() with VARREAD(). For menus, replace READVAR() with either PROMPT() or BAR().
Comment:	Clipper uses READVAR() primarily for debugging and as such is not appropriate in FoxPro. Use the Trace and Debug windows instead.

Example: **Clipper**

```
FUNCTION MyValid
xvar=READVAR()
xvalue=&READVAR()
SELECT Pcodes
SET ORDER TO &xvar
SEEK xvalue
IF !FOUND()
     RETURN .F.
ELSE
     RETURN .T.
ENDIF
```

FoxPro

```
FUNCTION MyValid
xvar=VARREAD()
xvalue=&READVAR()
SELECT Pcodes
SET ORDER TO (xvar)
SEEK xvalue
IF !FOUND()
     RETURN .F.
ELSE
     RETURN .T.
ENDIF
```

<u>RESTSCREEN()XE "RESTSCREEN()"§</u>

Level:	2
Clipper behavior:	RESTSCREEN() displays a previously saved screen region.
FoxPro behavior:	Generates an error.
Action:	Replace RESTSCREEN() with the equivalent FoxPro command RESTORE SCREEN.
Comment:	FoxPro has a sophisticated windowing manager that provides a more elegant mechanism for handling windows than saving and restoring screens. Commands such as DEFINE WINDOW, ACTIVATE WINDOW, SHOW WINDOW, and HIDE WINDOW provide an excellent alternative to a series of screen saves and restores.

Example: **Clipper**

```
RESTSCREEN( 0, 0, 24, 29, OldScreen)
```

FoxPro

```
RESTORE SCREEN FROM MyScreen
```

<u>SAVESCREEN()XE "SAVESCREEN()"§</u>

Level:	2
Clipper behavior:	SAVESCREEN() saves a specified screen area to be redisplayed later.
FoxPro behavior:	Generates an error.
Action:	Replace SAVESCREEN() with the equivalent FoxPro command, SAVE SCREEN.

Example: **Clipper**

```
OldScreen = SAVESCREEN(0,0,24,79)
```

FoxPro

```
SAVE SCREEN TO MyScreen
```

<u>SCROLL()XE "SCROLL()"§</u>

Level:	2
Clipper behavior:	SCROLL() designates a section of the screen to scroll up, down, or blank out.
FoxPro behavior:	Generates an error.
Action:	Replace with the FoxPro command SCROLL.

Example: **Clipper**

```
SCROLL( 4, 5, 21, 74, 0 )
```

FoxPro

```
SCROLL 4, 5, 21, 74, 0
```

SET CURSORXE "SET CURSOR"§

Level:	3
Clipper behavior:	SET CURSOR OFF turns off the cursor during screen painting.
FoxPro behavior:	SET CURSOR OFF prevents the cursor from being displayed during a pending @ ... GET, @ ... EDIT, WAIT or INKEY() statement.
Action:	No change required.
Comment:	You may wish to remove SET CURSOR OFF commands. By default, FoxPro doesn't display the cursor while painting the screen. However, if SET CURSOR is OFF, the cursor will be suppressed during pending GETs, EDITS, and WAITs.

SET FUNCTION
See INKEY().

SET KEYXE "SET KEY"§

Level:	2
Clipper behavior:	SET KEY executes a procedure when a designated key is pressed. The expression in SET KEY is the INKEY() value for a key stroke.
FoxPro behavior:	Generates an error.
Action:	Replace SET KEY with the ON KEY LABEL command. Change the value of the expression specified in SET KEY to the equivalent FoxPro key label.
Comment:	If there is no key label equivalent to the value used in SET KEY, use the FoxPro command ON KEY which, like SET KEY, takes a numeric argument (though the number for a particular key may be different). However, this command is included for backward compatibility only. See Appendix G for a table which shows the key value assignments in FoxPro and Clipper.

Example: **Clipper**

```
SET KEY -1 TO my_prog             && -1 is for the F2 key
```

FoxPro

```
ON KEY LABEL F2 DO my_prog   && Recommended
or
ON KEY = 316 DO my_prog      && 316 is for the F2 key
```

SET RELATIONXE "SET RELATION"§

Level:	2
Clipper behavior:	The TO keyword is repeated for each relation created in a single SET RELATION command.
FoxPro behavior:	The TO keyword is only used once, otherwise an error is generated.
Action:	Remove all the TO keywords except the first one.
Comment:	This is a very minor difference in syntax. These commands operate identically otherwise.

Example: **Clipper**

```
SET RELATION TO cust INTO invoice, TO state INTO state
```

FoxPro

```
SET RELATION TO cust INTO invoice, state INTO state
```

SET SOFTSEEKXE "SET SOFTSEEK"§

Level:	2
Clipper behavior:	SET SOFTSEEK toggles relative seeking on or off. A SOFTSEEK searches for the next higher key value when a SEEK fails.
FoxPro behavior:	Generates an error.
Action:	Remove the command. Replace it with the FoxPro equivalent SET NEAR.

Example: **Clipper**

```
SET SOFTSEEK ON
```

FoxPro

```
SET NEAR ON
```

SET WRAPXE "SET WRAP"§

Level:	2
Clipper behavior:	SET WRAP toggles wrapping in menus.
FoxPro behavior:	Generates an error.
Action:	Remove this command. It is unnecessary in FoxPro.
Comment:	FoxPro behaves as if SET WRAP is set on at all times.

SETCANCEL()XE "SETCANCEL()"§

Level: 2

Clipper behavior: SETCANCEL() toggles program termination with Alt-C, on or off.

FoxPro behavior: Generates an error.

Action: Remove the command. It is unnecessary in FoxPro. Most FoxPro programmers will either use ON ESCAPE or set a hotkey to bail out during application development.

Example: **Clipper**

```
SETCANCEL(.T.)     &&Turns ALTC on for termination.
```

FoxPro

```
ON KEY LABEL F12 SUSPEND
Or
SET ESCAPE ON
Or
ON ESCAPE SUSPEND
```

SETCOLOR()XE "SETCOLOR()"§

Level: 2

Clipper behavior: SETCOLOR() returns the current color setting or allows you to define a new color setting.

FoxPro behavior: Generates an error.

Action: Remove the command. Recoding using FoxPro color schemes is recommended. However, you may also substitute the SET('COLOR') function to get current color settings and use the SET COLOR TO command to define a new color setting.

Comment: See the section titled "Colors" for information on using FoxPro color schemes.

Example: **Clipper**

```
cur_col=SETCOLOR()
new_col=SETCOLOR("W+/B,W+/BG,N")
```

FoxPro

```
cur_col=SET('COLOR')
SET COLOR TO "W+/B,W+/BG,N"
```

SETPRC()XE "SETPRC()"§

Level: 2

Clipper behavior: SETPRC() sets the value of the internal PROW() and PCOL().

FoxPro behavior: Generates an error.

Action: Remove the function. Set the values of PROW() and PCOL() directly using the FoxPro system memory variables.

Example: **Clipper**

```
SETPRC(0,0)  && Resets row & column back to beginning
```

FoxPro

```
_LMARGIN=0
Or
EJECT
```

<u>TEXTXE "TEXT"§</u>

Level: 2

Clipper behavior: TEXT supports a TO PRINT and TO FILE option.

FoxPro behavior: The TO PRINT and TO FILE options generate errors.

Action: Remove TO PRINT and/or TO FILE. In FoxPro, you would SET CONSOLE OFF, and SET PRINTER TO to get the same effect.

Example: **Clipper**

```
TEXT TO PRINT
<Text to print on screen and to printer>
ENDTEXT
```

FoxPro

```
SET CONSOLE OFF
SET PRINTER TO LPT1      && or File MyFile.txt
TEXT
<Text to print to printer>
ENDTEXT
```

<u>TONE()XE "TONE()"§</u>

Level: 2

Clipper behavior: TONE() sounds a speaker tone for a specified frequency and duration.

FoxPro behavior: Generates an error.

Action: Replace with the SET BELL TO and ?? commands.

Example: **Clipper**

```
TONE(150,18)                  && sounds bell for one second
```

FoxPro

```
SET BELL TO 150, 1&& sounds bell for one second
?? CHR(7)
```

TYPE()XE "TYPE()"§

Level: 3

Clipper behavior: TYPE() can, among other values, return arrray, error syntactical and error indeterminate.

FoxPro behavior: TYPE() behaves like it does in Clipper Summer '87 except it does not return arrayXE "array"§, error syntactical and error indeterminate.

Action: If you need to test for an array, replace the TYPE() function with the ISARRAY() function below created by Malcolm Rubel.

Example: **Clipper**

```
TYPE('array_1')
```

FoxPro

```
=ISARRAY('array_1')

* Code sample from FoxPro 2.0 Power Tools
* by Malcolm Rubel, Performance Dynamics Associates
* Bantam Computer Books.  All rights reserved.
* Copyright (c) 1989, 90, 91
* Function returns TRUE if named variable is an array.

FUNCTION isarray
PARAMETERS var_name
PRIVATE var_name
IF type(var_name + '[1]') = 'U' .and. ;
   type(var_name + '[1,1]') = 'U'
  RETURN(.F.)
ELSE
  RETURN(.T.)
ENDIF
```

WORD()XE "WORD()"§

Level: 2

Clipper behavior: WORD() converts numeric parameters of the CALL command from type DOUBLE to type INT.

FoxPro behavior: Generates an error.

Action: Remove the function. This transformation is not necessary in FoxPro.

Using the Program Analyzer

What the Program Analyzer does

The Program Analyzer loads each selected program into a FoxPro memo field and then searches the code for dBASE IV or Clipper Summer '87 commands and functions that are not supported or might behave differently in FoxPro. (Your original files remain untouched on the disk.) The Program Analyzer creates a database of possible problem areas and their location in your program files (now stored in memo fields). After the Program Analyzer creates this database, you can select an issue and the Program Analyzer will open the memo field with the appropriate .PRG file in the text editor and place the cursor at the line of the potential problem.

Note that the Analyzer might flag code that will run properly in FoxPro. The Analyzer searches only for text strings--it does not check syntax or execute code. Some commands and functions are dependent on other parts of a program, and the Analyzer isn't designed to examine these dependencies.

The Program Analyzer doesn't modify files in any way. It only loads them into memo fields.

New Analysis

To analyze your programs, you must first have the Migration Tools software installed. (See the section titled "Migration Tools," above, for installation instructions.) Then DO MIGRATE.APP, either by choosing the Do command from the Program menu or typing DO MIGRATE.APP in the Command window.

A new pad called Migration Tools is added to the FoxPro menus. Choose New Analysis from the Migration Tools menu. This brings up a dialog box like the File Converter dialog box. Now, however, only .PRG files are displayed.

To select a file for analysis, either double-click on the filename, or highlight it and press the ENTER key. Selected files will have an asterisk placed next to the filename on the left. You can select all the files in a directory for analysis with the Select All button, or start over by clicking the Clear All button. To cancel selection of a single file, either double-click on the filename, or highlight it and press the ENTER key.

µ §

To create a new analysis, select .PRG files and then click the Process button.

To process files in another directory, click the Directory button and move to a new directory. To analyze files in multiple directories, first select and convert the files in one directory, and then select and convert files in another directory. Alternatively, you could place all the files in one directory.

Processing files

When you have finished selecting files, click the Process button. The Program Analyzer goes to work. First, all selected PRGs are copied into FoxPro memo fields. Then FoxPro searches the stored .PRGs for commands and functions that might cause problems when run in FoxPro. This will take some time depending on the number of files you process and the number of issues the Program Analyzer finds.

Note the Program Analyzer searches for full commands and abbreviated commands (e.g. activate menu and acti menu) but not partially abbreviated commands (e.g. activ menu).

Disk space requirements

By reading .PRGs into memo fields, the Program Analyzer effectively creates duplicate copies of your source files. Disk space requirements are roughly 1.2 times the size of all source files selected for analysis.

Settings

The Migration Kit supports both dBASE IV and Clipper Summer '87. If you have not selected the kind of program you wish to analyze in the Settings dialog, you will be prompted to choose either dBASE IV 1.5 or earlier, dBASE IV 2.0 or earlier, or Clipper Summer '87.

µ §

If you have not yet made a choice of the kind of application you are migrating, the Migration Tools will prompt you for a choice. The same dialog is available by choosing "Settings..." from the Migration Tools menu.

You may also bring up this dialog by choosing "Settings..." from the Migration Tools menu.   Choose the kind of application you're migrating and a sensitivity level, then click OK.

Sensitivity levels

The Analyzer divides possible problem areas into four categories:

- **Level XE "Level "§1** commands and functions are those not supported in FoxPro. These commands and functions must be removed or replaced with FoxPro equivalents.
- **Level 2** commands and functions will generate errors in FoxPro but often have very close FoxPro equivalents.
- **Level 3** commands and functions will not generate errors but behave differently enough to merit attention.
- **Level 4** commands and functions will not generate errors and will rarely cause a problem in your program. Searching for Level 4 issues flags many lines of code which will usually work fine.

By default, the Program Analyzer searches for Level 1, 2, and 3 issues. Note that setting the sensitivity in this dialog box is not the same as applying a filter based on sensitivity in the Program Analyzer window, discussed below.

All setup options are saved between sessions, and the previous settings are restored in subsequent sessions.

Open Analysis

This menu option allows you to return to previously created analyses. These are stored as database files with the extension .EXP. Note that new files cannot be added to analyses after they have been created.

The Program Analyzer interface

The Program Analyzer creates a database with a record for each instance of a potential problem. It stores the name of the command or function flagged and its location (line number) in the program file. This information is presented in the Program Analyzer.

µ §

The Program Analyzer displays a list of possible issues in your programs.

The upper left pane shows all the commands and functions that might be a problem if run in FoxPro. The leftmost column, beneath the heading "Level," shows the level of the command or function, 1, 2, 3, or 4. (See Filtering and sorting below for a description of these levels.)

The next column, under the "Program" heading, shows the line number and name of the program in which a possible problem was found. Finally, the name of the command or function is displayed beneath the heading "Issue."

Selecting an issue in the left pane brings up a corresponding explanation of that issue in the pane on the right side, under the "Overview" heading. This overview explains the behavior of the command or function in dBASE or Clipper Summer '87 (depending on the choice you make in the Settings dialog discussed above), the behavior in FoxPro, and the way to address problems that result from any differences.

The information in the Overview window is the same as that provided in this document in the sections titled with the alphabetical list of potential issues except that no examples are included in the Overview window.

Filtering and SortingXE "Order"§XE "View"§XE "Sensitivity"§

To help you resolve possible problems systematically, the Program Analyzer lets you sort issues in several ways: by program name, by issue (the name of the commands and functions), and by sensitivity (or level). The radio buttons under the heading "Order" determine the sort order.

You can also apply a filter to the issues and look only at the issues at each level: 1, 2, 3, or 4. Choose a radio button under the SensitivityXE "Sensitivity"§ heading to apply these filters. To filter issues based on whether they've been edited or not, select a radio button option beneath the "View" heading. (See the next section to learn about editing program files.)

Jumping to potential problem areas from the Program Analyzer

To go to the actual line where a problem might exist, double-click on the issue you want to address, or highlight the issue and press ENTER. The Program Analyzer automatically launches the program editor with the appropriate file and positions the cursor at the line number of the potential problem.

If you have made changes, pressing Ctrl+W or clicking the system box in the upper left corner of the window will save the changes back to the source code, and the Program Analyzer will reappear. Next to the number indicating the level of the issue, an asterisk will appear to indicate that the problem has been addressed.

To go back to the Program Analyzer without marking an issue as resolved and without saving any changes, press the ESC key while in the program editor.

If you have added or removed lines of code, future jumps from the Program Analyzer may not position the cursor at the exact line of the problem since line numbers will have changed since the analysis was created. Usually it will be only a few lines away, but if necessary, choose Find from the Edit menu and type in the name of the command or function and FoxPro will find it for you.

µ §

Double-clicking on an issue in the left pane launches the text editor and moves the cursor to the line appropriate line number.

Using your own XE "Text editor, using your own"§text editor

If you want to use your own text editor, you won't be able to jump to problems using the Program Analyzer in the manner described in the preceding section. Instead, print a report of all issues, and choose Close Tools from the Migration Tools menu to return to

FoxPro. Now you may launch your editor and go through the issues found in the report. dBASE users should find the FoxPro editor richer in features and significantly faster than the dBASE editor. In addition, most edits will be relatively minor, reducing the need for an external editor.

If you use your own editor, you will be editing the files directly on disk as opposed to the programs loaded into FoxPro memo fields. **Do not export program files from the Program Analyzer as these might overwrite the files you have edited!**

HelpXE "Help"§

For help on using the Program Analyzer, click the Help button or press the F1 key. To access regular FoxPro help on FoxPro commands and functions, click the Fox Help button. The Program Analyzer will attempt to find a FoxPro Help topic related to the issue currently highlighted in the lefthand pane.

ReportsXE "Reports, Program Analyzer"§

The Program Analyzer will create reports that can be viewed on-screen or printed. You can view the database of potential issues in any of three ways:

- Sorted by program filename and then by issue
- Sorted by program filename and then by line number
- Sorted by issue

These reports are designed to help you pinpoint aspects of your application that might require modifications. Additionally, the reports make it possible to utilize your own text-editing environment or other tools to perform global search-and-replace operations. You can create your own reports based on an analysis file (.EXP) using the FoxPro Report Writer. Note that any filters set using the Program Analyzer (such as viewing only edited issues) will affect report output.

Viewing the issues databaseXE "View Rules"§

To quickly find an explanation of how a particular command or function works in FoxPro and dBASE or Clipper, you can use the view the issues database. From the Migration Tools menu, choose View Issues. This brings up a window with all the dBASE IV or Clipper Summer '87 commands and functions that might cause a problem in FoxPro. The Program Analyzer searches program files for these key words. You can apply a filter to view only commands or only functions.

Selecting a particular command or function displays a corresponding explanation of how it behaves in dBASE or Clipper and in FoxPro. This is the same information displayed in the Program Analyzer and the same as the information in the section with the alphabetical list of potential issues.

After addressing the issues in the Program Analyzer list

After you have checked all the issues found by the Program Analyzer and made any changes necessary, you can export your programs back to regular .PRG files.

For analysis purposes, the Program Analyzer reads all selected .PRG files into individual memo fields. The Export .PRGXE " Export .PRG"§s bar on the Migration Tools menu allows you to write your programs back to regular .PRG files.

First, be sure you have an analysis open. Then click the Export .PRGs button.

Select the .PRGs you wish to export by double-clicking on the filename, or highlighting the filename and pressing ENTER. Use the directory button in the Export dialog box to select the output destination of the .PRG files. It is usually best to export .PRG files to a directory other than the one from which they were read. This will protect your original source code files. When you have selected the files and destination, click the Export button. If a file of the same name already exists, you will be warned that exporting will overwrite that file.

Make sure you really want to overwrite files before doing so! Overwritten files are lost permanently.

If you used your own editor to directly modify program files, do *not* export .PRGs. The .PRGs stored in the memo files will have none of the changes you made and could overwrite the files you did modify.

Note that any .PRGs that are exported remain in the analysis database and can be exported again. When you are finished using the Migration Tools, you can delete all files with the name of the analysis. There will be four files with the extensions .CDX, .FPT, .EXP, and .PXP. (These last two extensions are not regular FoxPro file extensions.) Be careful deleting files!

FoxPro Projects

You can run .PRGs directly by using the DO command as in dBASE. However, you should take advantage of the FoxPro Project Manager which keeps track of all the files in an application and will build a single application file. To use the Project Manager, create a project and add the name of the calling program to the project. Build your application and the Project Manager will take care of bringing in your other program files.

When building an application, if the FoxPro compiler encounters errors while compiling, it creates a file with the same name as the project file, but with a file extension of .err. That file will tell you if there is anything the Analyzer missed.

To create executable files, you will need the FoxPro Distribution Kit for either MS-DOS or Windows. With the Distribution Kit installed, the Project Manager becomes capable of creating stand-alone executable files.

Appendices

Appendix A: Effects of the SET COMPATIBLE command

Note that SET COMPATIBLE is off by default in FoxPro.

Command or Function	**SET COMPATIBLE**	**Effect**
All file-processing commands with a drive reference	DB4 ON	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.
	OFF	If a path is set and the command specifies a drive, the specified drive is searched, then the FoxPro path.
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 OFF, however, the file will be found and executed because the FoxPro path will also be searched.		
PARAMETERs passed by reference	DB4 ON	PARAMETERs that are passed by reference (SET UDFPARMS TO REFERENCE) remain available to the called procedure.

	OFF	PARAMETERs that are passed by reference (SET UDFPARMS TO REFERENCE) become "hidden" to the called procedure.
@...GET...RANGE	DB4 ON	The RANGE is always checked.
	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.
	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.
	OFF	Output that extends beyond the end of the screen is truncated.

Command or Function	**SET COMPATIBLE**	**Effect**
@...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.
	OFF	Output cannot 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 the PICTURE clause, the right-most digit in the PICTURE clause is rounded.
	OFF	When data is displayed using a PICTURE clause, the value is not rounded -- extra digits are truncated.
ACTIVATE SCREEN and ACTIVATE WINDOW	ON	When a screen or window is activated, the cursor position is set to 00,00.
	OFF	The cursor remains at its current position.
	ON	The default extension for the file will

APPEND MEMO		be .TXT if one is not specified.
	OFF	There is no default extension and none is used if one is not specified.
GO|GOTO (with TALK ON)	ON	FoxPro outputs a message that identifies the current work area alias and the record number positioned to.
	OFF	FoxPro does not output the message indicating position.
MENUs and POPUPs	ON	Popups are placed in the active 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.
	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 COMPATIBL E**	**Effect**
PLAY MACRO	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).
	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.
READ (with a VALID clause on a GET)	ON	If the ESC key 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.
	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 the input is invalid.
READ and TRANSFORM	ON	The value of the variable or expression will be rounded to the number of decimal places specified in the PICTURE clause.

(with a numeric PICTURE clause)		
	OFF	The value of the variable or expression will be truncated to fit the PICTURE clause.
READs (nested)	ON	When returning to a higher level READ, FoxPro does an implicit CLEAR GETS on the lower level before returning.
	OFF	When returning to a higher level READ, pending GETs in the lower level will remain pending.
RUN/!	ON	The cursor moves to row 24, column 0. All subsequent output begins displaying from this point.
	OFF	The cursor remains at its current position. All subsequent output begins displaying from this point.

Appendix A: Effects of the dBASE IV SET COMPATIBLE Command

Command or Function	**SET COMPATIBL E**	**Effect**
RUN/! (with STATUS ON)	ON	Output from the program that was RUN is scrolled up three lines before control returns to FoxPro.
	OFF	Output from the program that was RUN is scrolled up two lines before control returns to FoxPro.
SET COLOR TO	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.
	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	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.
	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).

Appendix A: Effects of the dBASE IV SET COMPATIBLE Command

SET MESSAGE	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.
	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.
SET PRINT TO <file>	ON	The output file is given a .PRT file extension, unless another extension is explicitly given.
	OFF	The output file is given a file extension only when one is explicitly given.
STORE	ON	STORE cannot be used to initialize all elements of an array.
	OFF	STORE may be used to initialize all elements of an array to a specified value.

Appendix A: Effects of the dBASE IV SET COMPATIBLE Command

Command or Function	SET COMPATIBLE	Effect
SUM	ON	The number specified in SET DECIMALS is the number of decimal places that will be output in the SUM.
	OFF	The number of decimal places specified in the database structure for the field being summed determines the number of decimal places that can be output.
INKEY() and LASTKEY()	ON	HOME and SHIFT+HOME key combination will return the value 26 and the CTRL+LEFT returns a value of 1.
	OFF	Refer to the table under INKEY() in the *Commands & Functions* manual for values.
LIKE()	ON	The pattern and target are both trimmed of trailing blanks before the comparison is made.
	OFF	the pattern and target are used "as is" and trailing blanks will be significant.
SELECT()	ON	SELECT() returns the number of the highest unused work area.

	OFF	SELECT() returns the number of the currently selected work area.
SYS(2001,'COLOR')	ON	The value returned is the current setting of SET COLOR ON | OFF.
	OFF	The value returned is the SET COLOR TO color pairs.

Appendix B: dBASE file types and what to do with them

FoxPro doesn't need all the files dBASE creates in order to run your application. Below is a list of all dBASE files, a description of their function, what to do with them, and where to look for assistance on using them in FoxPro.

File	Description	What to do	See
Database Files			Using dBASE database, memo, and index files
.DBF	Database file	Used by FoxPro. No action required.	
.DBT	Memo file	Automatically converted (and erased) by FoxPro when associated database is used.	
.MDX	Multiple tag index file	Ignored by FoxPro. FoxPro .CDX indexes are automatically created when the associated database is used in FoxPro.	
.NDX	Single expression index file	Ignored by FoxPro. Equivalent .IDX index is created if the .NDX is used in FoxPro.	

Format Files			
.SCR	Screen form object	Convert to FoxPro screen format (.SCX) using File Converter.	Converting Screens
.FMT	Generated format file	Used by FoxPro. (Modification might be needed.)	Modifying Format Files

.FMO	Compiled format file	Not required in FoxPro.	

Reports			
.FRM	Report form object	Convert to FoxPro report format (.FRX) using File Converter.	Converting Reports
.FRG	Generated report program	Not necessary if you convert the .FRM from which the .FRG is generated. Otherwise, run the .FRG in FoxPro.	Converting Reports
.FRO	Compiled report program	Not required in FoxPro.	

Queries			
.QBE	Program generated by dBASE QBE	Run in FoxPro after minor modifications.	Using dBASE Queries
.QBO	Compiled QBE program	Not required in FoxPro.	

Labels			
.LBL	Label form object	Convert to FoxPro label form object (.LBX) using File Converter.	Converting Labels
.LBG	Generated label program	Not necessary if you convert the .LBL file from which the .LBG was generated. Otherwise, run the .LBG in FoxPro.	
.LBO	Compiled label program	Not required in FoxPro.	

Programs			
.PRG	Program source code	Migrate to FoxPro using Program Analyzer.	Migrating .PRG files
.DBO	Compiled program	Not required in FoxPro.	
.PRS	SQL program source code	Rewrite in FoxPro.	

Applications Generated by ApplicatonsGenerator	Note on Applications Generator applications

.APP	Main application object	Not required in FoxPro.
.BAR	Bar menu object	Not required in FoxPro.
.BCH	Batch process object	Not required in FoxPro.
.FIL	Files list object	Not required in FoxPro.
.POP	Popup menu object	Not required in FoxPro.
.STR	Structure list object	Not required in FoxPro.
.VAL	Values list object	Not required in FoxPro.

Appendix C: dBASE error numbers that represent different errors in FoxPro

Error	dBASE IV	FoxPro
67	PROCEDUREs/FUNCTIONs nested too deep	Expression evaluator fault
95	Source does not correspond to the object	Statement not allowed in interactive mode
103	Command cannot be called	DO nesting too deep
130	Command not allowed in format files	Record is not locked
178	Function not found: <function>	MENU has not been activated
179	File not open in current work area: <file>	POPUP has not been activated
225	Right margin must be less than or equal to 255	"<name>" is not a memory variable
226	Tab stops must be in ascending order	"<name>" is not a file variable
232	ALIAS expression not in range	"<name>" is not an array

279	PROMPTS for this popup have already been defined	Menu/Popup was not pushed
411	Original memo cannot be larger than 64K	RUN/! command string too long
412	Not a valid disk drive: <drive>	Cannot locate COMSPEC environment variable
1001	Name longer than 10 characters	Feature not available
1002	Invalid character	I/O operation failure
1003	Missing end quotes for string	Free handle not found
1004	Undefined symbol	Use of invalid handle
1010	Name, constant, or expression expected	Area size exceeded during compaction
1011	Invalid constant	Area cannot contain handle
1012	Name expected (cannot be reserved word)	OS memory error
1098	Nested function not allowed	API function UserError() was called

1101	Only one column may be SELECTed in a subquery	Cannot open file "<file>"
1102	Index name already exists	Cannot create file
1103	Aggregate function not allowed in WHERE clause	Illegal seek offset
1104	Number of view columns does not match number of SELECT columns	File read error
1105	View column names must be specified	File write error
1106	INTO is not allowed in a view definition	Transaction in progress

Appendix C: dBASE error numbers that represent different errors in FoxPro

1108	View is not updatable : <view>	Picture too big
1111	View defined with GROUP BY cannot be used in a query with a GROUP BY clause	Invalid file descriptor
1112	Incorrect data type for arguments in dBASE function	File close error
1113	Incorrect number of arguments in dBASE function	File not open
1115	An illegal table is referenced in a subselect FROM clause: <table>	Invalid operation for CURSOR
1117	Views cannot be INDEXed : <view>	Wrong length key
1124	Cursor not open : <cursor name>	Key too big
1126	Different table name is specified in cursor declaration: <cursor>	Record too long
1127	INTO clause not allowed in cursor declaration	For/while need logical expressions
1130	Column name missing in AVG/MAX/MIN/SUM/COUNT	'Field' phrase not found
1134	Comparison operator or key word expectd	Variable must be in selected table

Appendix C: dBASE error numbers that represent different errors in FoxPro

1140	Catalog table(s) locked by another user: <catalog table>	FILTER expression too long
1141	Cannot DROP open database : <database>	Invalid index number
1145	Invalid character length	Must be a character, date or numeric key field
1147	SAVE TO TEMP clause not allowed	Target is already engaged in relation
1148	Number of SAVE TO TEMP columns does not match number of SELECT columns	Meaningless use of expression
1149	Column/field names must be specified in SAVE TO TEMP clause	No memory for buffer
1150	Invalid string operator	No memory for buffer
1151	Cannot ALTER views : <view name>	No memory for filename
1152	Invalid INSERT item	Cannot access selected table
1153	Numeric value too small	Attempt to move file to different device

Appendix C: dBASE error numbers that represent different errors in FoxPro

1154	No current row available for UPDATE or DELETE: <cursor>	Invalid buffpoint call
1155	Can't create subdirectory for new database	Invalid buffdirty call
1156	Name longer than 8 characters not allowed	Duplicate field names
1157	Invalid unary operator	Cannot update file
1160	This type of correlated subquery	Not enough disk space for "<filename>"
1161	Catalog tables are read-only : <table>	Too many records to BROWSE/EDIT in demo version
1162	Non-numeric array subscript	Procedure "<procedure>" not found
1163	Memory variable and dBASE function not allowed in SELECT clause with UNION	Browse table closed
1164	All SELECT columns must be inside an aggregate function	Browse structure changed
1201	Cannot GRANT or REVOKE a privilege to yourself	Too many names used
1202	Duplicate user ID	Program too large

Appendix C: dBASE error numbers that represent different errors in FoxPro

1217	Filename is same as existing synonym	Picture error in GET statement
1220	Internal SQL utility error #3	Invalid character in command
1221	Internal SQL utility error #4	Required clause not present in command
1223	Table not found in the SQL catalog tables	Invalid variable reference
1225	DBCHECK and RUNSTATS must be used with base tables	Must be a memory variable
1226	File is encrypted	Must be a file variable
1229	Catalog table Sysdbs does not exist	Too few arguments
1230	File is not legal dBASE/SQL : <file>	Too many arguments
1231	File not found in the current database	Missing operand
1245	Internal SQL error #29	Error in label field definition

Appendix C: dBASE error numbers that represent different errors in FoxPro

1282	File encryption error	Insufficient memory
1249	Unable to open the SYSDBS file in the SQL home directory	Too many READS in effect

Appendix D: Same errors with different numbers

dBASE	FoxPro	Error Message
29	1705	File not accessible
180	1621	PAD has not been DEFINEd for this MENU
207	19, 114	MDX file doesn't match database
244	1211	Missing ENDIF for previous IF command
245	1211	Missing ENDIF for previous IF/ELSE commands
247	1213	Missing ENDCASE for previous DO CASE command
249	1211	No previous IF to match this command
250	1213	No previous DO CASE to match this command
304	1649	No previous DO WHILE/SCAN/PRINTJOB to match this command
305	1649	No previous PRINTJOB to match this command

341	1214	Missing ENDTEXT for previous TEXT
1275	1282	Insufficient memory

Appendix E: Network and security libraries

NETWARE.PLB

Microsoft Corporation

Installed with FoxPro MS-DOS 2.5

Find NETWARE.PLB in the root of the directory where you have installed FoxPro 2.5. For documentation, see the help topic called Transaction Processing under General Topics. This library allows you to call Novell Netware Transactional Tracking System from FoxPro.

TTSAVAIL()	Checks for the presence of Novell Netware's TTS (Transactional Tracking System)
TTSATTRIB()	Specifies that a file be included in a transaction
BEGINTRAN()	Specifies the beginning of a transaction
COMMIT()	Writes changes to the files included in the transaction
RLLBACK()	Backs out changes made to the files included in the transaction

FPNET

Platinum Software
17 Thorburn Road
North Potomac, Maryland 20878
301-330-5118
Price: MS-DOS $295.00/Windows $295.00 or $395.00 for both

Versions Supported: 2.5

Bindery and Bindery Security functions

N_AddID()	Add a userid/password into the bindery

N_ChangeID()	Change an application password
N_DelID()	Delete a userid/password entry for an application
N_FindIDs()	List all application userids stored in the bindery
N_FullName()	Get a workstation's full name
N_Groups()	List all the groups to which a user belongs
N_IsEquiv()	Check security equivalences
N_IsMember()	Check group membership
N_LastLog()	Show a user's last login date and time
N_LoginID()	Get a workstation's login name
N_Members()	List all members of a group
N_ObjList()	List names of bindery objects

N_Operator()	List names of console operators
N_PQList()	List names of print queues

N_PropVal()	List the property values of a bindery object
N_SecEqs()	List all security equivalences
N_SList()	List names of file servers
N_UserList()	Show all users logged in with login time & conn no
N_VerId()	Verify a user's application password in the bindery
N_VerPswd()	Check if a user's network login password is correct

File, File Server, and Directory

N_AddTrust()	Assign directory trustee rights
N_ClrConn()	Clear a logical connection from a fileserver
N_Date()	Get the network date from the fileserver
N_DelTrust()	Remove a user's trustee rights to a directory
N_DownServ()	Bring a fileserver down

N_Flag()	Change/show attributes of sets of files
N_GetPath()	Get a path for a given drive letter
N_IsMap()	Check if a drive letter is mapped
N_IsOkLog()	Check if a fileserver has login enabled
N_Map()	Map a drive path to a drive letter
N_MapRem()	Remove the mapping of a network drive
N_NoLogin()	Disable all fileserver logins
N_NWRevDat()	Get NetWare revision date
N_NWVers()	Get NetWare version and revision
N_OkLogin()	Enable fileserver logins

N_RenDir()	Rename a directory
N_ServerID()	Connection ID number of a fileserver
N_ServFrID()	Get the fileserver name given a connection ID
N_ServName()	Get the fileserver name mapped to a drive
N_SetDate()	Set fileserver date
N_SetTime()	Change the fileserver time
N_Time()	Get the network time from the fileserver

Message Services

N_BFConGrp()	Broadcast a message from the console to a group
N_Brd2Cons()	Broadcast a message to the console
N_BrdFCons()	Broadcast a message from the console
N_Cast()	Set a workstation's broadcast mode
N_GetBMode()	Show a workstation's broadcast mode
N_GetBMsg()	Get a broadcast message stored in a file server
N_LogMsg()	Record a message in NET$LOG.MSG
N_SendMsg()	Send a message to a network user
N_SMsgGrp()	Send a message to a group

Printer Services

N_Capture()	Turn On/Off LPT redirection
N_GetPStat()	Show all printer redirection information

N_PJobList()	Show all info about print jobs in a queue
N_PJobMove()	Change queue position of a print job
N_PJobRem()	Remove a print job from a queue
N_SetPStat()	Set printer redirection info of LPT device

Semaphore and TTS

N_FlagTran()	Flag a file transactional or not
N_Semaphor()	Lock or unlock a semaphore string
N_SemClose()	Close a semaphore
N_SemOpen()	Open a semaphore
N_SemSig()	Signal a semaphore (unlock function)
N_SemVal()	Get the semaphore value

N_SemWait()	Wait on a semaphore (lock function)
N_TTSBeg()	Begin an explicit transaction
N_TTSEnd()	End an explicit or implicit transaction
N_TTSIsAvl()	Check if TTS is available
N_TTSOff()	Disable TTS on default server
N_TTSOn()	Enable TTS on default server
N_TTSZap()	Abort an implicit or explicit transaction

Workstation and Connection Services

N_Attach()	Attach to a fileserver
N_Detach()	Detach from a fileserver
N_DfServer()	Set the preferred fileserver
N_GetMATyp()	Get the workstation's machine type
N_GetOSTyp()	Get the workstation's OS name
N_GetOSVer()	Get the workstation's OS revision level
N_GetSMTyp()	Get the workstation's short machine type
N_IsAttach()	Is the workstation attached to a fileserver
N_IsLogged()	Show if a user is currently logged in
N_IsNWBoss()	Check workstation's console privileges
N_LocDrv()	Show the number of local drives

N_LogDate()	Get a workstation's login date
N_LogTime()	Get a workstation's login time
N_LoginFS()	Login to a fileserver
N_Logout()	Logout from all fileservers
N_LogoutFS()	Logout a user from a fileserver
N_StaAdr()	Get a workstation's physical node address
N_StaID()	Get a workstation's ID

Workstation Equipment Information

N_DosVer()	Get DOS version
N_DskFree()	Get free disk space on drive
N_DskSpace()	Get total capacity of drive

N_IsDisk()	Check for valid drive or disk on drive
N_IsMouse()	Check if mouse is active

GPLIB

George Sexton [73237,1665]

Price: Public domain (free). Download from CompuServe, FoxForum. The file is called GPLIB.ZIP

Currently supports FoxPro 2.0

AboutGPLIB()	Return version information for GPLIB
AMPM()	Function to return time as character string with AM or PM appended
ColdBoot()	Coldboot a computer
CtlAltShft()	Return true if Control, Alt or Shift key is held down
ElapseTime()	Function to return difference in minutes between two time strings
FileCount()	Function to return # of matching files in a directory
FindFirst()	Function to perform a DOS find first/find next
FKillAll()	Wild Card File Erase
Flag()	Set DOS file attributes
GenError()	Function to generate a FoxPro error

IsDiskIn()	Function to return whether a disk is in the floppy drive specified
Make_Dir()	Create directory
Math_Chip()	Return true if a 80x87 math coprocessor is installed
MReset()	Perform mouse reset & return number of buttons
Num_Serial()	Return Number of Serial Ports
N_AcctList()	Return list of user accounts from server
N_Attach()	Attach to a file server
N_Brd2Cons()	Broadcast message to server console
N_BrdFCons()	Broadcast message from server console
N_Capture()	Toggle capture on or off

N_Cast()	Set whether station will accept broadcast messages
N_Date()	Return File Server Date
N_Detach()	Detach from file server
N_DfServer()	Change default server
N_FlushLPT()	Flush capture of specified LPT port
N_FullName()	Return users full name
N_GetBMode()	Return current station broadcast mode
N_GetBMsg()	Retrieve a stored message from the file server
N_GetGroup()	Return list of groups from server
N_GetMaTyp()	Get Long Machine Type
N_GetOSTyp()	Return Operating System Type

N_GetOSVer()	Return Operating System Version
N_GetPath()	Return Path for a drive letter
N_GetPStat()	Get current Capture Settings
N_GetQList()	Return List of Queues from server
N_GetSList()	Return List of Servers from the file server
N_GetSMTyp()	Get Short Machine Type
N_GetWild()	Get List Of all bindery objects from server
N_IsAttach()	Return whether attached to a server
N_IsEquiv()	Return Security Equivalence
N_IsLogged()	Return whether specified user is logged into the network

N_IsMap()	Return whether drive is mapped to a NetWare file server
N_IsMember()	Return Group Membership
N_IsNWBoss()	Return whether workstation is a console operator
N_LastLog()	Return last login date & time for a user
N_LocDrv()	Return number of logical local drives
N_LogDate()	Return Login date
N_LoginFS()	Login to a file server
N_LoginID()	Return NetWare User ID
N_Logout()	Logout of all file servers
N_LogoutFS()	Logout of specified file server
N_LogTime()	Return Login Time

N_Map()	Map a drive
N_MapRem()	Remove A drive mapping
N_MaxConns()	Return Maximum NetWare connections
N_NoLogin()	Disable Login
N_NukeSta()	Clear Specified Connection Number
N_OkLogin()	Enable Login
N_PJobList()	Function to create an array containing print jobs for the specified queue
N_PJobMove()	Function to move print job within a print queue
N_PJobRem()	Function to remove a print job from a queue
N_PwdDate()	Return date users password expires

N_Semaphor()	Perform Netware semaphore locking
N_SendMsg()	Send a message to another network user
N_ServerID()	Return ID number of specified server
N_ServFrID()	Return server name from connection ID
N_ServName()	Return file server name for a mapped drive
N_SetPStat()	Modify Current capture settings
N_ShellVer()	Return Shell Version
N_StaAddr()	Return Physical station Address
N_StaID()	Return NetWare station number
N_SYSTime()	Synchronize stations clock to server
N_Time()	Return File Server Time

N_VerPwd()	Verify a users password
Rem_Dir()	Remove a directory
SetMLimit()	Set Mouse Limits
SetMPos()	Set Mouse Cursor Position
ShftPrtScr()	Invoke Hardware Interrupt 5 (Shift Printscreen)
TrimLen()	Return Trimmed Length of a String
UniqueName()	Return unique filename

NetLib

Pinnacle Publishing
PO Box 8099
Federal Way, Washington 98003,0099
800-788-1900
206-251-1900
Price: $299.00
Versions Supported: FoxPro 2.0

A version for FoxPro 2.5 for MS-DOS is due Q2 1993 and for FoxPro for Windows Q4 1993.

General network functions

N_ADDR()	Get physical network address
N_APPEND()	Append to .TXT or >SDF file
N_AREDIRECT() ***	Store redirected devices and npaths in array
N_SERVER()*	Return list of attached servers
N_ATTACH()*	Attach to specified server
N_CHECKF()	Get stations with RLOCK/FLOCK in file
N_CHECKR()	Get station with record locked
N_CHECKS()	Get stations with locked semaphore
N_CHECKU()	Get stations with .DBF in use
N_CONNECT()*	Initialize connection to station(s)
N_DATE()*	Get current date from server

N_DETACH()*	Detach from specified server
N_DISCON()*	Disconnect from station(s)
N_ENVLEN()	Return length of root environment string
N_ENVSIZ()	Return size of root environment space
N_ERROR()	Return error code from most recent NetLib function
N_FATTR()	Set or get file attribute
N_FCOPY()	Copy file
N_FDRIVE()	Get drive letter portion of filespec
N_FEXT()	Get extension portion of filespec
N_FMAP()	Get filespec associated with handle

N_FNAME()	Get base filename part of filespec
N_FPATH()	Get current directory or path portion of filespec
N_FSPEC()	Get full filespec for filename
N_FULLNAME() *	Get user full name
N_FVOL()	Get network volume name
N_ISEXCL()	.T. = current file is exclusive
N_ISFLOCK()	.T. = current file is locked
N_ISRLOCK()	.T. = specific record is locked
N_ISSLOCK()	.T. = semaphore is locked
N_JOURNAL()	Open file for journaling

N_LOGIN()*	Attach and log object into default server
N_LOGMSG()*	Write comment to system log
N_LOGOUT()*	Log out from specified server
N_MAPDRIVE()*	Create, query, or destroy a drive mapping
N_MLOCK()	Lock multiple records in file
N_MODENV()	Set new environment variable
N_NDX()	Get number and name of index file(s)
N_NETNAME()**	Return NetBIOS name table entry
N_PRTSC()	Print screen contents using INT5
N_READONLY()	Open file in read-only mode
N_READY()	.T. = on network

N_RECV()*	Receive message
N_REDIRECT()***	Set, get or cancel device redirection
N_RESTATTR()	Restore screen attributes
N_RIGHTS()*	Get driectory rights
N_SAVEATTR()	Save screen attributes
N_SECONDS()	Get seconds since 12 a.m. from server
N_SEND()*	Send message to station(s)
N_SERIAL()*	Return serial number of server
N_SERVER()*	Get or set server
N_SERVNUM()*	Return connection number of server

N_SETLOG()*	Enable or disable login to specified server
N_SETTIME()	End wait state, call event procedure
N_SLOCK()	Lock semaphore
N_SOFTSCR()	Print screen by software emulation
N_STAMAX()	Get highest station number
N_STANUM()	Get current station number
N_SUNLOCK()	Unlock semaphore
N_TIME()*	Get time from server
N_USE()	Open database file
N_VERSION()*	Return server version number and other statistics
N_WHERE()	Get station where user is logged in

N_WHOAMI()	Get user ID

Printing functions

N_BANNER()*	Set print banner page
N_CLASS()*	Set document print class
N_COPIES()*	Set print copy count
N_PRINTER()	Get target printer number
N_SPLFRM()*	Set print form name
N_SPLLPT()*	Set LPT capture number
N_SPLQUE()*	Set spool capture queue
N_SPLSRV()*	Set print server
N_SPLTABS()*	Set tab expansion count

N_SPLTMO()*	Set print timeout
N_SPOOL()	Start or stop spool capture

Encryption Functions

N_CODELVL()	Get encryption status
N_DECODE()	Decrypt file
N_DECODEST()	Decrypt string
N_ENCODE()	Encrypt file
N_ENCODEST()	Encryp string
N_SETKEY()	Set encryption key

Netware Bindery functions

N_B_CREATE()*	Create bindery object
N_B_DEL()*	Delete binder object

N_B_ID()*	Return Binder ID of specified object
N_B_ISMEMBER()*	.T. = member object is part of set
N_B_LINK()*	Connect object to set
N_B_MEMBERS()*	Store names of set members in specified array
N_B_NAME()*	Return name of specified Bindery object
N_B_PASSWORD()*	.T. = password is valid
N_B_PRCREATE()*	Create Bindery property
N_B_PRSCAN()*	Store names of all properties attached to specified array
N_B_PRREAD()*	Return value of specified property

N_B_PRTYPE()*	Return type of specified property
N_B_PRWRITE()*	Modify Bindery item property
N_B_SCAN()*	Scan Bindery for matching objects and place names into array
N_B_TYPE()*	Return type of specified Bindery object
N_B_UNLINK()*	Disconnect object from set

*Novell Netware only

**NetBIOS only

***NetBIOS and Banyan VINES only

Appendix F: Clipper 5.x incompatabilities

Preprocessor directives

#command
#define
#error
#ifdef
#ifndef
#include
#stdout
#undef
#xcommand

Operators

:=
++
--

Variable Types

STATIC
LOCAL

Data Types

Codeblocks
NIL

Classes

Error
Get
TBrowse
TBColumn

Commands

DELETE TAG
GO
INDEX
SEEK
SET INDEX
SET ORDER

Functions

AEVAL()
ALERT()
ARRAY()
ATAIL()
BREAK()
BROWSE()
DBAPPEND()
DBAPPEND()
DBCLEARFIL()
DBCLEARIND()
DBCLEARREL()
DBCLOSEALL()
DBCLOSEAREA()
DBCOMMIT()
DBCOMMITALL()
DBCREATE()
DBCREATEIND()
DBDELETE()
DBEVAL()
DBFILTER()
DBGOBOTTOM()
DBGOTO()
DBGOTO()
DBGOTOP()
DBRECALL()
DBREINDEX()
DBRELATION()
DBRLOCK()
DBRLOCKLIST()
DBRSELECT()
DBRUNLOCK()
DBSEEK()
DBSELECTAR()
DBSETDRIVER()
DBSETFILTER()
DBSETINDEX()
DBSETINDEX()
DBSETORDER()
DBSETRELAT()
DBSKIP()
DBSTRUCT()
DBUNLOCK()
DBUNLOCKALL()
DBUSEAREA()
DEVOUT()
DEVOUTPICT()
DEVPOS()
DISPBEGIN()
DISPBOX()
DISPCOUNT()
DISPEND()
DISPOUT()
ERRORBLOCK()
EVAL()
EXP()
FIELDBLOCK()
FIELDGET()
FIELDNAME()
FIELDPOS()
FIELDPUT()
FIELDWBLOCK()
HEADER()
MEMVARBLOCK()
NOSNOW()
ORDBAGEXT()
ORDBAGNAME()
ORDCREATE()
ORDDESTROY()
ORDFOR()
ORDKEY()
ORDLISTADD()
ORDLISTCLEAR()
ORDLISTREBUI()
ORDNAME()
ORDNUMBER()
ORDSETFOCUS()
OS()
OUTERR()
OUTSTD()
QOUT()
RDDLIST()
RDDNAME()
RDDSETDEFAULT()
READMODAL()
RECNO()
SETBLINK()
SETCANCEL()
SETCOLOR()
SETCURSOR()
SETKEY()

SETMODE()
SETPOS()
SETPRC()
VALTYPE()
VERSION()

GET System

GETACTIVE()
GETAPPLYKEY()
GETDOSETKEY()
GETPOSTVALIDATE()
GETPREVALIDATE()
GETREADER()
READFORMAT()
READKILL()
READUPDATED()

Appendix G: Key assignments

FoxPro ON KEY LABEL KEY assignments

Keycap identification	**Key label**
Left arrow	LEFTARROW
Right arrow	RIGHTARROW
Up arrow	UPARROW
Down arrow	DNARROW
Home	HOME
End	END
PgUp	PGUP
PgDn	PGDN
Del	DEL
Backspace	BACKSPACE
Spacebar	SPACEBAR

Ins	INS
Tab	TAB
Shift Tab	BACKTAB
Enter	ENTER
{	LBRACE
}	RBRACE
F1 to F12	F1, F2 ...
Ctrl+F1 to Ctrl+F12	Ctrl+F1, Ctrl+F2 ...
Shift+F1 to Shift+F9	Shift+F1, Shift+F9...
Shift+F11, Shift+F12	Shift+F11 ...
Alt+F1 to Alt+F12	Alt+F1, Alt+F2 ...
Alt+0 to Alt-9	Alt+0, Alt+1 ...
Alt+A to Alt+Z	Alt+A, Alt+B ...
Alt+PgUp	Alt+PGUP

Alt+PgDn	Alt+PGDN
Ctrl+left arrow	Ctrl+LEFTARROW
Ctrl+right arrow	Ctrl+RIGHTARROW
Ctrl+Home	Ctrl+HOME
Ctrl+End	Ctrl+END
Ctrl+PgUp	Ctrl+PGUP
Ctrl+PgDn	Ctrl+PGDN
Ctrl+A to Ctrl+Z	Ctrl+A, Ctrl+B ...
Right Mouse	RIGHTMOUSE
Left Mouse	LEFTMOUSE
Mouse	MOUSE
Escape	ESC

INKEY() codes for FoxPro, dBASE and Clipper Summer '87

Function and cursor movement keys

	FoxPro	dBASE	Clipper S'87		FoxPro	dBASE	Clipper S'87
Ctrl-End	373	23	23	Backspace	127	127	8
Ctrl-Home	375	26	29	Ctrl-Backsp	NA	-401	127
Ctrl-Leftarrow	371	1	26	Delete	339	NA	7
Ctrl-PgDn	374	30	30	Downarrow	336	24	24
Ctrl-PgUp	388	31	31	End	335	2	6
Ctrl-Return	10	-402	10	Home	327	26	1
Ctrl-Rightarrow	372	6	2	Insert	338	NA	22
Return/Enter	13	13	13	Leftarrow	331	19	19
Esc	27	27	27	PgDn	337	3	3

Appendix G: Key assignments

F1	315	28	28	PgUp	329	18	18
F2	316	-1	-1	Rightarrow	333	4	4
F3	317	-2	-2	Shift-Tab	15	-400	271
F4	318	-3	-3	Tab	9	9	9
F5	319	-4	-4	Uparrow	328	5	5
F6	320	-5	-5				
F7	321	-6	-6				
F8	322	-7	-7				
F9	323	-8	-8				
F10	324	-9	-9				

Appendix G: Key assignments

Alt and Ctrl keys

	FoxPro	**dBASE**	**Clipper S'87**		**FoxPro**	**dBASE**	**Clipper S'87**
Alt-A	286	-435	30	Ctrl-A	1	1	1
Alt-B	304	-434	48	Ctrl-B	2	2	2
Alt-C	302	-433	46	Ctrl-C	3	3	3
Alt-D	288	-432	32	Ctrl-D	4	4	4
Alt-E	274	-431	18	Ctrl-E	5	5	5
Alt-F	289	-430	33	Ctrl-F	6	6	6
Alt-G	290	-429	34	Ctrl-G	7	7	7
Alt-H	291	-428	35	Ctrl-H	8	8	8
Alt-I	279	-427	23	Ctrl-I	9	9	9
Alt-J	292	-426	36	Ctrl-J	10	10	10

Appendix G: Key assignments

Alt-K	293	-425	37	Ctrl-K	11	11	11
Alt-L	294	-424	38	Ctrl-L	12	12	12
Alt-M	306	-423	50	Ctrl-M	13	13	13
Alt-N	305	-422	49	Ctrl-N	14	14	14
Alt-O	280	-421	24	Ctrl-O	15	15	15
Alt-P	281	-420	25	Ctrl-P	16	16	16
Alt-Q	272	-419	16	Ctrl-Q	17	17	17
Alt-R	275	-418	19	Ctrl-R	18	18	18
Alt-S	287	-417	31	Ctrl-S	19	19	19
Alt-T	276	-416	20	Ctrl-T	20	20	20

Appendix G: Key assignments

Alt-U	278	-415	22	Ctrl-U	21	21	21
Alt-V	303	-414	47	Ctrl-V	22	22	22
Alt-W	273	-413	17	Ctrl-W	23	23	23
Alt-X	301	-412	45	Ctrl-X	24	24	24
Alt-Y	277	-411	21	Ctrl-Y	25	25	25
Alt-Z	300	-410	44	Ctrl-Z	26	26	26

Shift, Alt, and Ctrl-Function keys

	FoxPro	**dBASE**	**Clipper S'87**
Alt-F1	104	NA	-30
Alt-F2	105	NA	-31
Alt-F3	106	NA	-32
Alt-F4	107	NA	-33
Alt-F5	108	NA	-34
Alt-F6	109	NA	-35
Alt-F7	110	NA	-36
Alt-F8	111	NA	-37
Alt-F9	112	NA	-38
Alt-F10	113	NA	-39

Ctrl-F1	94	-10	-20
Ctrl-F2	95	-11	-21
Ctrl-F3	96	-12	-22
Ctrl-F4	97	-13	-23
Ctrl-F5	98	-14	-24
Ctrl-F6	99	-15	-25
Ctrl-F7	100	-16	-26
Ctrl-F8	101	-17	-27
Ctrl-F9	102	-18	-28

Ctrl-F10	103	-19	-29
Shift-F1	84	-20	-10
Shift-F2	85	-21	-11
Shift-F3	86	-22	-12
Shift-F4	87	-23	-13
Shift-F5	88	-24	-14
Shift-F6	89	-25	-15
Shift-F7	90	-26	-16
Shift-F8	91	-27	-17
Shift-F9	92	-28	-18

Shift-F10	93	-29	-19

Appendix I: FoxPro Reserved Words

If a dBASE IV program uses a FoxPro reserved word as an array name, FoxPro displays the error message "Attempt to use FoxPro function as an array." If you use a reserved word as a UDF name, FoxPro will call the internal command or function instead, with varying error messages.

#ENDIF	_PCOPIES	AFIELDS	AUTOSAVE	CHAIN
#IF	_PDRIVER	AFONT	AVERAGE	CHANGE
#ITSEXPRESSION	_PDSETUP	AFTER	AVG	CHR
#READCLAUSES	_PECODE	AGAIN	BAR	CHRSAW
#REGION	_PEJECT	AINS	BEFORE	CHRTRAN
#SECTION	_PEPAGE	ALEN	BEGIN	CLEAR
#WNAME	_PFORM	ALIAS	BELL	CLOCK
.AND.	_PLENGTH	ALL	BETWEEN	CLOSE
.F.	_PLINENO	ALLTRIM	BG	CMONTH
.NOT.	_PLOFFSET	ALTERNATE	BLANK	CNT
.OR.	_PPITCH	AMERICAN	BLINK	CNTBAR
.T.	_PQUALITY	AMPM( )	BLOCKSIZE	CNTPAD
@FUNCTION	_PRETEXT	AND	BOF	COL
@PROCEDURE	_PSCODE	ANSI	BORDER	COLOR
_ALIGNMENT	_PSCODE	ANSITOOEM	BOTTOM	COLOUR
_BOX	_PSPACING	APP	BOX	COLUMN
_CALCMEM	_PWAIT	APPEND	BR	COMMAND
_CALCVALUE	_RMARGIN	APPLICATION	BREAK	COMMIT
_CLIPTEXT	_TABS	ARRAY	BRITISH	COMPACT
_CUROBJ	_TALLY	AS	BROWSE	COMPATIBILITY
_DBLCLICK	_TEXT	ASC	BRSTATUS	COMPATIBLE
_DIARYDATE	_THROTTLE	ASCAN	BUILD	COMPILE
_DOS	_TRANSPORT	ASCENDING	BY	COMPLETED
_GENGRAPH	_WINDOWS	ASIN	CALCULATE	COMPRESS
_GENMENU	_WRAP	ASORT	CALL	CONFIRM
_GENPD	ABS	ASSIST	CANCEL	CONSOLE
_GENSCRN	ACCEPT	ASUBSCRIPT	CAPSLOCK	CONTINUE
_GENXTAB	ACCESS	AT	CARRY	CONVERT
_INDENT	ACOPY	ATAN	CASE	COPY
_LMARGIN	ACOS	ATC	CATALOG	COS
_MAC	ACTIVATE	ATCLINE	CDOW	COUNT
_MLINE	ADD	ATLINE	CDX	CREATE
_PADVANCE	ADDITIVE	ATN	CEILING	CTOD
_PAGENO	ADEL	ATN2	CENTER	CURDIR
_PBPAGE	ADIR	AUTO	CENTURY	CURRENCY
_PCOLNO	AELEMENT	AUTOMATIC	CGA	CURSOR

CYCLE	DIRECTORY	EXP	FSEEK	INKEY
DATABASE	DISABLE	EXPORT	FSIZE	INLIST
DATABASES	DISKSPACE	EXTENDED	FULLPATH	INPUT
DATE	DISPLAY	EXTERNAL	FV	INSERT
DAY	DISTINCT	FCHSIZE	FW	INSMODE
DB	DMY	FCLOSE	FW2	INSTRUCT
DB4	DO	FCOUNT	FWRITE	INT
DBASEII	DOHISTORY	FCREATE	GATHER	INTENSITY
DBDATE	DOS	FEOF	GB	INTO
DBF	DOUBLE	FERROR	GENERAL	ISALPHA
DBMEMO3	DOW	FFLUSH	GERMAN	ISCOLOR
DDEAbortTrans	DTOC	FGETS	GET	ISDIGIT
DDEAdvise	DTOR	FIELD	GETBAR	ISLOWER
DDEEnabled	DTOS	FIELDS	GETDIR	ISMARKED
DDEExecute	ECHO	FILE	GETENV	ISUPPER
DDEInitiate	EDIT	FILER	GETEXPR	ITALIAN
DDELastError	EGA25	FILES	GETFILE	JAPAN
DDEPoke	EGA43	FILL	GETFONT	JOIN
DDERequest	EJECT	FILTER	GETPAD	KEY
DDESetOption	ELSE	FIND	GETS	KEYBOARD
DDESetService	ELSEIF	FIXED	GO	KEYCOMP
DDESetTopic	EMPTY	FKLABEL	GOMONTH	LABEL
DDETerminate	ENABLE	FKMAX	GOTO	LAST
DEACTIVATE	ENCRYPT	FLOAT	GR	LASTKEY
DEBUG	ENCRYPTION	FLOCK	GROUP	LEDIT
DECIMALS	END	FLOOR	GROW	LEFT
DECLARE	ENDCASE	FLUSH	HAVING	LEN
DEFAULT	ENDDO	FONTMETRIC	HEADER	LEVEL
DEFINE	ENDFOR	FOOTER	HEADING	LIB
DELETE	ENDIF	FOPEN	HEIGHT	LIBRARY
DELETED	ENDPRINTJOB	FOR	HELP	LIKE
DELIMITED	ENDSCAN	FORM	HELPFILTER	LINE
DELIMITER	ENDTEXT	FORMAT	HIDE	LINENO
DELIMITERS	ENVIRONMENT	FOUND	HIGHLIGHT	LINKAGE
DESC	EOF	FOXPLUS	HISTORY	LIST
DESCENDING	ERASE	FOXPRO	HOURS	LKSYS
DESIGN	ERROR	FOXQ	IF	LOAD
DEVELOPMENT	ESCAPE	FOXSWAP	IIF	LOCATE
DEVICE	EVALUATE	FPUTS	IMPORT	LOCFILE
DIF	EXACT	FREAD	IN	LOCK
DIFFERENCE	EXCEPT	FREEZE	INDEX	LOG
DIMENSION	EXCLUSIVE	FRENCH	INDEXES	LOG10
DIR	EXIT	FROM	INFORMATION	LOGERRORS

LOGOUT	MULTILOCKS	NOZOOM	POINT	REDIT
LOOKUP	NDX	NPV	POP	REFERENCE
LOOP	NEAR	NUMBER	POPUP	REFRESH
LOWER	NETWORK	NUMLOCK	POPUPS	REGION
LPARTITION	NEXT	OBJECT	PRECISION	REGIONAL
LTRIM	NOALIAS	OBJNUM	PREFERENCE	REINDEX
LUPDATE	NOAPPEND	OCCURS	PREVIEW	RELATION
MACKEY	NOCLEAR	ODOMETER	PRIMARY	RELATIVE
MACRO	NOCLOSE	OEMTOANSI	PRINT	RELEASE
MACROS	NOCONSOLE	OF	PRINTER	REMARK
MARGIN	NODEBUG	OFF	PRINTJOB	RENAME
MARK	NODELETE	ON	PRINTSTATUS	REPLACE
MASTER	NOEDIT	ONKEY	PRIVATE	REPLICATE
MAX	NOEJECT	ONLY	PRMBAR	REPORT
MCOL	NOENVIRONMENT	OPEN	PRMPAD	REPROCESS
MDOWN	NOFLOAT	OPTIMIZE	PRODUCTION	RESET
MDX	NOFOLLOW	OR	PROGRAM	RESOURCE
MDY	NOGROW	ORDER	PROJECT	REST
MEMLINES	NOINIT	OS	PROMPT	RESTORE
MEMO	NOLGRID	OTHERWISE	PROPER	RESUME
MEMORY	NOLINK	OVERWRITE	PROW	RETRY
MEMOWIDTH	NOLOCK	PACK	PUBLIC	RETURN
MEMVAR	NOMARGIN	PAD	PUSH	RG
MENU	NOMENU	PADC	PUTFILE	RIGHT
MENUS	NOMODIFY	PADL	PV	RLOCK
MESSAGE	NOMOUSE	PADR	QUERY	ROLLBACK
MESSAGES	NONE	PAGE	QUIT	ROUND
MIN	NOOPTIMIZE	PALETTE	RAND	ROW
MINIMIZE	NOOVERWRITE	PANEL	RANDOM	RPD
MLINE	NOREFRESH	PARAMETERS	RANGE	RTOD
MOD	NORGRID	PARTITION	RANK	RTRIM
MODAL	NORM	PATH	RAT	RUN
MODIFY	NORMAL	PAUSE	RATLINE	SAFETY
MODULE	NOSAVE	PAYMENT	RB	SAME
MONO43	NOSHADOW	PCOL	RDLEVEL	SAMPLE
MONTH	NOSHOW	PDOX	READ	SAVE
MOUSE	NOT	PDRIVER	READERROR	SAY
MOVE	NOTAB	PDSETUP	READKEY	SCAN
MOVER	NOTE	PFS	RECALL	SCATTER
MRKBAR	NOTIFY	PI	RECCOUNT	SCHEME
MRKPAD	NOUPDATE	PICTURE	RECNO	SCOLS
MROW	NOWAIT	PLAIN	RECORD	SCOREBOARD
MULTI	NOWINDOW	PLAY	RECSIZE	SCREEN

SCROLL	SUMMARY	USER	WREAD
SDF	SUSPEND	USERS	WRK
SECONDARY	SYLK	USING	WROWS
SECONDS	SYS	VAL	WTITLE
SECTION	SYSMENU	VALID	WVISIBLE
SEEK	SYSMETIRC	VALUE	XLS
SELECT	SYSTEM	VALUES	YEAR
SELECTION	TAB	VAR	YMD
SEPARATOR	TABLE	VARREAD	ZAP
SEQUENCE	TABS	VERSION	ZOOM
SET	TAG	VGA	
SHADOW	TALK	VGA25	
SHADOWS	TAN	VGA50	
SHOW	TARGET	VIEW	
SIGN	TEXT	WAIT	
SIN	TEXTMERGE	WBORDER	
SINGLE	TIME	WCHILD	
SIZE	TIMEOUT	WCOLS	
SKIP	TITLE	WEXIST	
SKPBAR	TITLES	WFONT	
SKPPAD	TO	WHEN	
SNAP	TOP	WHERE	
SNAPCODE	TOPIC	WHILE	
SNAPMACRO	TOTAL	WIDTH	
SOFTSEEK	TRANSACTION	WINDOW	
SORT	TRANSFORM	WINDOWS	
SOUNDEX	TRAP	WITH	
SPACE	TRBETWEEN	WK	
SQL	TRIM	WK1	
SQRT	TXTWIDTH	WK3	
SROWS	TYPE	WKS	
STATUS	TYPEAHEAD	WLAST	
STD	UDFPARMS	WLCOL	
STEP	UNION	WLROW	
STICKY	UNIQUE	WMAXIMUM	
STORE	UNLOCK	WMINIMUM	
STR	UNPACK	WONTOP	
STRTRAN	UPDATE	WORKAREA	
STRUCTURE	UPDATED	WOUTPUT	
STUFF	UPPER	WPARENT	
STYLE	USA	WR	
SUBSTR	USE	WR1	
SUM	USED	WRAP	

INDEX

µ.DBF, 6, 18
.DBO, 18
.DBT, 7, 18
.FMO, 18
.FMT, 12, 18
.FRG, 13, 18
.FRM, 10, 12, 18
.FRO, 18
.LBG, 14, 18
.LBL, 10, 14, 18
.LBO, 18
.MDX, 7
.NDX, 7, 18, 58
.PRG, 18
.PRS, 15, 18
.QBE, 7, 18
.SCR, 10
<=, 26
=<, 26
== (comparison operator), 71
=>, 26
>=, 26
@...GET, 19, 20
@...GET MESSAGE, 35
@...GET OPEN WINDOW, 20
@...GET REQUIRED, 19
@...SAY, 20
ACCESS, 43
ACHOICE(), 71

ACTIVATE MENU, 21
ACTIVATE POPUP, 21
ACTIVATE SCREEN, 21
ACTIVATE WINDOW, 21
ADIR(), 75
AFIELDS(), 75
AFILL(), 75
ALTD(), 76
APPEND FROM FIELDS, 76
APPEND FROM WHILE, 76
APPEND MEMO, 21
Applications Generator, 15
array, 102
Arrays, 53
BARCOUNT(), 22
BARPROMPT(), 22
BEGIN SEQUENCE...[BREAK]...END, 66
BEGIN SEQUENCE..[BREAK]...END, 77
BEGIN TRANSACTION, 55
BIN2I(), 77
BIN2L(), 77
BIN2W(), 77
BLANK (version 1.5 only), 23
BROWSE, 23, 24, 25
BROWSE COMPRESS, 23
BROWSE NOFOLLOW, 24
BROWSE NOINIT, 24
BROWSE NOMENU, 24
BROWSE NOORGANIZE, 25
CALL, 25, 78
CALL(), 25
CANCEL, 78

CATALOG(), 25

CLEAR, 33, 78

COMMIT, 79

Comparison operators, 26

COMPLETED, 55

COMPRESS, 23

CTOD(), 26

Databases, 6

DBEDIT(), 79

DBFILTER(), 80

DBRELATION(), 80

DBRSELECT(), 81

DEFINE BAR, 35

DEFINE MENU, 27, 35

DEFINE PAD, 35

DEFINE POPUP, 35

DESCEND(), 81

DESCENDING(), 27

DOSERROR(), 82

END TRANSACTION, 55

Error handling, 65

ERROR(), 28

ERRORLEVEL(), 83

EXCLUSIVE, 56

Export .PRG, 108

EXTERNAL, 83

FCREATE(), 28

FDATE(), 29

FGETS(), 29

File Converter, 10, 11

File extensions, 18

FLDCOUNT(), 29

FLDLIST(), 30

FOPEN(), 28

FOR(), 30

Format files, 12

format files, 12

FPUTS(), 30

FREAD(), 84

FREADSTR(), 84

FSIZE(), 31

FTIME(), 31

Function Calls, 64

HARDCR(), 85

Help, 107

HOME(), 31

I2BIN(), 85

ID(), 32

IF, 86

IF(), 86

Indexes, 7

INDEXEXT(), 87

INDEXKEY(), 87

INDEXORD(), 87

INKEY(), 32, 70, 88

ISBLANK(), 33

ISMARKED, 55

ISMOUSE(), 33

ISPRINTER(), 88

KEYBOARD, 33

KEYBOARD CLEAR, 33

KEYMATCH(), 34

L2BIN(), 88

Labels, 14

LASTKEY, 17

LASTKEY(), 32, 70, 88

LASTREC(), 88

Level, 18, 70, 105

LIKE(), 34

LIST USERS, 34

LOCK(), 34

Memo files, 7, 57

MEMOEDIT(), 89

MEMOLINE(), 90

MEMOREAD(), 90

MEMORY(), 35, 91

MEMOTRAN(), 91

MEMOWRIT(), 92

MESSAGE, 35

MLCOUNT(), 92

MLPOS(), 92

NETERR(), 93

NETNAME(), 93

Network functions, 36

NETWORK(), 36

NEXTKEY(), 70, 93

NOFOLLOW, 24

NOINIT, 24

NOLOG, 55, 56

NOMENU, 24

NOORGANIZE, 25

NOSAVE, 8, 56

ON BAR, 36

ON EXIT BAR, 37

ON EXIT MENU, 37

ON EXIT POPUP, 37

ON KEY LABEL, 38
ON MENU, 38
ON MOUSE, 38
ON PAD, 39
ON POPUP, 39
ON SELECTION POPUP, 40
ON SELECTION POPUP BLANK, 40
OPEN WINDOW, 20
Order, 106
PADPROMPT(), 40
Parameter passing, 17
PCOUNT(), 41, 94
Printer drivers, 13
PROCLINE(), 94
PROCNAME(), 94
PROTECT, 43
Queries, 7
READEXIT(), 95
READINSERT(), 95
READKEY, 17
READKEY(), 32
READVAR(), 96
REPLACE FROM ARRAY, 41
Reports, 12
Reports, Program Analyzer, 108
REQUIRED, 19
RESET, 55
RESTORE FROM, 42
RESTORE SCREEN, 42
RESTSCREEN(), 97
RLOCK(), 34
ROLLBACK, 55

RUN(), 42

SAVESCREEN(), 97

Screens, 11

SCROLL(), 97

Security, 43

SELECT(), 43

Sensitivity, 106

SET ("TRAP"), 52

SET CATALOG, 8, 44

SET CENTURY, 69

SET COLOR TO, 45

SET CONFIRM, 69

SET CONSOLE, 69

SET CURSOR, 69, 98

SET DATE, 69

SET DBTRAP, 45

SET DELETED, 69

SET DELIMITERS, 69

SET DESIGN, 45

SET DIRECTORY, 46

SET ENCRYPTION, 43

SET ESCAPE, 69

SET EXACT, 69

SET EXCLUSIVE, 69

SET FIELDS, 8, 47

SET FIXED, 69

SET FORMAT, 47

SET FUNCTION, 88

SET INSTRUCT, 48

SET INTENSITY, 69

SET KEY, 48, 98

SET LDCHECK, 49

SET LIBRARY, 49
SET MBLOCK, 50
SET MESSAGE, 50
SET PAUSE, 50
SET PRECISION, 50
SET PRINT, 69
SET PRINTER TO FILE, 51
SET RELATION, 99
SET SCOREBOARD, 69
SET SOFTSEEK, 69, 99
SET TITLE, 51
SET TRAP, 52
SET UNIQUE, 69
SET VIEW, 52
SET WRAP, 69, 99
SET("ATTRIBUTES"), 44
SET("BORDER"), 44
SET("CATALOG"), 8, 44
SET("DESIGN"), 45
SET("DIRECTORY"), 46
SET("DISPLAY"), 46
SET("ENCRYPTION"), 43
SET("INSTRUCT"), 48
SET("LIBRARY"), 49
SET("PAUSE"), 50
SET("PRECISION"), 51
SET("SQL"), 51
SET("TITLE"), 51
SET("TRAP"), 52
SET("VIEW"), 52
SET("WINDOW"), 53
SETCANCEL(), 100

SETCOLOR(), 100

SETPRC(), 100

SQL, 15, 53

Steps to Take, 57

STORE, 53

SUM(), 53

TAG(), 54

TAGCOUNT(), 54

TAGNO(), 8, 54

TEXT, 101

Text editor, using your own, 107

TONE(), 101

Transaction processing, 55

TYPE(), 102

UNIQUE(), 55

USE, 56

USER, 43

View, 106

View Rules, 108

WINDOW(), 56

WORD(), 102