

Language Reference

Microsoft **FOXPRO**®

*Relational Database Management System
for MS-DOS® and Windows™*

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

©1989–1993 Microsoft Corporation. All rights reserved.

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

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

Macintosh and MultiFinder are registered trademarks of Apple Computer, Inc.

RapidFile is a registered trademark and Framework II is a trademark of Ashton-Tate Corporation.

HP and LaserJet are registered trademarks of Hewlett-Packard Company.

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

Netware and Novell are registered trademarks of Novell, Inc.

DESQview is a registered trademark of Quarterdeck Office Systems.

UNIX is a registered trademark of UNIX Systems Laboratories.

Contents

Overview of the FoxPro Language

Command Window	1-2
Creating Programs	1-3
Saving Programs	1-3
Modifying Programs	1-4
Executing Programs	1-4
Additional Program Information	1-4
FoxPro Online Help Facility	1-5
Online Help in FoxPro for MS-DOS	1-5
Online Help in FoxPro for Windows	1-5
Understanding the Language	1-7
Syntax Conventions	1-7
Expressions	1-9
Additional Abbreviations	1-16
Specifying Record Range	1-18
Table/.DBF Work Areas	1-21
Functions and Memo Fields	1-23
Typographic Conventions	1-24

Commands and Functions by Category

Key FoxPro Commands and Functions	2-2
Command and Function Categories	2-6
Data Types	2-7
Character Functions	2-7
Numeric Functions	2-10
Date and Time Functions	2-12
Data Conversion	2-13
Tables/.DBFs	2-14
Field Manipulation	2-14
Table/.DBF Manipulation	2-15
Record Manipulation	2-17
Indexes	2-19
Relations	2-20

Programming	2-21
Debugging and Error Handling	2-21
Program Execution	2-22
Structured Programming	2-24
Low-Level File Functions	2-24
Memory Variable Manipulation	2-25
Array Manipulation	2-27
Event Handlers	2-28
DDE and OLE	2-29
Input and Output	2-30
Data Formatting	2-30
Printing	2-34
Menu Bars and Popups	2-35
Windows	2-37
Typing, Keyboard and Mouse	2-39
Reports and Labels	2-41
FoxPro Environment	2-42
Environment	2-42
File Management	2-47
FoxPro in a Multi-User Environment	2-49
Network Commands and Functions	2-49

Commands and Functions Alphabetically

This chapter contains complete descriptions of the FoxPro commands and functions. The chapter is organized alphabetically and begins on page 3-1.

System Memory Variables Alphabetically

This chapter contains complete descriptions of the FoxPro system memory variables. The chapter is organized alphabetically and begins on page 4-1.

Index

1 Overview of the FoxPro Language

FoxPro[®] is a product developed for data management — the cataloging, tracking and processing of information. Although you can perform many data management operations non-procedurally through the FoxPro interface, accessing the tremendous power of FoxPro requires some knowledge of the FoxPro programming language.

This manual is a complete language reference for FoxPro on two platforms — MS-DOS[®] and Microsoft[®] Windows[™]. FoxPro is used generically in this manual to refer to both FoxPro for MS-DOS and FoxPro for Windows. Whenever a language feature is unique to FoxPro for MS-DOS or FoxPro for Windows, the platform is noted and the feature is fully documented. See the table at the beginning of the Commands and Functions Alphabetically chapter for a complete listing of new and enhanced language features in FoxPro 2.5.

This chapter contains a variety of topics that will help you use the FoxPro programming language, including an explanation of the conventions this manual uses to describe the syntax of FoxPro commands and functions.

Command Window

When you use the interface, FoxPro generates commands in the Command window that correspond to your actions. The Command window provides a gateway between the interface and the FoxPro language.

The Command window contains a history of all the commands created during an interactive FoxPro session, providing a means to learn the FoxPro language. By examining the commands that are generated when you use the interface, you can quickly learn the language.

A command generated in the Command window can be re-executed by moving the cursor to the line containing the command and pressing Enter. After a long interactive FoxPro session, a lengthy list of FoxPro commands is generated in the Command window. You can copy and paste these commands into a FoxPro *program*. A program can be repeatedly run, making it easy to execute literally thousands of commands again and again.

A FoxPro command can be directly executed by typing it into the Command window and pressing Enter. To re-execute a command, move the cursor to the line containing the command and press Enter.

Since the Command window is a special editing window, you can edit commands using the editing tools available in FoxPro. You can edit, insert, delete, cut, copy or paste text in the Command window.

For more information about the Command window, see the Window Menu chapter in the FoxPro *User's Guide*.

Creating Programs

FoxPro programs, composed of FoxPro commands, provide incredible power to manipulate data.

You can execute a series of commands that you enter into a FoxPro program file.

Running a program has many advantages over entering individual commands in the Command window:

- Programs execute multiple commands automatically, saving time and lots of typing.
- Programs can be modified and run again.
- Programs can run other programs.
- Certain structured programming commands can be used only in programs.
- Programs run in FoxPro provide unparalleled performance.

FoxPro programs are text files containing a series of commands. You can create a program in FoxPro in one of these ways:

- Choose **New...** from the **File** menu. The New dialog appears. Choose the **Program** radio button, then choose **OK** in FoxPro for MS-DOS or **New** in FoxPro for Windows.
- In the Command window, type the command **MODIFY COMMAND** and press Enter.

FoxPro opens a new window named UNTITLED.PRG. You can now enter your program into this window.



If you have used FoxPro interactively and want to save the commands generated in the Command window, you can copy and paste them directly into a program editing window. You can also copy program examples from the help file and paste them directly into a program. Copying program examples from the help file is a quick and easy way to learn how FoxPro commands and functions work.

Saving Programs

After you create a program, be sure to save it. To save an untitled program, choose **Save As...** from the **File** menu. Enter a name for the program in the Save As dialog that appears. Once you save your program you can execute the program or modify it.

Modifying Programs

After you save your program, you can modify it. First, open the program in one of these ways:

- Choose **Open...** from the **File** menu. A dialog appears with a list of available files. Choose the **Program** option from the **Type** popup. From the list of files, choose the program you want to modify, then choose **OK** in FoxPro for MS-DOS or **Open** in FoxPro for Windows.
- In the Command window, type the command `MODIFY COMMAND` followed by a space and the name of the program to modify, then press Enter. The program is opened for editing. For example, to modify a program named `MYPROGRM`, enter this command in the Command window:

```
MODIFY COMMAND myprogram
```

- In the Command window, type the command `MODIFY COMMAND ?` then press Enter. From the list of files, choose the program you want to modify, then choose **OK** in FoxPro for MS-DOS or **Open** in FoxPro for Windows.

After you open the program you can make changes. When you finish making your changes, be sure to save the program.

Executing Programs

After you create a program, you can execute it. To execute a program:

- Choose **Do...** from the **Program** menu. From the program list, choose the program to execute, then choose **OK** in FoxPro for MS-DOS or **Do** in FoxPro for Windows.
- Type the command `DO` followed by a space and the name of the program to execute, then press Enter. For example, to execute the program named `MYPROGRM`, enter this command in the Command window:

```
DO myprogram
```

Additional Program Information

For complete information on creating, editing and saving program files, see the Text Editing section of the Interface Basics chapter in the *FoxPro User's Guide*. For information on executing program files, see the Program Menu chapter in the *FoxPro User's Guide* or the `DO` command in this manual.

FoxPro Online Help Facility

An online help facility is always available in FoxPro. The online help facility contains specific information about every FoxPro command, function and system memory variable, as well as general information about FoxPro.

Context-sensitive help is available for FoxPro system windows, dialogs and menu options. Press F1 when a system window or dialog is frontmost to see help information about it. To get help for a command or function in the Command window or a program, select the command or function and press F1.

Online Help in FoxPro for MS-DOS

To access online help in FoxPro for MS-DOS, press F1 or choose **Help...** from the **System** menu. The Help window is displayed with an extensive list of topics. Choose a topic in the Help window to see details about the topic.

You can get help on related topics in the help facility by choosing options from the **See Also** popup. You can also select text in a help topic and choose the **Look Up** push button. If the text you select is a help topic, help information appears for the topic. If a help topic does not correspond to the text you select, a dialog is displayed containing a list of help topics.

For more information about help in FoxPro for MS-DOS, refer to SET HELP in this manual or the System Menu chapter in the *FoxPro User's Guide*.

Online Help in FoxPro for Windows

To access online help in FoxPro for Windows, press F1 or choose **Help...** from the **Help** menu. The Help window opens.

FoxPro for Windows supports two styles of help:

- .DBF-style help, familiar to FoxPro for MS-DOS users.
- Windows-style help, familiar to Windows users.

Windows-style help is the default help system.

You can change from one style to another in the Command window or in a program:

- To change to FoxPro .DBF-style help, issue `SET HELP TO FOXHELP.DBF` in the Command window or a program. FoxPro .DBF-style help behaves identically to the online help for FoxPro for MS-DOS, described in the previous section except you choose **Contents** from the **Help** menu to display the Help window.
- To change to Windows-style help, issue `SET HELP TO FOXHELP.HLP` in the Command window or a program.

With Windows-style help, you can:

- Display information about the current window or dialog at any time by pressing the F1 key.
- Get help on a menu option by pressing Shift+F1, then selecting the menu option.
- Search for topics associated with a word or phrase by choosing **Search for Help on...** from the **Help** menu.
- Add notes or comments to a topic by choosing **Annotate...** from the **Edit** menu in the Help window, then typing your notes. A paper clip symbol appears in topics with annotations. To view the annotation, click the paper clip.
- Mark a topic you refer to frequently so that you can access it quickly. You do this by choosing **Define...** from the **Bookmark** menu in the Help window, then specifying a bookmark name for the current topic. Bookmarks that you define are added to the bottom of the **Bookmark** menu.
- Print help topics by choosing **Print Topic** from the **File** menu in the Help window.
- Copy the contents of help topics to the clipboard by choosing **Copy...** from the **Edit** menu in the Help window and selecting the topic you want copied.

For more information about the online help facility in FoxPro for Windows, refer to the Help Menu chapter in the FoxPro for Windows *User's Guide*, the System Menu chapter in the FoxPro for MS-DOS *User's Guide* or SET HELP in this manual.

Understanding the Language

The FoxPro language consists of two elements:

- Commands that perform an action. For example, you can use the BROWSE command to look at the contents of a table/.DBF.
- Functions that return a value. For example, the DATE() function returns the system date from your computer. Functions contain a pair of parentheses that distinguish them from commands.

Commands and functions can be combined to create a FoxPro *statement*. Functions are not used by themselves, but are always combined with a FoxPro command.

For example, the question mark (?) is a command that sends output to the screen. DATE() is a function that returns the current system date. They can be combined to output the system date to the screen:

```
? DATE( )
```

The STORE command assigns a value to a memory variable. In this example, the TIME() function returns the current system time, and this value is placed in the memory variable BEGINTIME:

```
STORE TIME( ) TO begintime
```

Syntax Conventions

FoxPro commands and functions consist of one or more components. These components determine how a command or function can be used.

Look at the syntax of a typical FoxPro command. The REPLACE command is used to update fields in a table/.DBF record. Here's the syntax for the REPLACE command:

```
REPLACE <field1> WITH <expr1> [ADDITIVE]  
    [, <field2> WITH <expr2> [ADDITIVE]] ...  
    [SCOPE] [FOR <expL1>] [WHILE <expL2>]  
    [NOOPTIMIZE]
```

The following table lists the components of the REPLACE command.

Component Type	Examples in the REPLACE Command
Key Words	REPLACE, WITH, ADDITIVE, FOR, WHILE, NOOPTIMIZE
Expressions	expr1, expr2, expL1, expL2
Names	field1, field2

This manual uses special typographic conventions in command and function syntax. These conventions indicate whether a component of a command or function should be entered as shown or supplied by you, whether the component is required or optional, and so on.

UPPER-CASE UPPER-CASE designates a FoxPro key word. Every FoxPro command and function contains at least one key word. FoxPro uses *key words* to identify a command or function. Key words are also called *reserved words* since FoxPro reserves these special words for internal use.

Some examples of key words in the REPLACE command are REPLACE (which FoxPro identifies as a command), WITH, ADDITIVE, FOR and WHILE. WITH, ADDITIVE, FOR and WHILE are also referred to as *clauses* because they are additional parts of the main command.

< > Angle brackets < > surrounding lower-case text indicate information that you provide. This information can be a file name, an expression, a memory variable name and so on. You replace the text and angle brackets with the appropriate name or expression. Do not include the angle brackets when entering the command.

() Parentheses must be included in all FoxPro functions. Include the parentheses as they appear in the function's syntax.

- [] Square brackets indicate a portion of a FoxPro command or function that is optional. Do not type the brackets. For example, the REPLACE command supports numerous options, and you have the choice of including these options. The ADDITIVE, <scope>, FOR <expL1>, WHILE <expL2> and NOOPTIMIZE clauses are placed in square brackets to indicate they are optional.
- | A vertical bar separating options indicates that you can enter only one of the options. Do not type the vertical bar.
- ... An ellipsis indicates that a command or function or a portion of a command or function can be continued in a similar fashion. For example, the INLIST() function supports a large number of arguments. The ellipsis in the INLIST() syntax indicates that you can include more arguments than the three listed:

INLIST(<expr1>, <expr2> [, <expr3> ...])

Expressions

A FoxPro expression is a combination of field values, functions, memory variable values, array element values, constants and operators. There are four types of FoxPro expressions — character, numeric, date and logical.

Expressions are denoted in FoxPro commands and functions as shown in the following table:

Expression	Expression Type
<expC>	Character
<expN>	Numeric
<expD>	Date
<expL>	Logical
<expr>	Character, Numeric, Date or Logical
<expr list>	List of expressions separated by commas

When more than one expression of the same type appears in a command or function, a number is added to indicate its position in the command or function. For example, the syntax for the `CHRTRAN( )` function is:

CHRTRAN(<expC1>, <expC2>, <expC3>)

`CHRTRAN( )` requires three character expression arguments: <expC1>, <expC2> and <expC3>.

Fields, functions, memory variables and array elements that are combined to create an expression must be the same data type. If they are different types, FoxPro displays an “Operator/operand type mismatch” message.

Character Expressions

Character expressions are composed of the following:

- Character type fields.
- Functions that return character values.
- Memory variables and array elements that contain character data.
- Character type constants, called *string literals*.

A string literal is a character string surrounded by single or double quotation marks. For example, `'Washington'` or `"Washington"`. The quotation marks must match. That is, you cannot begin a character string with a single quotation mark and end it with a double quotation mark.

You can embed a quotation mark in a character string by surrounding the character string with quotation marks of the opposite type. For example, `"Don't touch!"` is a legitimate string literal.



Throughout the FoxPro documentation you will see the *null string* mentioned. The null string is a character string of length 0 — it contains no characters. The null string can be expressed with a pair of quotation marks with no space between them — `' '` or `""`.

The following operators can be used with character expressions:

Character Expression Operators	
Operator	Operation
+	Character expression concatenation (two strings are joined together)
-	Character expression concatenation (trailing blanks are moved from the end of the first string to the end of the second string)
\$	Character expression comparison

You can use character expression operators to:

- Concatenate strings. For example:

```
'Good ' + 'Morning!'
```

- Concatenate a string and a field. For example:

```
"Please respond by " + invoice.date
```

- Concatenate two fields and remove trailing blanks. For example:

```
customer.first - customer.last
```

- Concatenate a string and a memory variable. For example:

```
"Dear " + mfirstname
```

Numeric Expressions

Numeric expressions are composed of the following:

- Numeric type fields.
- Functions that return numeric values.
- Memory variables and array elements that contain numeric data.
- Numeric constants.

Numeric constants are numbers — 0, 1, -1, 3.1415 and so on.

Numeric Expression Operators (in order of precedence)	
Operator	Operation
()	Parentheses used to group expressions
**, ^	Exponentiation
*, /	Multiplication and division
%	Modulus (Remainder)
+, -	Addition and subtraction

Date Expressions

Date expressions are composed of the following:

- Date type fields.
- Functions that return date values.
- Memory variables and array elements that contain dates.
- Date constants.
Date constants are numbers — 1, 2, 3, 100 and so on. Date constants represent days, and you can add or subtract a date constant (a number of days) to or from a given date.

A date is specified by surrounding it with braces. For example, the following command creates the date type memory variable XMAS and stores the date 12/25/92 to it:

```
STORE {12/25/92} TO xmas
```

The following commands both create an empty date type memory variable BLANKDATE:

```
STORE { / / } TO blankdate  
STORE { } TO blankdate
```

FoxPro treats an invalid date as an empty date.

Logical Expressions

Logical expressions have one of two values — true or false. `.T.` and `.F.` are used to specify true and false in FoxPro.

Logical expressions are composed of the following:

- Logical type fields.
- Functions that return logical values.
- Memory variables and array elements that contain logical values.
- Other expression types (character, numeric or date) separated by special relational operators. For example:

```
invoice.date >= {06/01/92} AND invoice.amount > 15000
```

FoxPro evaluates logical expressions from left to right, and only for as long as is necessary. In the following example the AND operator is used to create a logical expression. If any of the values in the expression are false (`.F.`), the entire expression is false. When FoxPro encounters the first logical false (`.F.`), it doesn't evaluate the rest of the expression.

```
.T. AND .T. AND .F. AND .T. AND .T. AND .T.
```

Logical Expression Operators (in order of precedence)	
Operator	Operation
()	Parentheses used to group expressions
!, NOT	Logical negative
AND	Logical AND
OR	Logical inclusive OR

Logical expression relational operators are used with all data types to return a logical value. The following relational operators can be used with all data types create a logical expression.

Logical Expression Relational Operators	
Operator	Operation
<	Less than
>	Greater than
=	Equal to
< >, #, !=	Not equal to
< =	Less than or equal to
> =	Greater than or equal to
= =	Character string comparison

The following commands demonstrate how the relational operators can be used with numeric, date and character type data:

```
? 1 > 2                && Displays .F.  
? {01/01/92} <= {01/01/92}  && Displays .T.
```

Names

Certain FoxPro commands and functions require you to supply a name. Some examples of names used in FoxPro are:

- Table/.DBF file names.
- Table/.DBF alias names.
- Table/.DBF field names.
- Index file names.
- File names.
- Memory variable and array names.
- Window names.
- Menu names.
- Popup names.

In the following table, the first column lists names as they appear in the syntax of commands and functions. The second column describes what to include in the command or function.

Name	Name to Include
<file>	File name
<field>	Field name
<field list>	List of field names
<array>	Memory variable array name
<memvar>	Memory variable or array element name
<memvar list>	List of memory variable names
<menu name>	Menu name
<pad name>	Menu pad name
<popup name>	Popup name
<popup list>	List of popup names
<window name>	Window name
<window list>	List of window names

When including a list of names, the names must be separated by commas. A name is not an expression, a memory variable or array element or a field. A name should not be surrounded by quotation marks.

The REPLACE command used as an example requires that you include at least one name, field1, a field name.

Name Expressions

Although a name cannot be a memory variable or an array element, you can create a *name expression* that substitutes the *value* of a character type memory variable or array element as the name. Name expressions provide flexibility in FoxPro commands and functions.

Store the name to the memory variable (or array element). Then, you can substitute the name into a command or function by enclosing the memory variable in parentheses.

For example, the REPLACE command requires a field name. You can store a field name to a memory variable and use a name expression in REPLACE where the field name occurs:

```
STORE 'city' TO mvarcity  
REPLACE (mvarcity) WITH 'Redmond'
```

Here the field name CITY is stored to the memory variable MVARCITY. The name expression (mvarcity) replaces the memory variable with the field name for use by the REPLACE command.

A similar process called macro substitution enables memory variables to replace names. In *macro substitution*, you place an ampersand (&) before the memory variable to tell FoxPro to use the value of the memory variable as a name. For more information about macro substitution, see the & topic in this manual. A command or function containing a name expression executes faster than one containing macro substitution, so use a name expression instead of macro substitution whenever possible.

Additional Abbreviations

The following table contains terms and abbreviations commonly used in the description of FoxPro commands and functions.

Abbreviation	Meaning
alias	Work area alias
array	Array name
border string	Border definition string
column	Screen or window column
command	FoxPro command

Abbreviation	Meaning
delimiter	Separation character
dir	Directory
drive	Disk drive name
key	Index expression
list	Set of expressions, fields, and so on
macro name	Keyboard macro name
memo field	Memo field name
memvar	Memory variable or array element
pad name	Menu pad name
parm list	List of parameters
path	Disk directory specification
port	Serial or parallel output port name
procedure name	Procedure file name
row	Screen or window row
scope	Range of table/.DBF records
skel	File specification skeleton
statements	Set of FoxPro commands
text lines	Text that is not a FoxPro command
topic	Subject in FoxPro help file
work area	Table/.DBF work area

Specifying Record Range with Scope, FOR and WHILE

When a command operates on records in a table/.DBF, you can typically specify a range of records that the command affects. You specify the record range using scope, FOR and WHILE clauses.

Scope

When a scope is included, the command acts on a specific range of records in the table/.DBF. You can include one of the following clauses to specify a record range:

- ALL – The command acts on all records in the table/.DBF.
- NEXT <expN> – The command acts on a range of records beginning with the current record and continuing for the specified number <expN> of records. NEXT 1 acts on the current record.
- RECORD <expN> – The command acts on the specified table/.DBF record number <expN>.
- REST – The command acts on a range of records beginning with the current record and ending with the last record in the table/.DBF.

FOR <expL>

The record range of a command can also be specified using FOR and WHILE clauses.

When a FOR clause is used, each record that meets the specified logical condition is affected by the command.

Rushmore Technology and FOR Clauses

The Rushmore Technology is a patent-pending data access technique that permits sets of records to be accessed very efficiently, at speeds comparable to single-record indexed access. Using the Rushmore Technology, some complex database operations run hundreds or even thousands of times faster than before. FoxPro enables personal computers to handle truly gigantic databases, containing millions of records, at speeds comparable to mainframe database systems.

In its simplest form, Rushmore speeds up single table/.DBF commands utilizing FOR clauses that specify sets of records in terms of existing indexes. You can take advantage of Rushmore anywhere that a FOR clause appears. Rushmore is designed so that its speed is proportional to the number of records retrieved.

Potentially Optimizable Commands With FOR Clauses			
AVERAGE	COUNT	JOIN	REPORT
BROWSE	DELETE	LABEL	SCAN
CALCULATE	DISPLAY	LIST	SORT
CHANGE	EDIT	LOCATE	SUM
COPY TO	EXPORT	RECALL	TOTAL
COPY TO ARRAY	INDEX	REPLACE	

The data retrieval optimization that Rushmore provides is dependent on the FOR clause expression. The FOR clause expression may be simple or complex, and data retrieval speeds can be enhanced by including an optimizable FOR expression.

Optimizable expressions can be combined with the AND, OR and NOT logical operators to form a complex FOR clause expression. When basic expressions are combined into a complex expression, the resulting expression may also be optimizable.

A basic optimizable expression takes one of the following forms:

`<index expression> <relational operator> <constant expression>`

or

`<constant expression> <relational operator> <index expression>`

In a basic optimizable expression:

- `<index expression>` must exactly match the expression on which an index is constructed and `<index expression>` must not contain aliases.
- `<relational operator>` must be one of the following: `<`, `>`, `=`, `<=`, `>=`, `<>`, `#`, `!=`.
- `<constant expression>` can be any expression, including memory variables and fields from *unrelated* tables/.DBFs.

For example, if you have indexes on the following expressions:

```
firstname  
custno  
UPPER(lastname)  
hiredate  
addr
```

then the following are basic optimizable expressions:

```
firstname = 'Fred'  
custno >= 1000  
UPPER(lastname) = 'SMITH'  
hiredate < {12/30/90}
```

If you issue this command:

```
STORE 'WASHINGTON AVENUE' TO X
```

then the following are also basic optimizable expressions:

```
addr = X  
addr = SUBSTR(X,8,3)
```

For additional information on Rushmore, see the Optimizing Your Application chapter in the FoxPro *Developer's Guide*.

WHILE <expL>

The WHILE expression causes the command to act on each record for as long as the logical expression evaluates to true. The first time the expression evaluates to false, the command ceases without considering any remaining records. This expression is typically used with a table/.DBF that's been sorted or indexed on the field(s) included in the WHILE expression.

Scope, FOR and WHILE expressions can all be used in the same FoxPro command. When FOR and WHILE are both specified, the WHILE expression takes precedence and the FOR clause is used to filter the records selected by WHILE.

Table/.DBF Work Areas

In FoxPro for Windows and the 32-bit Extended version of FoxPro for MS-DOS, you can open and manipulate tables/.DBFs in 225 work areas. Work areas can be identified with the letters A through J for the first ten work areas, the numbers 1 – 225 or by the table/.DBF alias if a table/.DBF is open in a work area.

In the Standard version of FoxPro for MS-DOS you can open and manipulate tables/.DBFs in 25 work areas. Work areas can be identified with the letters A through J for the first ten work areas, the numbers 1 – 25, or the table/.DBF alias if a table/.DBF is open in a work area.

To open a table/.DBF in a work area other than the current work area, you must specify or select the work area. When a table/.DBF is opened, it is given an alias by which it can be identified. Unless you specify otherwise, the alias is the name of the table/.DBF (excluding the .DBF extension).

Default Alias

If you open the file named CUSTOMER.DBF in work area A with the following commands, the alias CUSTOMER is automatically assigned:

```
SELECT A
USE customer
```

The alias can then be used to identify the table/.DBF in a command or function.

User Assigned Alias

You can assign an alias to a table/.DBF when you open the file. You can open CUSTOMER.DBF (in work area A) and assign it an alias of PEOPLE with this command:

```
SELECT A
USE customer ALIAS people
```

The alias PEOPLE can then be used to refer to the table/.DBF. An alias can consist of up to 10 letters, digits or underscores and it must begin with a letter or an underscore. FoxPro automatically creates an alias if the alias you provide contains an unsupported alias character.

For complete information on assigning table/.DBF aliases, see the USE command.

FoxPro Assigned Alias

There are instances where FoxPro will automatically assign an alias to a table/.DBF. If a single table/.DBF is opened simultaneously in multiple work areas by including the AGAIN key word in USE and you do not specify an alias when opening the table/.DBF in each work area, a default alias is assigned automatically.

In FoxPro for Windows and the 32-bit Extended version of FoxPro for MS-DOS, the default aliases assigned in the first ten work areas are the work area letters A through J; the aliases assigned in work areas 11 through 225 are W11 through W225.

In the Standard version of FoxPro for MS-DOS the default aliases assigned in the first ten work areas are the work area letters A through J; the aliases assigned in work areas 11 through 25 are W11 through W25.

If a conflict occurs with aliases, a default alias is assigned.

Using an Alias

If CUSTOMER.DBF is opened in work area C (or work area 3) and the default alias of CUSTOMER is assigned, you can move to this work area from another with any one of the following SELECT commands:

```
SELECT C
SELECT 3
SELECT customer
```

You can also refer to fields in other work areas by prefacing the field name with the alias pointer. Preface the field name with the alias name or the work area letter and a period or the -> operator.

If you're in work area C and you want to access a field called CONTACT from CUSTOMER.DBF open in work area A, use one of the following:

```
customer.contact
A.contact
```

If you're in a work area other than A, CUSTOMER.DBF is open in work area A and has the alias PEOPLE, you can use one of these commands:

```
people.lastname
A.lastname
```


Referencing a Work Area

You can reference a work area before a table/.DBF is opened by using the work area letter or number as shown below:

```
SELECT A
SELECT 1
```

Memory Variables

Occasionally, memory variables are created with the same names as field names. If a memory variable has a name that's the same as a field name, FoxPro always gives the field precedence over the memory variable. You can reference a memory variable with the same name as a field by including the M period or M-> prefix to refer to the memory variable. For example:

```
M.lastname
```

Functions and Memo Fields

FoxPro memo fields can contain any data whatsoever, including nulls, assembler code, printer drivers, and so on, and they can be any length. However, not all string and character functions can be used with memo fields. The following functions are the only ones that operate directly on unlimited length memo fields:

```
$
LEN( )
AT( )
ATC( )
ATLINE( )
ATCLINE( )
MEMLINES( )
MLINE( )
SUBSTR( )
```

In addition, the REPLACE command permits an arbitrary length memo field or a substring of an arbitrary length memo field to replace another memo field.

Memo fields can be stored as memory resident strings. These strings can then be manipulated with all functions that operate on character data. String length is limited to 64K in the Standard version of FoxPro for MS-DOS, and is limited only by available memory in FoxPro for Windows and the 32-bit Extended version of FoxPro for MS-DOS.

Typographic Conventions

While most of the material in this manual deals directly with the FoxPro language, some of the commands have counterparts in the FoxPro interface. When describing actions that occur in the interface, this manual uses the following conventions.

Controls

When we refer to FoxPro controls, such as lists, menus, menu options, popups, popup options, push buttons, radio buttons and check boxes, the name of the control appears in **bold type**. For example:

Choose **Save** from the **File** menu.

Windows and Dialogs

Names of FoxPro system windows and dialogs always appear in regular text with an initial capital. For example:

Type this statement in the Command window to open the View window.

Input and Output

Examples of commands that you can type into the Command window or a program appear in `Courier` font. For example:

```
STORE amt_due - amt_paid TO new_bal  
REPLACE amt_due WITH new_bal
```

When you issue a command or function that causes output to appear, the resulting output appears in the documentation in **Courier Bold**. For example:

```
USE customer  
AVERAGE ytdpurch FOR state = 'NY'  
46 records averaged.  
YTDPURCH  
3628.19
```

2 Commands and Functions by Category

FoxPro contains over 600 commands, functions and system memory variables. If you are not familiar with the FoxPro language, this chapter will help you focus on the commands and functions that meet your needs.

This chapter is divided into two sections:

- The first section, Key FoxPro Commands and Functions, lists the common commands and functions you can use to carry out basic tasks in FoxPro. If you are new to FoxPro, familiarize yourself with these commands and functions.
- The second section, Command and Function Categories, presents the FoxPro commands and functions in an abbreviated form, grouped into categories based on purpose. Consult this section if you know what task you'd like to accomplish but are not sure of the command or function to use.

Key FoxPro Commands and Functions

This section provides new users with an overview of FoxPro commands and functions for accomplishing basic tasks. These commands and functions represent the most powerful and frequently used commands and functions in the FoxPro language. Once you understand these commands and functions, you'll be ready to create FoxPro programs. While there are many other commands and functions you will want to learn about, this list provides a good starting point.

Many FoxPro commands and functions provide backward compatibility with FoxBASE+® and other Xbase products. This is noted in the descriptions of these commands and functions in the Commands and Functions Alphabetically chapter. It isn't necessary for you to become familiar with these commands and functions.

For additional information about a specific command or function, see the Commands and Functions Alphabetically chapter.

Key Commands and Functions	
Command or Function	Description
? ??	Evaluates expressions and outputs the results.
@ ... SAY	Displays and formats a specified expression, field, memory variable or array element. If you use the Screen Builder to create your data entry screens, you may not have to use @ ... SAY.
@ ... GET	Creates an editing region or a control. If you use the Screen Builder to create your data entry screens, you may not have to use @ ... GET.
ACTIVATE WINDOW	Displays and activates a window created with DEFINE WINDOW.
ALLTRIM()	Returns a character expression with leading and trailing blanks removed.
APPEND FROM	Adds records to the end of the current table/.DBF from another file.
AT()	Returns the beginning numeric position of the first occurrence of a character expression or memo field within another character expression or memo field, counting from the leftmost character.

Key Commands and Functions	
Command or Function	Description
BOF()	Determines if the record pointer is positioned at the beginning of a table/.DBF.
BROWSE	Opens the Browse window, in which you can view and/or edit table/.DBF records.
CANCEL	Cancels execution of the current program file.
COPY TO	Copies the contents of the current table/.DBF to a new file.
COUNT	Counts table/.DBF records.
CREATE	Creates a new table/.DBF.
DEFINE WINDOW	Creates a window.
DELETE	Marks records in a table/.DBF for deletion.
DELETED()	Returns true if the current record is marked for deletion.
DISPLAY	Displays table/.DBF records or expressions.
DISPLAY MEMORY	Displays the contents of memory variables and arrays.
DISPLAY STATUS	Displays the status of the FoxPro environment.
DISPLAY STRUCTURE	Displays the structure of a table/.DBF.
DO	Executes a FoxPro program.
DO CASE	Executes a set of commands based on a logical condition.
DO WHILE	Executes commands inside a loop while a logical condition remains true.
EMPTY()	Determines if an expression is blank.
EOF()	Returns true if the record pointer is positioned at the end of a table/.DBF.
FLOCK()	Attempts to lock a table/.DBF and returns true if successful.
FOR ... ENDFOR	Executes statements in a loop a specified number of times.
FOUND()	Returns true if CONTINUE, FIND, LOCATE or SEEK is successful.

Key Commands and Functions	
Command or Function	Description
GATHER	Copies the contents of memory variables or array elements to fields.
IF ... ENDIF	Conditionally processes commands.
INDEX	Creates an index file to order a table/.DBF.
INSERT INTO	Appends a record to an existing table/.DBF.
LABEL	Creates labels from a table/.DBF and a label file.
LOCATE	Locates a table/.DBF record.
LOCK()	Attempts to lock a table/.DBF record or records.
LOWER()	Returns a lower-case character string from a character expression.
MODIFY COMMAND	Opens a text editing window for a FoxPro program.
MODIFY FILE	Opens a text editing window for a text file.
MODIFY LABEL	Opens the Label Design window in FoxPro for MS-DOS or the Report Layout Window in FoxPro for Windows.
MODIFY MENU	Opens a Menu Design window.
MODIFY PROJECT	Opens a Project window.
MODIFY QUERY	Opens an RQBE window.
MODIFY REPORT	Opens the Report Layout window.
MODIFY SCREEN	Opens the Screen Design Window.
MODIFY STRUCTURE	Modifies the structure of a table/.DBF.
ON ERROR	Specifies a command that executes when an error occurs.
PACK	Removes all records marked for deletion, and reduces the size of a memo file.
PARAMETERS	Assigns data passed from a calling program to local memory variables.
PRIVATE	Hides specified memory variables or arrays that were defined in a calling program from the current program.

Key Commands and Functions	
Command or Function	Description
PROCEDURE	Identifies the beginning of a procedure in a program file.
QUIT	Exits FoxPro.
READ	Activates objects created with @ ... GETs and @ ... EDITS.
REPLACE	Updates table/.DBF records.
REPORT	Produces a report.
SCAN	Moves the record pointer through the current table/.DBF and executes a block of commands for each record.
SEEK	Searches the current indexed table/.DBF.
SELECT	Activates the specified work area.
SELECT — SQL	Retrieves data from one or more tables/.DBFs.
SET INDEX	Opens one or more index files for use with the current table/.DBF.
SET ORDER	Designates a controlling master index file or master tag.
SET RELATION	Creates a relationship between two or more tables/.DBFs.
SET SKIP	Creates one-to-many relationships between tables/.DBFs.
SET TALK	Determines whether or not FoxPro displays command results.
SHOW GETS	Redisplays all objects.
SKIP	Moves the record pointer.
SORT	Sorts a table/.DBF.
STORE	Stores data to a memory variable or a memory variable array.
UPPER	Returns an upper-case character string from a character expression.

Command and Function Categories

The table below shows the command and function categories, grouped by their general purpose, and the page number in this chapter where you can find each category and its associated commands and functions. For additional information about a specific command or function, see the Commands and Functions Alphabetically chapter.

Group	Categories	Page
Data Types	Character Functions	L2 - 7
	Numeric Functions	L2 - 10
	Date and Time Functions	L2 - 12
	Data Conversion	L2 - 13
Tables	Field Manipulation	L2 - 14
	Table/.DBF Manipulation	L2 - 15
	Record Manipulation	L2 - 17
	Indexes	L2 - 19
	Relations	L2 - 20
Programming	Debugging and Error Handling	L2 - 21
	Program Execution	L2 - 22
	Structured Programming	L2 - 24
	Low-Level File Functions	L2 - 24
	Memory Variable Manipulation	L2 - 25
	Array Manipulation	L2 - 27
	Event Handlers	L2 - 28
Input and Output	DDE and OLE	L2 - 29
	Data Formatting	L2 - 30
	Printing	L2 - 34
	Menu Bars and Popups	L2 - 35
	Windows	L2 - 37
	Typing, Keyboard and Mouse	L2 - 39
FoxPro Environment	Reports and Labels	L2 - 41
	Environment	L2 - 42
FoxPro in a multi-user environment	File Management	L2 - 47
	Network Commands & Functions	L2 - 49

Data Types

Character Functions

Functions in this category operate on character type data.

\$ – Determines whether a character string is contained inside another character string or memo field.

ALLTRIM() – Returns the specified character expression with the leading and trailing blanks removed.

ANSITOOEM() – In FoxPro for Windows, converts each character in a character expression to the corresponding character in the MS-DOS (OEM) character set.

ASC() – Returns the ASCII code for the leftmost character in a character string.

AT() – Returns the position from the left (as an integer) of the first occurrence of a character expression within another character expression or memo field.

ATC() – Returns the from the left position (as an integer) of the first occurrence of a character expression within another character expression or memo field. Unlike the AT() function, ATC() is not case sensitive.

ATCLINE() – Returns the line number of the first occurrence of a character expression within another character expression or memo field. ATCLINE() is not case sensitive.

ATLINE() – Returns the line number of the first occurrence of a character expression within another character expression or memo field.

BETWEEN() – Determines whether the value of an expression lies between the values of two other expressions of the same data type.

CHR() – Returns the character associated with the specified numeric ASCII code.

CHRTRAN() – Translates a character string using two other strings as translation tables.

CTOD() – Converts a character type date to a date type date.

DIFFERENCE() – Returns a number representing a relative phonetic difference between two character strings.

DTOC() – Converts a date to a character type date.

EMPTY() – Determines whether or not an expression is empty.

INLIST() – Determines whether or not an expression matches one in a series of expressions of the same data type.

INT() – Converts a number to its integer value by discarding the decimal portion.

ISALPHA() – Returns true (.T.) if the leftmost character in the specified character expression is an alphabetical character.

ISDIGIT() – Returns true (.T.) if the leftmost character of the specified character expression is a digit (0 - 9).

ISLOWER() – Returns true (.T.) if the leftmost character in the specified character expression is a lower-case alphabetic character.

ISUPPER() – Returns true (.T.) if the leftmost character in a character expression is an upper-case alphabetic character.

LEFT() – Returns a specified number of characters, starting from the leftmost character in a character string.

LEN() – Returns the number of characters in a character expression.

LIKE() – Determines whether or not a character expression, which can contain wildcard characters, matches another character expression.

LOWER() – Returns a specified character expression in all lower-case letters.

LTRIM() – Returns the specified character expression with leading blanks removed.

MAX() – Returns the expression with the highest ASCII or numeric value, or latest date, from a list of character, numeric or date expressions.

MIN() – Returns the expression with the lowest ASCII or numeric value, or earliest date, in a series of character, numeric or date expressions.

OCCURS() – Returns the number of occurrences of a character string within another character string.

OEMTOANSI() – In FoxPro for Windows, converts each character in a character expression to the corresponding character in the ANSI character set.

PADC() – Returns the specified expression padded on both sides with a specified character.

PADL() – Returns the specified expression padded on the left with a specified character.

PADR() – Returns the specified expression padded on the right with a specified character.

PROPER() – Returns the specified character expression with the initial letter of each word upper-case and the remaining characters lower-case.

RAT() – Searches a character string or memo field for the last occurrence of another character string and returns as an integer the position where it was found.

RATLINE() – Searches a character string or memo field for the last occurrence of another character string and returns as an integer the line number where it was found. The value returned is dependent upon the value specified by the SET MEMOWIDTH command.

REPLICATE() – Returns a character string that contains a specified character expression repeated a given number of times.

RIGHT() – Returns the specified number of characters from the rightmost position in a character string.

RTRIM() – Returns the specified character expression with all trailing blanks removed.

SOUNDEX() – Returns a phonetic code representing a character expression.

SPACE() – Returns a character string composed of a specified number of spaces.

STR() – Returns the character string equivalent of a numeric expression.

STRTRAN() – Searches a character expression for occurrences of a second character expression, then replaces each occurrence with a third character expression.

STUFF() – Returns a character string resulting from the replacement of a specified portion of one character expression with another character expression.

SUBSTR() – Returns a specified number of characters from the given expression or memo field.

SYS(10) – Returns a character type date from a numeric day number.

SYS(15) – Returns a character from a translation table.

SYS(20) – Transforms a character expression containing German text to a character string.

SYS(2007) – Returns the check sum value of a character string.

TRANSFORM() – Formats a character or numeric expression using PICTURE and FUNCTION codes.

TRIM() – Returns the specified character expression with all trailing blanks removed.

TXTWIDTH() – Returns the length of a character expression in a specific font in FoxPro for Windows.

TYPE() – Returns the data type of the expression (character, numeric, logical, date or memo).

UPPER() – Returns the specified character expression in upper-case.

Numeric Functions

You use the commands and functions in this category to operate on and/or return numeric data.

% – Returns the remainder obtained by dividing a numeric expression by another numeric expression.

ABS() – Returns the absolute value of a specified numeric expression.

ACOS() – Returns the arc cosine of a specified numeric expression.

ASIN() – Returns the arc sine of a specified numeric expression.

ATAN() – Returns the arc tangent of a specified numeric expression.

ATN2() – Returns the arc tangent from the specified sine and cosine values.

BETWEEN() – Determines whether or not the value of an expression lies between the values of two other expressions of the same data type.

CALCULATE – Performs financial and statistical operations.

CEILING() – Returns the nearest integer that is greater than or equal to the specified numeric expression.

COS() – Returns the cosine of an angle.

DTOR() – Converts degrees to radians.

EMPTY() – Determines whether or not a numeric expression equals zero.

EXP() – Returns the value of e^x , where x is the specified numeric expression.

FLOOR() – Returns the nearest integer less than or equal to the specified numeric expression.

FV() – Returns the future value of an investment.

INLIST() – Determines whether or not an expression matches one in a series of expressions of the same data type.

INT() – Returns the integer portion of a numeric expression.

LOG() – Returns the natural logarithm (base e) of the specified numeric expression.

LOG10() – Returns the common logarithm (base 10) of the specified numeric expression.

MAX() – Returns the expression with the highest ASCII or numeric value, or latest date, from a list of character, numeric or date expressions.

MIN() – Returns the expression with the lowest ASCII or numeric value, or the earliest date, in a series of character, numeric or date expressions.

MOD() – Returns the remainder from a division operation.

PAYMENT() – Returns the amount of each periodic payment on a fixed interest loan.

PI() – Returns the numeric constant π .

PV() – Returns the present value of an investment.

RAND() – Returns a random number between 0 and 1.

ROUND() – Returns a numeric expression rounded to a specified number of decimal places.

RTOD() – Converts radians to degrees.

SET DECIMALS – Specifies the number of decimal places displayed in numeric results.

SIGN() – Returns -1 if the numeric expression is negative or 1 if the numeric expression is positive.

SIN() – Returns the sine of an angle.

SQRT() – Returns the square root of the specified numeric expression.

TAN() – Returns the tangent of an angle.

VAL() – Returns a numeric expression from a specified character expression composed of numbers.

Date and Time Functions

You use the functions in this section to generate and manipulate date and time data.

BETWEEN() – Determines whether or not the value of an expression lies between the values of two other expressions of the same data type.

CDOW() – Returns the day of the week from a given date expression.

CMONTH() – Returns the name of the month for a given date.

CTOD() – Returns a date type date for a given character type date.

DATE() – Returns the current system date.

DAY() – Returns the numeric day of the month for a given date.

DMY() – Returns a date expression in Day Month Year format.

DOW() – Returns the numeric day of the week for a given date.

DTOC() – Returns a character type date for a specified date expression.

DTOS() – Returns a character string date in YYYYMMDD format for a specified date expression.

EMPTY() – Determines whether or not an expression is empty.

GOMONTH() – Returns the date that is a specified number of months before or after a given date.

INLIST() – Determines whether or not an expression is contained in a series of expressions of the same data type.

MAX() – Returns the expression with the highest ASCII or numeric value, or latest date, from a list of character, numeric or date expressions.

MDY() – Returns the specified date expression in Month Day Year format.

MIN() – Returns the expression with the lowest ASCII or numeric value, or earliest date, from a list of character, numeric or date expressions.

MONTH() – Returns the numeric month for a given date.

SECONDS() – Returns the number of seconds elapsed since midnight in the format seconds.thousandths.

SET HOURS – Sets the system clock to a 12- or 24-hour format.

SET MARK TO – Specifies a delimiter in the display of date expressions.

SYS(1) – Returns the Julian day number for the current system date.

SYS(2) – Returns the number of seconds elapsed since midnight.

SYS(10) – Returns a character type date from a specified Julian day number.

SYS(11) – Returns a Julian day number equivalent to a given date.

TIME() – Returns the current system time.

YEAR() – Returns the year from the specified date expression.

Data Conversion

You use the functions in this category to convert data from one type to another.

ANSITOOEM() – In FoxPro for Windows, converts each character in a character expression to the corresponding character in the MS-DOS (OEM) character set.

CHR() – Returns the character corresponding to the specified ASCII number.

CTOD() – Returns a date type date from a specified character type date.

OEMTOANSI() – In FoxPro for Windows, converts each character in a character expression to the corresponding character in the ANSI character set.

SYS(10) – Returns a character type date from a specified Julian day number.

SYS(15) – Returns a character from a translation table.

TRANSFORM() – Formats character and numeric expressions using PICTURE and FUNCTION codes.

Tables/.DBFs

Field Manipulation

You use the commands and functions in this section to manipulate table/.DBF fields.

AFIELDS() – Places table/.DBF structure information into an array.

DELETED() – Returns true (.T.) if the current record is marked for deletion.

FCOUNT() – Returns the number of fields in the current or a specified table/.DBF.

FIELD() – Returns the name of a field, referenced by number, in the current or specified table/.DBF.

FSIZE() – Returns the size in bytes of a specified field.

GATHER – Stores the contents of memory variables or array elements to fields.

LUPDATE() – Returns the date a table/.DBF was last updated.

NDX() – Returns the name of an open .IDX index file for a table/.DBF in the selected or a specified work area.

SCATTER – Copies data from the current record to an array or a set of memory variables.

SET BLOCKSIZE – Specifies how FoxPro allocates disk space for the storage of memo fields.

SET EXACT – Specifies whether an exact character for character match of two strings is necessary in order for FoxPro to consider the strings equal.

SET TEXTMERGE – Enables or disables evaluation of table/.DBF fields, memory variables, functions and expressions that are surrounded by text merge delimiters.

SET TEXTMERGE DELIMITERS – Specifies the text merge delimiters.

SYS(14) – Returns the key expression of an index file or index tag.

Table/.DBF Manipulation

You use the commands and functions in this section to create, manipulate and monitor tables/.DBFs.

ALIAS() – Returns the alias of a specified work area.

APPEND – Enables you to add records to the end of a table/.DBF.

APPEND FROM – Adds records to the end of a table/.DBF from another file.

AVERAGE – Computes the arithmetic mean of numeric expressions or fields.

BOF() – Returns true (.T.) if the record pointer is positioned at the beginning of a table/.DBF.

BROWSE – Opens a Browse window so you can view and/or edit table/.DBF records.

CHANGE or **EDIT** – Displays fields for editing.

CLEAR FIELDS – Releases all fields created with SET FIELDS TO.

CLOSE DATABASES – Closes all open table/.DBF, index, memo and format files.

CLOSE MEMO – Closes memo editing windows.

COPY MEMO – Copies the contents of a memo field to a text file.

COPY STRUCTURE – Copies the structure of a table/.DBF to another table/.DBF.

COPY STRUCTURE EXTENDED – Copies information about the structure of one table/.DBF into records in a new table/.DBF.

COPY TO – Copies the contents of the current table/.DBF to another table/.DBF.

COUNT – Counts specified records.

CREATE – Creates a new table/.DBF.

CREATE FROM – Creates a new table/.DBF whose structure is obtained from a COPY STRUCTURE EXTENDED table/.DBF.

CREATE QUERY – Opens the RQBE window so you can create a query.

CREATE TABLE – Creates a table/.DBF with specified fields.

DBF() – Returns the path and name of the table/.DBF open in the specified work area.

DELETE – Marks records for deletion.

DELETED() – Returns true (.T.) if the current record is marked for deletion.

DISPLAY STRUCTURE – Displays the structure of a table/.DBF.

EOF() – Returns true (.T.) if the record pointer is positioned at the end of a table/.DBF.

EXPORT – Copies data from a FoxPro table/.DBF to a file with a different format.

FLUSH – Saves table/.DBF and index modifications to disk.

IMPORT – Imports data from a different file format to create a new FoxPro table/.DBF.

JOIN – Creates a table/.DBF by joining two existing tables/.DBFs.

MEMLINES() – Returns the number of lines in a memo field.

MLINE() – Returns a specific line from a memo field as a character string.

MODIFY GENERAL – Opens an editing window for a general field in FoxPro for Windows.

MODIFY MEMO – Opens an editing window for a memo field.

MODIFY QUERY – Opens the RQBE window in which you can modify or create a query.

MODIFY STRUCTURE – Allows you to modify the current table/.DBF structure.

PACK – Removes all records marked for deletion and reduces the size of a memo file.

RECALL – Unmarks records that are marked for deletion.

REPLACE – Updates one or more fields with data from one or more expressions.

SELECT – Activates the specified work area.

SELECT – SQL – Retrieves data from one or more tables/.DBFs.

SET ANSI – Specifies how SQL string comparisons are made.

SET AUTOSAVE – Determines whether or not FoxPro flushes data buffers to disk when you exit a READ or return to the Command window.

SET DELETED – Specifies whether or not FoxPro processes records marked for deletion.

SET FIELDS – Specifies access to all fields or only those fields included in a field list.

SET OPTIMIZE – Enables or disables Rushmore optimization.

SORT – Sorts records in the current table/.DBF and outputs the sorted data to a new table/.DBF.

SUM – Totals all or specified numeric field values in the current table/.DBF.

TOTAL – Creates summary records in a new table/.DBF, with one record for each group of identically-keyed records in the current table/.DBF.

UPDATE – Updates fields in the current table/.DBF with data from another table/.DBF.

USE – Opens a table/.DBF and associated index files.

USED() – Returns true (.T.) if a table/.DBF is open in a specified work area.

ZAP – Removes all records from a table/.DBF.

Record Manipulation

You use the commands and functions in this section to select records within a table/.DBF or position the record pointer.

APPEND – Adds a record to the end of a table/.DBF.

APPEND FROM – Adds records to a table/.DBF from another file.

APPEND GENERAL – Imports OLE objects into a general field.

APPEND MEMO – Copies the contents of a file to a memo field.

BOF() – Returns true (.T.) if the record pointer is positioned at the beginning of a table/.DBF.

CONTINUE – Continues the previous LOCATE.

EOF() – Returns true (.T.) if the record pointer is positioned at the end of a table/.DBF.

FILTER() – Returns the filter expression specified by SET FILTER for the current or specified table/.DBF.

FOUND() – Returns true (.T.) if most recent CONTINUE, FIND, LOCATE or SEEK was successful.

GO *or*

GOTO – Positions the record pointer at the specified record number.

INSERT – Inserts a new record into a table/.DBF.

INSERT – SQL – Appends a record to the end of a table/.DBF.

LOCATE – Sequentially searches the selected table/.DBF for the first record that matches a given expression.

LOOKUP() – Searches a table/.DBF for the first record with a field matching the specified expression and, if it is found, moves the record pointer to the record and returns the value of a specified field in that record.

RECCOUNT() – Returns the number of records in a table/.DBF.

RECNO() – Returns the current record number.

RECSIZE() – Returns the size of a table/.DBF record.

SEEK – Searches the current indexed table/.DBF for the first occurrence of a record whose index key expression matches the specified expression, and positions the record pointer on the record if it is found.

SEEK() – Searches the current indexed table/.DBF for the first occurrence of a record whose index key expression matches the specified expression, and positions the record pointer on the record. Returns true (.T.) if the search was successful.

SET FILTER – Specifies a condition that records in the selected table/.DBF must meet to be accessible.

SET NEAR – Determines where the record pointer is positioned after SEEK unsuccessfully searches for a record.

SET RELATION – Links or unlinks two or more open tables/.DBFs.

SET SKIP – Set a one-to-many relation between two related tables/.DBFs.

SKIP – Moves the record pointer forward or backward in the current or specified table/.DBF.

Indexes

You use the commands and functions in this section to perform operations on and return information about index files.

CDX() – Returns the names of open .CDX compound index files.

CLOSE INDEX – Closes all open index files in the selected work area.

COPY INDEXES – Copies single-entry index files to a compound index file.

COPY TAG – Creates a single-entry index file from a tag in a compound index file.

DELETE TAG – Removes a tag or tags from a .CDX compound index file.

INDEX – Creates an index file to display a table/.DBF in logical order.

KEY() – Returns the key index expression of the master (controlling) index file.

MDX() – Returns the names of open .CDX compound index files.

NDX() – Returns the name of an open index in a specified work area.

ORDER() – Returns the name of the controlling index file or tag for the current or specified table/.DBF.

REINDEX – Rebuilds all open index files for the current table/.DBF.

SET INDEX – Selects one or more index files for use with the current table/.DBF.

SET ORDER – Designates a controlling index file or tag for the current or specified table/.DBF.

SET UNIQUE – Specifies whether or not records with duplicate index key values appear in an index file.

SYS(14) – Returns the index expression of an index file.

SYS(21) – Returns the index position number of the controlling index for the current table/.DBF.

SYS(2021) – Returns the index filter expression for a single-entry index file or a tag, for the current or specified table/.DBF.

TAG() – Returns a tag name from a .CDX compound index file, or the name of an .IDX single-entry index file.

Relations

You use the commands and functions in this section to establish or break relations between tables/.DBFs.

RELATION() – Returns the relational expression for the current or specified table/.DBF.

SELECT – SQL – Retrieves data from one or more tables/.DBFs.

SET RELATION – Links or unlinks two or more open tables/.DBFs.

SET RELATION OFF – Breaks an established relation between the current table/.DBF and another open table/.DBF.

SET SKIP – Creates one-to-many relationships between related tables/.DBFs.

TARGET() – Returns the alias of a table/.DBF that is the target for a relation.

Programming

Debugging and Error Handling

You use the commands and functions in this section to perform various debugging operations for FoxPro programs.

ERROR() – Returns the error number for an error that triggered an ON ERROR routine.

EXTERNAL – Alerts the Project Manager to an undefined reference.

LINENO() – Returns the line number of the line being executed in a program.

MESSAGE() – Returns the current error message or the contents of the program line that caused the error.

NOTE | * | && – Indicates non-executing comments in a program.

ON ERROR – Specifies a command that executes when an error occurs.

ON READERROR – Specifies a command that executes in response to a data input error.

ON() – Returns the command assigned to ON ERROR, ON ESCAPE, ON KEY, ON KEY LABEL, ON PAGE or ON READERROR.

PROGRAM() – Returns the name of the program currently being executed or the name of the program that was executing when the last error occurred.

RETRY – Returns control to the calling program and re-executes the most recently executed command in that program.

SET DEBUG – Enables or disables access to the Debug and Trace windows.

SET DEVELOPMENT – Causes FoxPro to compare the creation date and time of a program file with those of its compiled object file when you run the program.

SET ECHO – Activates the Trace window for program debugging.

SET TALK – Determines whether or not FoxPro displays command processing information.

SET TRBETWEEN – Enables or disables tracing between breakpoints in the Trace window.

SYS(2018) – Returns the error message parameter for the most recent error.

VARREAD() – Returns the name of the current memory variable, array element or field used to create the current @ ... GET field or control.

WAIT – Halts FoxPro operation until you press a key or click the mouse.

Program Execution

You use the commands and functions in this section to perform actions associated with executing, suspending and terminating FoxPro programs.

BUILD APP – Creates an application .APP file from a project file.

BUILD EXE – For owners of the FoxPro Distribution Kit, creates an executable (.EXE) file from a project.

BUILD PROJECT – Creates and builds a project file.

CALL – Executes a binary file that has been placed in memory with LOAD.

CANCEL – Cancels execution of the currently running program.

CLEAR PROGRAM – Clears the compiled program buffer.

CLOSE ALL – Closes all files in all work areas and selects work area 1.

CLOSE PROCEDURE – Closes the file you opened with SET PROCEDURE.

COMPILE – Compiles one or more program files and creates an object file for each source file.

CREATE PROJECT – Opens the Project window and enables you to create a project file.

#DEFINE ... #UNDEF – Creates and releases compile-time constants.

DO – Executes a command or procedure file.

FUNCTION – Defines the start of a routine.

#IF ... #ENDIF – Conditionally includes or excludes source code at compile time.

LOAD – Places a binary file in memory to be executed with CALL.

MODIFY PROJECT – Opens the Project window and enables you to modify or create a project file.

PARAMETERS – Assigns data passed from a calling program to local memory variables.

PARAMETERS() – Returns the number of parameters that were passed to the most recently called procedure.

PROCEDURE – Identifies the beginning of a procedure within a program file.

RELEASE LIBRARY – Closes an external library file.

RELEASE MODULE – Releases a loaded binary file from memory.

RESUME – Resumes the execution of a suspended program.

RETURN – Returns control to the calling program.

RUN ! ! – Runs an external command or application from within FoxPro.

SET ESCAPE – Determines whether or not pressing the Escape key interrupts program and command execution.

SET LIBRARY – Opens an external library file.

SET LOGERRORS – Determines whether or not FoxPro sends compilation error messages to a text file.

SET PROCEDURE – Opens a procedure file.

SET SYSMENU – Enables or disables access to the FoxPro system menu bar during program execution.

SET UDFPARMS – Specifies whether parameters are passed to a user-defined function (UDF) by value or reference.

SUSPEND – Suspends program execution.

SYS(2015) – Returns a unique procedure name.

WAIT – Halts FoxPro operation until you press a key or click the mouse.

Structured Programming

You use the command and functions in this section to control program execution and branching within a FoxPro program.

DO CASE ... ENDCASE – Executes the first statement block after DO CASE and before ENDCASE whose associated conditional statement evaluates to true (.T.).

DO WHILE ... ENDDO – Executes a block of statements within a conditional loop.

EXIT – Exits a DO WHILE, FOR ... ENDFOR or SCAN ... ENDSCAN loop.

FOR ... ENDFOR – Executes statements in a loop a specified number of times.

#IF ... #ENDIF – Conditionally includes or excludes source code at compile time.

IF ... ENDIF – Conditionally executes a set of commands based on the outcome of a logical expression.

IIF() – Returns one of two values depending on the value of a logical expression.

SCAN ... ENDSCAN – Moves the record pointer through the current table/.DBF and executes a block of commands for each record that meets the specified conditions.

Low-Level File Functions

You use the following functions to create and manipulate files and communication ports at a low level. Low-level file functions should be used carefully, especially when manipulating tables/.DBFs.

FCHSIZE() – Changes the size of a file opened with a low-level file function.

FCLOSE() – Closes a file or communication port opened with a low-level file function.

FCREATE() – Creates and opens a low-level file.

FEOF() – Determines whether or not the file pointer is positioned at the end of a file.

FERROR() – Returns 0 if the last low-level function was successfully executed; otherwise, returns a value corresponding to the cause of the error.

FFLUSH() – Flushes to disk a file opened with a low-level file function.

FGETS() – Returns a series of bytes from a file or a communication port opened with a low-level file function.

FOPEN() – Opens a file or communication port for use with low-level file functions.

FPUTS() – Writes a character string, carriage return and line feed to a file or a communication port opened with a low-level file function.

FREAD() – Returns as a character string a specified number of bytes from a file or a communication port.

FSEEK() – Moves the file pointer in a file opened with a low-level file function.

FWRITE() – Writes a character string to a file or communication port opened with a low-level file function.

HEADER() – Returns the number of bytes in the header of the current or specified table/.DBF.

Memory Variable Manipulation

You use the commands and functions in this section to assign values to memory variables.

& – Performs macro substitution on the memory variable it precedes.

= – Assigns the value on the right of the sign to the memory variable on the left.

ACOPY() – Copies elements from one array to another array.

ADEL() – Deletes an element, row or column from an array.

ADIR() – Places information about specified files into an array.

AELEMENT() – Returns an array element's number from its row and column subscripts.

AFIELDS() – Places table/.DBF structure information into an array.

AINS() – Inserts an element, row or column into an array.

ALEN() – Returns the number of elements, rows or columns in an array.

ASCAN() – Searches an array for an expression.

ASORT() – Sorts elements in an array in ascending or descending order.

ASUBSCRIPT() – Returns a row or column element from an array element number.

AVERAGE – Computes the arithmetic mean of numeric expressions or fields.

CLEAR ALL – Releases from memory all memory variables and all user-defined menu bars, popups and windows, closes all tables/.DBFs and associated files, and selects work area 1.

CLEAR MEMORY – Removes all public and private memory variables from memory.

COUNT – Counts the number of records that meet the specified conditions, optionally storing the result in a memory variable.

DISPLAY *or*

LIST MEMORY – Displays the current contents of memory variables.

GETEXPR – Invokes the Expression Builder dialog and stores the user-constructed expression to a memory variable.

PARAMETERS – Identifies variables passed from a calling program.

PRIVATE – Makes previously defined memory variables unavailable to the currently executing program.

PUBLIC – Defines global memory variables.

REGIONAL – Creates regional memory variables.

RELEASE – Releases memory variables, libraries, or the definitions of user-defined menus, menu pads, popups, popup options and windows.

RESTORE FROM – Restores memory variables from a memory variable file or a memo field.

SAVE TO – Saves memory variables to a memory variable file or a memo field.

STORE – Assigns a value to a memory variable or initializes an array.

SUM – Totals all or specified numeric field values in the current table/.DBF.

WAIT – Halts FoxPro operation until you press a key or click the mouse.

Array Manipulation

You use the commands and functions in this category to work with arrays.

ACOPY() – Copies elements from one array to another array.

ADEL() – Deletes an element, row or column from an array.

ADIR() – Places information about specified files into an array.

AELEMENT() – Returns the number of an array element from the element's subscripts.

AFIELDS() – Places table/.DBF structure information into an array.

AFONT() – In FoxPro for Windows, places information about available fonts into an array.

AINS() – Inserts an element, row or column into an array.

ALEN() – Returns the number of elements, rows or columns in an array.

APPEND FROM ARRAY – Adds records to the current table/.DBF and fills the records with data from an array.

ASCAN() – Searches an array for an expression.

ASORT() – Sorts a memory variable array in ascending or descending order.

ASUBSCRIPT() – Returns a row or column element from an array element number.

COPY TO ARRAY – Copies data from the current table/.DBF to an array.

DECLARE – Creates one or more arrays.

DIMENSION – Creates one or more arrays.

GATHER – Stores the contents of memory variables or array elements to fields.

INSERT FROM ARRAY – Inserts records into the current table/.DBF immediately after the selected record and fills them with data from an array.

SCATTER – Copies data from the current record to an array or a set of memory variables.

Event Handlers

These commands and functions specify the action FoxPro takes when a given event occurs.

ON ERROR – Specifies a command that executes when an error occurs.

ON ESCAPE – Specifies a command that executes when you press Escape during program and command execution.

ON KEY – Specifies a command that executes when you press any key during program execution.

ON KEY LABEL – Specifies a command that executes when you press a specific key.

ON PAD – Assigns a routine to a menu pad on a menu bar.

ON SELECTION PAD – Specifies a command that executes when you choose a specific menu bar pad.

ON SELECTION POPUP – Assigns a routine to a menu popup.

ON() – Returns the command assigned to ON ERROR, ON ESCAPE, ON KEY, ON KEY LABEL, ON PAGE or ON READERROR.

DDE and OLE

You use the commands and functions in the section, available only in FoxPro for Windows, to operate with other Windows-based applications.

@ ... SAY – Displays an OLE object.

APPEND GENERAL – Imports OLE objects into a general field.

DDEAbortTrans() – Aborts an asynchronous DDE transaction.

DDEAdvise() – Creates a DDE notify link or an automatic link.

DDEEnabled() – Enables or disables DDE processing or returns its status.

DDEExecute() – Sends a command to another application.

DDEInitiate() – Establishes a channel between FoxPro for Windows and another Windows-based application.

DDELastError() – Returns an error number for the last DDE function.

DDEPoke() – Sends data between client and server applications.

DDERequest() – Requests data from a server application.

DDESetOption() – Changes or returns DDE settings.

DDESetService() – Creates, releases or modifies service names and settings.

DDESetTopic() – Creates or releases a topic name for a service name.

DDETerminate() – Closes a channel established with DDEInitiate().

MODIFY GENERAL – Opens a general field editing window.

RUN | ! – Runs an external command or application from within FoxPro.

Input and Output

Data Formatting

You use these commands and functions to format and display data from records, fields, memory variables or arrays. Entries in this section also support general display features.

**** – Outputs the line of text that follows \ or \\ to the text merge file specified with SET TEXTMERGE TO and/or to the screen.

? – Displays the value of the specified expression on the next line.

?? – Displays the value of the specified expression at the current position on the current line.

@ ... SAY – Formats and displays a specified expression, field, memory variable or array element.

@ ... GET – Creates an editing region for a field, memory variable or array element.

@ ... GET – Check Boxes – Creates check boxes.

@ ... GET – Invisible Buttons – Creates invisible buttons.

@ ... GET – Lists – Creates lists.

@ ... GET – Popups – Creates menu popups.

@ ... GET – Push Buttons – Creates push buttons.

@ ... GET – Radio Buttons – Creates radio buttons.

@ ... GET – Spinners – Creates spinners in FoxPro for Windows.

@ ... Edit – Text Edit Regions – Creates text editing regions.

@ ... BOX – Draws a box using specified coordinates.

@ ... CLEAR – Clears a portion of the screen or active window.

@ ... FILL – Changes colors within an area of the screen or a window.

ACTIVATE SCREEN – Sends all subsequent output to the screen instead of to the current window.

AFONT() – Places information about available Windows fonts into an array.

APPEND – Adds a record to the end of a table/.DBF.

BROWSE – Opens a Browse window so you can view and/or edit table/.DBF records.

CHANGE *or*

EDIT – Displays fields for editing.

CLEAR – Erases the contents of the screen or current output window.

CLEAR GETS – Releases all pending GET statements.

CLEAR READ – Exits the active READ.

CLOSE FORMAT – Closes the format file open in the current work area.

COL() – Returns the current column position of the cursor.

CREATE SCREEN – Opens the Screen Design window and enables you to create a screen.

DISPLAY *or*

LIST – Displays records from a table/.DBF.

FONTMETRIC() – Returns Windows font attributes.

INKEY() – Returns a number corresponding to the first key or mouse click in the typeahead buffer.

MODIFY SCREEN – Displays the Screen Design window and enables you to modify or create a screen.

OBJNUM() – Returns the object number of the specified @ ... GET object.

RDLEVEL() – Returns the current READ level.

READ – Activates @ ... GET objects and @ ... EDIT objects for editing.

READKEY() – Returns an integer that represents the key pressed by the user in order to exit an editing command.

ROW() – Returns the current row position of the cursor.

SCOLS() – Returns the number of columns available on the screen.

SCROLL – Scrolls an area of a user-defined window, the desktop (-FoxPro for MS-DOS) or the main FoxPro window (FoxPro for Windows) up, down, left or right.

SET BLINK – Blinks screen elements or increases the number of colors available in FoxPro for MS-DOS.

SET CARRY – Determines whether or not FoxPro carries data forward from the current record to the new record created with APPEND or INSERT.

SET CLEAR – Determines whether or not SET FORMAT TO and QUIT clear the screen.

SET COLOR – Controls the color of the desktop (FoxPro for MS-DOS), the main FoxPro window (FoxPro for Windows), windows, dialogs, and so on.

SET CONFIRM – Specifies whether or not Enter or tab must be pressed to exit an input field.

SET CONSOLE – Enables or disables output to the desktop (FoxPro for MS-DOS) or the main FoxPro window (FoxPro for Windows).

SET CURSOR – In FoxPro for MS-DOS, determines whether or not the cursor displays during a pending @ ... GET, INKEY(), or WAIT.

SET DECIMALS – Sets the number of decimal places displayed in certain output operations.

SET DEVICE – Directs the output of @ ... SAY commands to the screen, a printer or a file.

SET FIXED – Fixes the number of decimal places in the display of numeric data.

SET HEADINGS – Determines whether or not column titles are displayed for each field in the DISPLAY, LIST, SUM and AVERAGE commands.

SET MARK TO – Specifies a delimiter in the display of date expressions.

SET MEMOWIDTH – Sets the width of memo field output for LIST and DISPLAY and sets the default width for REPORT.

SET MESSAGE – Defines a message for display at the bottom of the desktop in FoxPro for MS-DOS or in the status bar in FoxPro for Windows. Also specifies the location of messages in menu bar and popup commands.

SET ODOMETER – Determines the reporting interval of the record counter for commands that process records.

SET PALETTE – Specifies if the FoxPro for Windows color palette is used.

SET SPACE – Determines whether or not a space displays between fields or expressions when you use the ? or ?? commands.

SET TEXTMERGE – Enables or disables evaluation of fields, memory variables, functions and expressions that are surrounded by text merge delimiters.

SET TEXTMERGE DELIMITERS – Specifies the delimiters for text merge.

SHOW GET – Redisplays a single specified @ ... GET object.

SHOW GETS – Redisplays all @ ... GET objects.

SHOW OBJECT – Redisplays the @ ... GET object that has the specified object number.

SROWS() – Returns the number of rows available on the screen.

SYS(100) – Returns the current SET CONSOLE setting.

SYS(101) – Returns the current SET DEVICE setting.

SYS(2002) – In FoxPro for MS-DOS, turns the cursor on or off.

TEXT ... ENDTEXT – Outputs lines of text, the results of expressions and functions and the contents of memory variables.

TRANSFORM() – Formats character and numeric expressions using PICTURE and FUNCTION codes.

UPDATED() – Returns true (.T.) if you changed any data during the most recent READ.

Printing

These commands and functions print or display data from records or memory variables on the screen to a printer or to an output file.

**\ | ** – Outputs the line of text that follows \ or \| to the text merge file specified with SET TEXTMERGE and/or to the screen.

? – Displays the value of the specified expression on the next line.

?? – Displays the value of the specified expression at the current position on the current line.

??? – Sends output directly to the printer, bypassing the printer driver.

@ ... SAY – Formats and displays a specified expression at a specified row and column position.

CLOSE ALTERNATE – Closes an open ALTERNATE file.

DEFINE BOX – Draws a box around printed text.

DISPLAY *or*

LIST – Displays the contents of records in the current table/.DBF and the results of expressions.

EJECT – Sends a form feed to the printer.

EJECT PAGE – Sends a conditional page advance to the printer.

PCOL() – Returns the current column position of printer output.

PRINTSTATUS() – Returns true (.T.) if the printer is ready or false (.F.) if it is not.

PROW() – Returns the current row position of the printer output.

SET MARGIN – Specifies the left margin in printed output.

SET PDSETUP – Loads a printer driver setup.

SET PRINTER – Enables or disables output to the printer and routes printer output to a file or to a port.

SYS(6) – Returns the current SET PRINTER TO setting.

SYS(13) – Returns the status of the printer.

SYS(102) – Returns the current SET PRINTER ON | OFF setting.

TYPE – Displays the contents of an ASCII file.

Menu Bars and Popups

You use these commands and functions to develop, display and activate user-defined menu bars and popups.

ACTIVATE POPUP – Displays and activates a popup created with **DEFINE POPUP** or a FoxPro menu system popup.

BAR() – Returns the number of the most recently chosen option from a popup defined with **DEFINE POPUP** or a FoxPro menu system popup.

CLEAR MENUS – Releases all menu bars from memory and clears them from the desktop (FoxPro for MS-DOS), the main FoxPro window or a user-defined window.

CLEAR POPUPS – Releases all popups from memory and clears an active popup from the screen.

CNTBAR() – Returns the number of bars on a popup defined with **DEFINE POPUP** or on a FoxPro menu system popup.

CNTPAD() – Returns the number of pads on a user-defined menu bar or the FoxPro system menu bar.

CREATE MENU – Displays the Menu Design window and enables you to create a menu system.

DEACTIVATE MENU – Deactivates a user-defined menu bar and removes it from the screen, but does not remove the menu bar definition from memory.

DEACTIVATE POPUP – Deactivates a popup defined with **DEFINE POPUP** and removes it from the screen, but does not remove the popup definition from memory.

DEFINE BAR – Creates a menu option on a popup defined with **DEFINE POPUP** or a FoxPro menu system popup.

DEFINE MENU – Creates a menu bar.

DEFINE PAD – Creates a pad on a user-defined menu bar or the FoxPro system menu bar.

DEFINE POPUP – Creates a popup.

GETBAR() – Returns the number of bars on a popup defined with **DEFINE POPUP** or a FoxPro menu system popup.

GETPAD() – Returns the name of a menu pad from its position on a user-defined menu bar or on the FoxPro system menu bar.

MODIFY MENU – Displays the menu design window and enables you to modify or create a menu system.

MOVE POPUP – Moves a popup to a new location.

MRKBAR() – Returns whether or not a bar in a popup defined with **DEFINE POPUP** or in a FoxPro menu system popup is marked with **SET MARK OF**.

MRKPAD() – Determines whether or not a pad on a user-defined menu bar or on the FoxPro system menu bar is marked with **SET MARK OF**.

ON BAR – Activates a popup or menu bar when an option on a popup is chosen.

ON PAD – Activates a popup or menu bar when a menu pad is chosen.

ON SELECTION BAR – Specifies a command that executes when you choose a bar on a popup.

ON SELECTION MENU – Specifies a command that executes when you choose any pad on a menu bar defined with **DEFINE MENU**.

ON SELECTION PAD – Specifies a command that executes when you choose a specific pad on a user-defined menu bar or on the FoxPro system menu bar.

ON SELECTION POPUP – Specifies a command that executes when you choose any bar on a popup defined with **DEFINE POPUP** or a popup in the FoxPro menu system.

PAD() – Returns the name of the menu pad that was last chosen from a menu bar.

POP MENU – Pulls the specified user-defined menu bar or FoxPro system menu bar off the stack.

POP POPUP – Pulls the specified popup off the stack.

POPUP() – Returns the name of the active popup.

PRMBAR() – Returns the text from an option on a popup.

PRMPAD() – Returns the text from a menu pad.

PROMPT() – Returns the option chosen from an active menu bar or popup.

PUSH MENU – Places a menu bar definition on the stack.

PUSH POPUP – Places a popup definition on the stack.

RELEASE BAR – Removes one or all bars from a popup.

RELEASE MENUS – Releases all or specified user-defined menu bar definitions from memory.

RELEASE PAD – Removes specified pads from a user-defined or the FoxPro system menu bar.

RELEASE POPUPS – Releases all or specified popup definitions established with **DEFINE POPUP** from memory.

SET BORDER – Defines a border for popups created with **DEFINE POPUP** and for windows created with **DEFINE WINDOW**.

SET MARK OF – Specifies a mark character for a user-defined menu pad or a bar on a popup defined with **DEFINE POPUP**.

SET SKIP OF – Enables or disables a menu bar, menu pad, popup, or popup bar, for a user-defined menu or the FoxPro system menu.

SIZE POPUP – Changes the size of a popup.

SKPBAR() – Determines whether a menu bar is enabled or is disabled with **SET SKIP OF**.

SKPPAD() – Determines whether a menu pad is enabled or disabled with **SET SKIP OF**.

SHOW MENU – Displays a user-defined menu bar without activating it.

SYS(2013) – Returns the names of the FoxPro system menu bar, each pad in the system menu bar, menu system popups and each option in the menu system popups.

Windows

You use these commands and functions to develop, display and activate user-defined windows.

ACTIVATE WINDOW – Displays and activates one or more user-defined windows or FoxPro system windows.

CLEAR WINDOWS – Releases all user-defined window definitions from memory and removes the windows from the screen.

CREATE SCREEN – Displays the screen layout window and enables you to create a screen.

DEACTIVATE WINDOW – Deactivates specific user-defined windows or FoxPro system windows and removes them from the screen, but does not remove the window definitions from memory.

DEFINE WINDOW – Creates a window and specifies its attributes.

HIDE WINDOW – Hides an active user-defined window or FoxPro system window.

MODIFY SCREEN – Displays the screen layout window and enables you to modify or create a screen.

MODIFY WINDOW – In FoxPro for Windows, modifies a defined window.

MOVE WINDOW – Moves a window to a specified screen location.

RELEASE WINDOWS – Releases specified window definitions from memory and removes the windows from the screen.

RESTORE WINDOW – Restores all or specified window definitions to memory from a window file or a memo field.

SAVE WINDOW – Saves all or specified window definitions to a window file or a memo field.

SET BORDER – Defines a border for popups created with `DEFINE POPUP` and for windows created with `DEFINE WINDOW`.

SET SHADOWS – In FoxPro for MS-DOS, places or removes shadows from all user-defined windows, popups created with `DEFINE POPUP`, and FoxPro system windows and popups.

SHOW WINDOW – Displays one or more user-defined windows or FoxPro system windows without activating them.

SYS(2016) – Returns the window name specified with the most recent `SHOW GETS WINDOW`.

WBORDER() – Returns true (.T.) if a window has a border.

WCHILD() – Returns either the number of child windows or the names of child windows, for the current or specified parent window.

WCOLS() – Returns the number of columns in a specified window.

WEXIST() – Returns true (.T.) if the specified window exists.

WFONT() – Returns the name, size or style of the current font for a window in FoxPro for Windows.

WLAST() – Returns the name of the window that was active prior to the current window, or true (.T.) if the named window was active prior to the current window.

WLCOL() – Returns the column coordinate of the top left corner of the current or specified window.

WLROW() – Returns the row coordinate of the top left corner of the current or specified window.

WMAXIMUM() – Returns true (.T.) if the specified window is maximized.

WMINIMUM() – Returns true (.T.) if the specified window is minimized.

WONTOP() – Returns true (.T.) if the specified window is in front of all other active windows.

WOUTPUT() – Returns true (.T.) if output is currently being directed to the specified window.

WPARENT() – Returns the name of the parent window of the current or specified child window.

WREAD() – Returns true (.T.) if specified window is involved in the current READ.

WROWS() – Returns the number of rows in a specified window.

WTITLE() – Returns the title assigned to the current or specified window.

WVISIBLE() – Returns true (.T.) if the specified window is not hidden.

ZOOM WINDOW – Changes the size of a user-defined window or a FoxPro system window

Typing, Keyboard and Mouse

These commands and functions affect keyboard entry and the mouse.

CAPSLOCK() – Returns the current mode of the CapsLock key, or sets the CapsLock key mode on or off.

CHRSAW() – Checks the keyboard buffer for the presence of a character.

CLEAR MACROS – Releases all keyboard macros from memory including any Function key assignments.

CLEAR TYPEAHEAD – Clears the keyboard typeahead buffer.

INSMODE() – Returns the current mode of the Insert key, or sets the Insert key mode on or off.

KEYBOARD – Places the specified character expression in the keyboard buffer.

LASTKEY() – Returns a value corresponding to the last key pressed.

MCOL() – Returns the column position of the mouse pointer on the screen or in a window.

MDOWN() – Returns a logical value corresponding to the state of the left mouse button.

MROW() – Returns the screen or window row position of the mouse pointer.

NUMLOCK() – Returns the current mode of the NumLock key, or sets the mode of the NumLock key on or off.

ON KEY – Specifies a command that executes when you press any key during program execution.

ON KEY LABEL – Specifies a command that executes when you press a specific key.

PLAY MACRO – Executes a keyboard macro.

POP KEY – Restores ON KEY LABELs placed on the “stack” with PUSH KEY.

PUSH KEY – Places all current ON KEY LABELs on a “stack” in memory.

RESTORE MACROS – Restores a set of keyboard macros to memory from a disk file or a memo field.

SAVE MACROS – Saves a set of keyboard macros to a keyboard macro file or a memo field.

SET FUNCTION – Assigns a character string to a function key.

SET KEYCOMP – Controls FoxPro keystroke navigation on different platforms.

SET MACKEY – Specifies a key or key combination that displays the Macros dialog.

SET TYPEAHEAD – Specifies the maximum number of characters that can be stored in the typeahead buffer.

SYS(2008) – In FoxPro for MS-DOS, specifies the shape of the insert and overwrite cursors.

SYS(2009) – Swaps the insert and overwrite cursors.

Reports and Labels

You use these commands to control report creation and output.

CREATE LABEL – Creates labels and saves them to a label definition file.

CREATE REPORT – Opens the Report Layout window and enables you to create a report and save it to a report definition file.

LABEL – Prints labels from a table/.DBF and a label definition file.

MODIFY LABEL – Allows you to create or edit labels and save them to a label definition file.

MODIFY REPORT – Opens the Report Layout window and enables you to create or edit a report and save it to a report definition file.

REPORT – Prints a report from a table/.DBF and a report definition file.

FoxPro Environment

Environment

You use these commands and functions to manage a wide range of FoxPro system and environment parameters.

ADIR() – Places information about specified files into an array.

CAPSLOCK() – Returns the current mode of the CapsLock key, or sets the CapsLock key mode on or off.

COL() – Returns the current column position of the cursor.

CREATE COLOR SET – Creates a new color set from the current color settings.

DIR *or*

DIRECTORY – Displays the contents of a disk directory.

DISKSPACE() – Returns the number of bytes available on the default disk drive.

DISPLAY STATUS *or*

LIST STATUS – Displays the status of the FoxPro environment.

_DOS – Contains a logical true (.T.) if you are using FoxPro for MS-DOS.

FKLABEL() – Returns the name of the Function key corresponding to a numeric expression.

FKMAX() – Returns the number of programmable function keys or Function key combinations available on your keyboard.

FONTMETRIC() – Returns font attributes for installed Windows fonts in FoxPro for Windows.

GETDIR() – Displays the Select Directory dialog and returns the directory you choose.

GETENV() – Returns the contents of a specified MS-DOS environment variable.

GETFILE() – Displays the Open dialog and returns the name of the file you choose.

GETFONT() – Displays the FoxPro for Windows Font dialog and returns the name of the font you choose.

HELP – Invokes the online help facility.

INKEY() – Returns a number corresponding to the first key or mouse click in the typeahead buffer.

INSMODE() – Returns the current mode of the Insert key, or sets the Insert key mode on or off.

ISCOLOR() – Determines whether or not your computer's graphics card can display color.

LOCFILE() – Returns the path of the specified file name.

_MAC – Contains a logical true (.T.) if you are using FoxPro for Macintosh[®].

MEMORY() – Returns the amount of available memory.

MESSAGE() – Returns the current error message string or the contents of the program line that caused an error.

NUMLOCK() – Returns the current mode of the NumLock key, or sets the mode of the NumLock key on or off.

OS() – Returns the name of the operating system under which FoxPro is running.

PCOL() – Returns the current column position of the printer output.

PROW() – Returns the current row position of the printer output.

PUTFILE() – Invokes the Save As dialog and returns the name of the file you specify.

QUIT – Quits FoxPro and returns to the MS-DOS prompt in FoxPro for MS-DOS, or to Windows in FoxPro for Windows.

READKEY() – Returns a number corresponding to the key you pressed to exit one of certain editing commands.

ROW() – Returns the current row position of the cursor.

RUN | ! – Runs an external command or application from within FoxPro.

SCHEME() – Returns a color pair or a color pair listing from a color scheme.

SET – Opens the View window.

SET ANSI – Specifies how FoxPro compares strings of different lengths with the = operator in SQL commands.

SET BELL – Turns the computer bell on or off, and sets the bell attributes.

SET BLOCKSIZE – Specifies how FoxPro allocates disk space for the storage of memo fields.

SET CENTURY – Determines whether or not FoxPro displays the century portion of date expressions.

SET CLEAR – Determines whether or not SET FORMAT TO and QUIT clear the screen.

SET CLOCK – Determines whether or not FoxPro displays the system clock, and specifies the clock location on the screen.

SET COLOR OF SCHEME – Specifies the colors in a color scheme or copies one color scheme to another color scheme.

SET COLOR SET – Loads a previously defined color set.

SET COLOR TO – Specifies the colors of user-defined menus and windows.

SET COMPATIBLE – Controls compatibility with FoxBASE+.

SET CURRENCY – Defines the currency symbol and the position of the currency symbol in the display of numeric expressions using the “\$” PICTURE or FUNCTION code.

SET DATE – Sets the date format for the display of date expressions.

SET DEBUG – Enables or disables access to the Debug and Trace windows.

SET DEFAULT – Specifies the default drive and directory.

SET DISPLAY – Lets you change the current display mode on monitors that support different modes.

SET FIXED – Fixes the number of decimal places in the display of numeric data.

SET FUNCTION – Assigns an expression to a Function key.

SET HELP – Enables or disables the FoxPro online help facility, or specifies a help file.

SET HELPFILTER – Enables the Help window to display a subset of help topics.

SET KEYCOMP – Controls FoxPro keystroke navigation on different platforms.

SET LIBRARY – Opens an external library file.

SET MARK TO – Specifies a delimiter in the display of date expressions.

SET MOUSE – In FoxPro for MS-DOS, controls the sensitivity of the mouse.

SET NOTIFY – Enables or disables the display of certain system messages.

SET PATH – Specifies a file search path.

SET POINT – Determines the decimal point character in the display of numeric expressions.

SET RESOURCE – Specifies or updates a resource file.

SET SCOREBOARD – Specifies where the state of the NumLock, CapsLock and Insert keys is displayed on the screen in FoxPro for MS-DOS.

SET SEPARATOR – Specifies the character that separates each group of three digits to the left of the decimal point.

SET STATUS BAR – Displays a Windows-style status bar in FoxPro for Windows.

SET STICKY – Specifies how the mouse displays menus in the FoxPro menu system.

SET TALK – Enables or disables the display of command results to the screen or a window.

SET TEXTMERGE – Enables or disables evaluation of fields, memory variables, functions and expressions that are surrounded by text merge delimiters.

SET TEXTMERGE DELIMITERS – Specifies the text merge delimiters.

SET TOPIC – Specifies the initial help topic when you invoke the FoxPro help system.

SET TYPEAHEAD – Specifies the maximum number of characters that can be stored in the typeahead buffer.

SET VIEW – Opens or closes the View window or restores the FoxPro environment from a view file.

SET() – Returns the status of the specified SET command.

SYSMETRIC() – Returns the size of a Windows display element in FoxPro for Windows.

SYS(3) – Returns a unique, legal file name.

SYS(5) – Returns the current SET DEFAULT setting.

SYS(6) – Returns the current SET PRINTER TO setting.

- SYS(9)** – Returns your FoxPro serial number.
- SYS(12)** – Returns the amount of memory available for the RUN command in bytes.
- SYS(17)** – Returns the type of central processing unit you are using.
- SYS(24)** – Returns the EMS limit that was set through the CONFIG.FP configuration file in FoxPro for MS-DOS.
- SYS(100)** – Returns the current SET CONSOLE setting.
- SYS(101)** – Returns the current SET DEVICE setting.
- SYS(102)** – Returns the current SET PRINTER setting.
- SYS(103)** – Returns the current SET TALK setting.
- SYS(1001)** – Returns the amount of memory available to the FoxPro memory manager.
- SYS(1016)** – Returns the amount of memory being used by user-defined objects, open tables/.DBFs and low-level files, memory variables, and so on.
- SYS(2003)** – Returns the name of the current directory on the default drive.
- SYS(2004)** – Returns the name of the directory that contains the FoxPro for MS-DOS FOXPRO.EXE executable file or the FoxPro for Windows FOXPROW.EXE executable file.
- SYS(2005)** – Returns the name of the active resource file.
- SYS(2006)** – Returns as a character string the type of graphics card and monitor in use.
- SYS(2008)** – Specifies the shape of the insert and overwrite cursors.
- SYS(2009)** – In FoxPro for MS-DOS, swaps the shapes of the insert and overwrite cursors.
- SYS(2010)** – Returns as a character string the files setting in your MS-DOS CONFIG.SYS configuration file.
- SYS(2013)** – Returns the names of the FoxPro system menu bar, each pad on the system menu bar, system menu popups and each option on the system menu popups.
- SYS(2016)** – Returns the name of the window specified with the most recent SHOW GETS WINDOW.
- SYS(2017)** – Clears the screen and displays the FoxPro sign-on screen.

SYS(2019) – Returns the name and path of the FoxPro for MS-DOS configuration file CONFIG.FP, or of the FoxPro for Windows configuration file CONFIG.FPW.

SYS(2020) – Returns the total size (in bytes) of the default drive.

SYS(2022) – Returns the cluster size (in bytes) of a specified disk.

SYS(2023) – Returns the drive where the FoxPro temporary files are stored.

TEXTWIDTH() – Returns the length of a character expression with respect to a font in FoxPro for Windows.

_UNIX – Contains a logical true (.T.) if you are using FoxPro for UNIX®.

VERSION() – Returns the version of FoxPro you are using.

WFONT() – Returns the name of the current font for a specified window in FoxPro for Windows.

_WINDOWS – Contains a logical true (.T.) if you are using FoxPro for Windows.

File Management

You use the commands and functions in this section to manage and manipulate disk files.

ADIR() – Places information about specified files into an array.

CLOSE ALL – Closes all files and selects work area 1.

CLOSE ALTERNATE – Closes the file you opened with SET ALTERNATE.

CLOSE DATABASES – Closes all open tables/.DBFs.

CLOSE INDEX – Closes all open index files.

CLOSE PROCEDURE – Closes the file opened with SET PROCEDURE.

COPY FILE – Duplicates any type of file.

CURDIR() – Returns the current MS-DOS directory on a specified drive.

DBF() – Returns the path and name of the table/.DBF open in the specified work area.

DELETE FILE – Deletes a file from disk.

DIR | DIRECTORY – Displays the contents of a directory.

DISPLAY *or*

LIST FILES – Displays information for specified files.

DISKSPACE() – Returns the number of bytes currently available on the default disk drive.

ERASE – Deletes a file from disk.

FILE() – Returns true (.T.) if a specified file can be found on disk.

FILER – Invokes the FoxPro Filer disk and file maintenance utility.

FLUSH – Saves table/.DBF and index modifications to disk.

FULLPATH() – Returns the MS-DOS path for the given file.

GETFILE() – Displays the Open dialog and returns the name of the file you choose.

LOCFILE() – Returns the path of the specified file.

MODIFY COMMAND | FILE – Opens a text editing window and allows you to modify or create a program or text file.

PUTFILE() – Invokes the Save As dialog and returns the name of the file you specify.

RENAME – Renames a file.

SET ALTERNATE – Directs screen or window output to a text file.

SET DEFAULT – Specifies the default drive and directory.

SET FULLPATH – Specifies whether or not DBF() and NDX() return a file name that includes a path.

SET PATH – Specifies a file search path.

SET SAFETY – Determines whether or not FoxPro displays a warning before overwriting an existing file.

SET VIEW – Opens or closes the View window or restores the FoxPro environment from a view file.

SYS(3) – Returns a unique file name.

SYS(2000) – Returns the name of the first file that matches the specified skeleton.

SYS(2014) – Returns the minimum path relative to the current directory for a specified file.

TYPE – Displays the contents of an ASCII file.

FoxPro in a Multi-User Environment

Network Commands and Functions

FoxPro allows multiple users on a network to share tables/.DBFs. The following commands and functions are used in connection with a network.

FLOCK() – Attempts to lock a table/.DBF and returns true (.T.) if successful.

LOCK() – Attempts to lock one or more records in the current or specified table/.DBF, and returns true (.T.) if successful.

RLOCK() – Attempts to lock one or more records in the current or specified table/.DBF, and returns true (.T.) if successful.

SET EXCLUSIVE – Specifies if FoxPro opens tables/.DBFs for exclusive or shared use on a network.

SET LOCK – Enables or disables automatic file locking in certain commands.

SET MULTLOCKS – Determines whether or not you can lock multiple records with **LOCK()** or **RLOCK()**.

SET REFRESH – Determines whether or not your screen displays changes made to records by other users on the network while you are using **BROWSE**, **CHANGE**, **EDIT**, or **MODIFY MEMO**.

SET REPROCESS – Specifies the number of times or for how long FoxPro attempts to lock a file or record after an unsuccessful locking attempt.

UNLOCK – Releases a record lock, multiple records locks or a file lock from the current or specified table/.DBF, or releases all record and file locks from all open tables/.DBFs.

3 Commands and Functions Alphabetically

This chapter contains a complete listing of the FoxPro commands and functions. A condensed version of this information is also available in the FoxPro online help facility. We recommend that you explore the help facility. It's a convenient way of quickly finding an answer while you're using FoxPro.

Examples



All the examples in this chapter and the System Memory Variables chapter that follows appear in the FoxPro online help facility. It isn't necessary to type an example into a program to run it – an example from the help facility can be copied and pasted directly into a program. See the first chapter of this manual (Overview of the FoxPro Language) for a discussion on creating and executing programs, and how to copy and paste an example from the help facility into a program.

The files used in these examples are in the LANGUAGE directory, located off the directory where you installed FoxPro. FoxPro must be able to locate the files for the examples to run properly. SET DEFAULT is used to tell FoxPro where these examples are located.

For example, assume you install FoxPro for MS-DOS in the default FOXPRO25 directory on the C drive. The LANGUAGE directory is located off the FOXPRO25 directory. The command

```
SET DEFAULT TO C:\FOXPRO25\LANGUAGE
```

tells FoxPro for MS-DOS where to look for the example files.

Assume you install FoxPro for Windows in the default FOXPROW directory on the C drive. The LANGUAGE directory is located off the FOXPROW directory. The command

```
SET DEFAULT TO C:\FOXPROW\LANGUAGE
```

tells FoxPro for Windows to look in the LANGUAGE directory for the example files.

For more information how FoxPro locates files, see SET DEFAULT and SET PATH in this chapter.

The examples included in this chapter are intended to illustrate how FoxPro commands and functions work. The examples are usually short and do not include error trapping. Because there is no error trapping, they may not always execute as expected. For instance, if certain files or windows are already open or cannot be located, if there is a lack of disk space or memory and so on, an error message may be generated.

If an error occurs while running an example, a message is displayed on the screen. Make note of the error message. A complete listing of FoxPro error messages with suggestions to correct the error is available to FoxPro for Windows users in the FoxPro *Developer's Guide* and to FoxPro for MS-DOS users in the FoxPro *Update* manual.

FoxPro Version 2.5

FoxPro 2.5 features a multitude of new and enhanced commands, functions, preprocessor directives and system memory variables. This manual contains a complete description of each.

The following table provides a list of all commands, functions, preprocessor directives and system memory variables that are new or enhanced in FoxPro 2.5. The available platforms for each is listed along with descriptions of all FoxPro 2.5 enhancements.

Command, Function, Preprocessor Directive or System Memory Variable	Available Platforms	New	Enhanced	Enhancements
? ??	All		✓	FONT clause in FoxPro for Windows
@ ... SAY	All		✓	FONT, STYLE, BITMAP, CENTER, ISOMETRIC, STRETCH and VERB clauses, support for BMP pictures, OLE objects and RGB color pairs in FoxPro for Windows
@ ... GET	All		✓	FONT and STYLE clauses and support for RGB color pairs in FoxPro for Windows
@ ... GET — Check Boxes	All		✓	FONT and STYLE clauses, support for pictures, masks and RGB color pairs in FoxPro for Windows
@ ... GET — Invisible Buttons	All		✓	Support for RGB color pairs in FoxPro for Windows
@ ... GET — Lists	All		✓	FONT and STYLE clauses and support for RGB color pairs in FoxPro for Windows
@ ... GET — Popups	All		✓	FONT and STYLE clauses and support for RGB color pairs in FoxPro for Windows

Command, Function, Preprocessor Directive or System Memory Variable	Available Platforms	New	Enhanced	Enhancements
@ ... GET — Push Buttons	All		✓	FONT and STYLE clauses, support for pictures, masks and RGB color pairs in FoxPro for Windows
@ ... GET — Radio Buttons	All		✓	FONT and STYLE clauses, support for pictures, masks and RGB color pairs in FoxPro for Windows
@ ... GET — Spinners	Windows	✓		
@ ... GET — Text Editing Regions	All		✓	FONT and STYLE clauses and support for RGB color pairs in FoxPro for Windows
@ ... FILL	All		✓	Support for RGB color pairs in FoxPro for Windows
@ ... TO	All		✓	Rounded rectangles, ellipses, circles, fill patterns, outline size and support for RGB color pairs in FoxPro for Windows
AFONT()	Windows	✓		
ANSITOOEM()	Windows	✓		
APPEND FROM	All		✓	Support for Microsoft Excel 3.0 and 4.0, Paradox [®] 3.5 and 4.0
APPEND GENERAL	Windows	✓		
BROWSE	All		✓	FONT and STYLE clauses and support for RGB color pairs in FoxPro for Windows
CHANGE	All		✓	FONT and STYLE clauses and support for RGB color pairs in FoxPro for Windows

Command, Function, Preprocessor Directive or System Memory Variable	Available Platforms	New	Enhanced	Enhancements
DDEAbortTrans()	Windows	✓		
DDEAdvise()	Windows	✓		
DDEEnabled()	Windows	✓		
DDEExecute()	Windows	✓		
DDEInitiate()	Windows	✓		
DDELastError()	Windows	✓		
DDEPoke()	Windows	✓		
DDERequest()	Windows	✓		
DDESetOption()	Windows	✓		
DDESetService()	Windows	✓		
DDESetTopic()	Windows	✓		
DDETerminate()	Windows	✓		
#DEFINE ... #UNDEF	All	✓		
DEFINE BAR	All		✓	Support for RGB color pairs in FoxPro for Windows
DEFINE MENU	All		✓	Support for RGB color pairs in FoxPro for Windows
DEFINE PAD	All		✓	Support for RGB color pairs in FoxPro for Windows
DEFINE POPUP	All		✓	Support for RGB color pairs in FoxPro for Windows
DEFINE WINDOW	All		✓	Support for RGB color pairs, FONT, STYLE, AT, SIZE, FILL FILE, ICON FILE, IN DESKTOP, MDI, NOMDI and HALFHEIGHT clauses in FoxPro for Windows
DIR DIRECTORY	All		✓	PROMPT clause in FoxPro for Windows
DISPLAY/LIST	All		✓	PROMPT clause in FoxPro for Windows
_DOS	All	✓		
FONTMETRIC()	Windows	✓		
GETDIR()	All	✓		
GETFONT()	Windows	✓		

Command, Function, Preprocessor Directive or System Memory Variable	Available Platforms	New	Enhanced	Enhancements
#IF ... #ENDIF	All	✓		
IMPORT	All		✓	Support for Microsoft Excel 3.0 and 4.0, Paradox 3.5 and 4.0
LABEL	All		✓	PROMPT clause in FoxPro for Windows
_MAC	All	✓		
MODIFY COMMAND/FILE	All		✓	SAME clause in FoxPro for Windows
MODIFY GENERAL	Windows	✓		
MODIFY MEMO	All		✓	SAME clause in FoxPro for Windows
MODIFY WINDOW	Windows	✓		
MOVE WINDOW	All		✓	CENTER clause
OEMTOANSI()	Windows	✓		
READ	All		✓	Support for RGB color pairs in FoxPro for Windows
REPORT	All		✓	PROMPT clause in FoxPro for Windows
SELECT - SQL	All		✓	Query results sent to a Browse window, NOWAIT, PREFERENCE and TO SCREEN clauses, PROMPT clause in FoxPro for Windows
SET CLOCK	All		✓	STATUS clause in FoxPro for Windows
SET DEVELOPMENT	All		✓	Cancel program execution during a READ
SET DEVICE	All		✓	PROMPT clause in FoxPro for Windows
SET HELP	All		✓	Windows style help in FoxPro for Windows
SET KEYCOMP	Windows	✓		
SET PALETTE	Windows	✓		
SET PRINT	All		✓	PROMPT clause in FoxPro for Windows
SET STATUS	All		✓	TIMEOUT clause in FoxPro for Windows
SET STATUS BAR	Windows	✓		

Command, Function, Preprocessor Directive or System Memory Variable	Available Platforms	New	Enhanced	Enhancements
SHOW GET	All		✓	Support for RGB color pairs in FoxPro for Windows
SHOW GETS	All		✓	Support for RGB color pairs in FoxPro for Windows
SHOW OBJECT	All		✓	Support for RGB color pairs in FoxPro for Windows
SHOW WINDOW	All		✓	REFRESH clause updates memo editing windows
SYS(20)	All	✓		
SYSMETRIC()	Windows	✓		
_TRANSPORT	All	✓		
TYPE	All		✓	PROMPT clause in FoxPro for Windows
TXTWIDTH()	Windows	✓		
_UNIX	All	✓		
USE	All		✓	SHARE clause, support for 0 as a work area
WFONT()	Windows	✓		
_WINDOWS	All	✓		

%

Purpose	Modulus operator - Returns the remainder obtained by dividing a numeric expression by another numeric expression
Syntax	<expN1> % <expN2>
Parameters	<expN1> The dividend (the number being divided) <expN2> The divisor (the number the dividend is divided by)
See Also	MOD()

Description The modulus operator % returns the remainder obtained when dividing a numeric expression by another numeric expression. The modulus operator and MOD() return identical results.

The modulus operator % is an arithmetic operator like + (addition), - (subtraction), * (multiplication), / (division) and ^ (exponentiation). When these operators are combined in a numeric expression, % has the same precedence as * and /. See the Overview of the FoxPro Language chapter in this manual for a further discussion of operators and their order of precedence.

Parameters **<expN1>**
<expN1> is the dividend (the numeric expression being divided). The number of decimal places in **<expN1>** determines the number of decimals places in the result.

<expN2>
<expN2> is the divisor (the numeric expression dividing the dividend **<expN1>**). A positive number is returned if **<expN2>** is positive and a negative number is returned if **<expN2>** is negative. **<expN2>** cannot be zero.

Examples

```
? 36 % 10
6
? (4*9) % (90/9)
6
? 25.250 % 5.0
0.250
? IIF(YEAR( DATE( ) ) % 4 = 0, 'Summer Olympics this year';
, 'No Summer Olympics this year')
```

&

Purpose	Macro substitution
Syntax	& <memvar> [. <expC>]

Description Macro substitution treats the contents of a memory variable or array element as a character string literal. When an ampersand & precedes a character type memory variable or array element, the contents of the variable or element replace the macro reference. You can use macro substitution in any command or function that accepts a character string literal.



Whenever possible, use a name expression instead of macro substitution. A name expression operates like macro substitution. However, a name expression is limited to passing character strings as names. Use a name expression for significantly faster processing if a command or function accepts a name (a file name, window name, popup name ...). See the Overview of the FoxPro Language chapter in this manual for additional information on name expressions.

While the following commands are acceptable:

```
STORE 'Customer' TO dbf_file  
STORE 'company' TO tagname  
USE &dbf_file ORDER &tagname
```

use a name expression instead:

```
USE (dbf_file) ORDER (tagname)
```

Macro substitution is useful for substituting a key word in a command. In the following example, the TALK setting is saved to a memory variable so the setting can be restored later in the program. The original TALK setting is restored with macro substitution.

```
STORE SET('TALK') TO savetalk
SET TALK OFF
*
* Additional program code
*
SET TALK TO &savetalk&& Restore original TALK setting
```

Parameters <memvar>

This is the name of the memory variable or array element to reference in the macro substitution. Do not include the M. prefix that distinguishes memory variables from fields. This causes a syntax error. The macro should not exceed the maximum statement length permitted in FoxPro.

A variable cannot reference itself recursively in macro substitution. For example, the following generates an error message:

```
STORE '&X' TO X
? &X
```

Macro substitution statements that appear in DO WHILE, FOR and SCAN are only evaluated at the start of the loop and are not reevaluated on subsequent iterations. Any changes to the memory variable or array element that occur within the loop are not recognized.

.<expC>

The optional . (period) delimiter and character expression <expC> is used to append additional characters to a macro. The character expression <expC> appended to the macro with .<expC> can also be a macro.

\$

Purpose	Returns true if a character expression is contained in another character expression
Syntax	<expC1> \$ <expC2>
Parameters	<expC1> Search string <expC2> String to be searched
Returns	Logical
See Also	AT()

Description This function returns true (.T.) if a character expression is contained in another character expression. If the character expression isn't found, false (.F.) is returned. This function isn't Rushmore optimizable.

Parameters **<expC1>, <expC2>**

The character expression <expC2> is searched for an occurrence of <expC1>. If <expC1> is found in <expC2>, .T. is returned. If <expC1> isn't found in <expC2>, .F. is returned. This function is case sensitive (upper- and lower-case are respected).

<expC1> and <expC2> can be character type memory variables or array elements, character type fields, character string literals or memo fields of any length.

Memo fields can be manipulated like character expressions, fields in tables/DBFs, memory variables or array elements. For example, if MEMO_FLD is a memo field, the following is perfectly acceptable:

```
LIST FOR 'FOX' $ UPPER(memo_fld)
```

Example

The following example creates a table/.DBF called MEMOTEST containing one memo field. Three records are appended to the table/.DBF and a Browse window is opened to display the 3 records. \$ is used to list the records which contain the string "FOX". The files created for the example are then deleted.

```
CLOSE DATABASES
CLEAR
CREATE CURSOR memotest (text C(3), memo M)
INSERT INTO memotest (text, memo) VALUES ('Fox', 'Fox')
INSERT INTO memotest (text, memo) VALUES ('Cat', 'Cat')
INSERT INTO memotest (text, memo) VALUES ('FOX', 'FOX')

LIST FIELDS memo, text FOR 'FOX' $ UPPER(memo)
USE
```


=

Purpose	Evaluates one or more expressions
Syntax	<code>= <expr1> [, <expr2> ...]</code>
See Also	?, User-Defined Functions

Description The = command evaluates one or more expressions <expr1>, <expr2> ... and discards the return values. This is particularly useful when a FoxPro function or a user-defined function has a desired effect but there is no need to assign the function's return value to a memory variable, array element or field.

For example, to turn insert mode on you can issue the command:

```
= INSMODE(.T.)
```

INSMODE normally returns a true (.T.) or false (.F.) value. In this example the function is executed but the return value is discarded.



There are two other unrelated uses for the equal sign =. It can be used as an operator in logical expressions to make a comparison or to assign values to memory variables and array elements. In these two cases the equal sign = is an operator and not a command.

Purpose	Outputs lines of text
Syntax	<code>\ <text line> \ \ <text line></code>
See Also	<code>_PRETEXT</code> , <code>SET TEXTMERGE</code> , <code>SET TEXTMERGE DELIMITERS</code> , <code>_TEXT</code> , <code>TEXT ... ENDTEXT</code>

Description The `\` and `\ \` commands facilitate text merge in FoxPro. Text merge lets you output text to a file to create form letters or programs.

Use `\` and `\ \` to output a text line to the current text merge output file and the screen. `SET TEXTMERGE` is used to specify the text merge output file. If text merge isn't directed to a file, the text line is output only to the desktop (FoxPro for MS-DOS) or the main FoxPro window (FoxPro for Windows). `SET TEXTMERGE NOSHOW` suppresses output to the desktop or main FoxPro window.

When you use `\`, the output line is preceded by a carriage return and a linefeed. When you use `\ \`, the output line isn't preceded by a carriage return and linefeed.

Any spaces preceding `\` and `\ \` are not included in the output line, but spaces following `\` and `\ \` are.

You can embed an expression in the text line. If the expression is enclosed in the text merge delimiters (`<<` `>>` by default) and `TEXTMERGE` is `SET ON`, the expression is evaluated and its value is output as text.

Example

USE customer

SET TEXTMERGE ON

SET TEXTMERGE TO letter.txt

\<<CDOW(DATE())>>, <<CMONTH(DATE())>>

\\<<DAY(DATE())>>, <<YEAR(DATE())>>

\

\

\Dear <<contact>>

\

\Additional Text

\

\Thank you,

\

\XYZ Corporation

CLOSE ALL

MODIFY FILE letter.txt NOEDIT

? | ??

Purpose	Evaluates expressions and outputs the results
Syntax	? ?? <expr1> [PICTURE <expC1>] [FUNCTION <expC2>] [AT <expN1>] [FONT <expC3> [, <expN2>] [STYLE <expC4> <expr2>] [, <expr3>] ...
Remarks	FONT clause — FoxPro for Windows only
See Also	???, @ ... SAY, SET MEMOWIDTH, SET PRINTER

Description ? and ?? evaluate expressions and send the results to the desktop (FoxPro for MS-DOS), the main FoxPro window (FoxPro for Windows), an active user-defined window or the printer.

If SET PRINTER is ON, the expression results are directed to the printer and the desktop, the main FoxPro window or an active user-defined window. If SET PRINTER is ON and SET CONSOLE is OFF, the results are only directed to the printer.

Clauses ?

A single question mark sends a carriage return and line feed before the expression results. The results are displayed on the next line of the desktop, main FoxPro window or the active user-defined window and printed at the left margin of a page unless a function code <expC2> or the _ALIGNMENT system memory variable specifies otherwise.

If the expressions are omitted, a blank line is printed. A space is placed between the expression results when multiple expressions are included.

??

Two question marks display the expression results on the current line at the current position of the desktop, main FoxPro window or an active user-defined window or printer — a carriage return and line feed isn't sent before the results.

PICTURE <expC1>

If the PICTURE clause is present, the result of <expr1> is displayed according to the format specified by <expC1>. <expC1> can consist of function codes, picture codes or a combination of both.

You can use the same picture and function codes available in @ ... SAY. Refer to @ ... SAY for a list of picture and function codes.

Function codes affect the overall format of the result; picture codes act on individual characters in the result. If function codes are used in <expC1>, they must appear before the picture codes and they must be preceded by @. Multiple function codes with no embedded spaces can immediately follow @. The last function code must be followed by one or more spaces. The space(s) signals the end of the function codes and the start of the picture codes.

FUNCTION <expC2>

The FUNCTION character expression <expC2> offers another method of including function codes in ? and ?? output. If the function clause is included, the function codes are not preceded by @. Function codes must be preceded by @ when included in PICTURE.

V<n>

V<n> is a special function code that can be used with ? and ??. This code enables the results of a character expression to stretch vertically in a limited number of columns. <n> is the number of columns in the output.

```
? 'This is an example of how the V(n) function works.' ;
  FUNCTION 'V10'
```

AT <expN1>

AT is used to specify the column number <expN1> where the output is displayed. This option lets you align output in columns to create a table/DBF. The numeric expression <expN1> can be a UDF that returns a numeric value.

FONT <expC3> [, <expN2>] — FoxPro for Windows only

The character expression <expC3> is the name of the font, and the numeric expression <expN2> is the font size. For example, the following command displays the system date in 16 point Roman font:

```
? DATE( ) FONT 'ROMAN',16
```

If you include the FONT clause but omit the font size <expN2>, a 10 point font is used.

If the FONT clause is omitted and ? | ?? output is placed in the main FoxPro window, the main FoxPro window font is used for the output. If the FONT clause is omitted and ? | ?? output is placed in a user-defined window, the user-defined window font is used for the output.

If the font you specify is not available, Windows substitutes a font with similar font characteristics.

STYLE <expC4> | <expr2>

In FoxPro for Windows, include the STYLE clause to specify a font style for ? | ?? output. The styles that are available for a font are determined by Windows. If the font style you specify is not available, Windows substitutes a font style with similar characteristics.

The font style is specified with <expC4>. If the STYLE clause is omitted, the standard font style is used.

Character	Font Style
B	Bold
I	Italic
N	Normal
O	Outline
Q	Opaque
S	Shadow
—	Strikeout
T	Transparent
U	Underline

You can include more than one character to specify a combination of font styles. For example, the following command displays the system date in Bold Italic:

```
? DATE( ) STYLE 'BI'
```

In FoxPro for MS-DOS, the STYLE clause lets you specify the font style and size of output sent to the printer with ? and ??. The expression <expr2> included in a STYLE clause is sent to three procedures, PDOBJST, PDOBJECT and PDOBJEND, in the current FoxPro printer driver program. These procedures determine how output from ? and ?? are printed. The STYLE clause is ignored when output from ? and ?? is sent to the desktop or a user-defined window.

Example

```
? 15*(10+10)
300
```

```
? 'Welcome to ' PICTURE '@!'
?? 'FoxPro'
WELCOME TO FoxPro
```

???

???

Purpose	Sends output directly to the printer
Syntax	??? <expC>
See Also	? ??, @ ... SAY, @ ... GET, CHR()

Description Three question marks direct the contents of <expC> directly to the printer without incrementing the printer column or row. ??? allows you to send printer control codes directly to a printer without advancing the print head.

Printer control codes allow you to reset the printer, change type styles and sizes, enable or disable boldface printing, etc. These codes can consist of any combination of printable or non-printable characters that are specific to the printer you're using. You can direct control codes to the printer in several different ways:

- Use CHR() to send ASCII characters directly to the printer.
- Use braces { } to send printer codes or ASCII characters.
- Use a combination of CHR() and braces.
- Codes can be sent to the printer before printing begins and after printing ends with the _PSCODE and _PECODE system memory variables. See the System Memory Variables chapter in this manual for more information.

Printer control codes vary from printer to printer. The best source for information on printer control codes is the manual which accompanied your printer.

@ ... SAY

Purpose	Displays output at a specified row and column position
Syntax	<pre> @ <row, column> SAY <expr> [FUNCTION <expC1>] [PICTURE <expC2>] [SIZE <expN1>, <expN2>] [FONT <expC3> [, <expN3>]] [STYLE <expC4>] [COLOR SCHEME <expN4>] COLOR <color pair list>] </pre>
Remarks	FONT and STYLE clauses, support for RGB color pairs — FoxPro for Windows only
See Also	@ ... GET, CREATE SCREEN, SET COLOR, SET CURRENCY, SET DEVICE, SHOW GETS, User-Defined Functions

Description Use this command to display formatted output on the desktop in FoxPro for MS-DOS, the main FoxPro window in FoxPro for Windows or a user-defined window. It can also be used to format output for a printer.

If you use the Screen Builder to create your data entry screens, you may not have to use @ ... SAY at all. The Screen Builder automatically generates the @ ... SAYs.

You can combine @ ... SAY and @ ... GET into a single command. If both the SAY and GET clauses are included, specify a single set of coordinates <row, column> where @ ... SAY output begins. A space is automatically inserted between the @ ... SAY output and the @ ... GET text editing region.

If you issue SET DEVICE TO SCREEN, output from @ ... SAY appears on the desktop, the main FoxPro window or an active user-defined window. SET DEVICE TO SCREEN is the startup default. If you issue SET DEVICE TO PRINTER, output is directed to the printer.

Clauses **<row, column>**

Row and column are numeric expressions with values 0 or greater that determine where @ ... SAY output appears.

The first row is number 0 on the desktop, main FoxPro window or a user-defined window. In FoxPro for MS-DOS, row 0 is the row the FoxPro system menu bar occupies. See SET SYSMENU in this chapter for information about manipulating the system menu bar so you can place output on row 0. In FoxPro for Windows, row 0 is the row immediately under the FoxPro system menu bar.

Rows are numbered from top to bottom. For printers, the first row is 1 and the physical page size and the number of lines per page determines the maximum row number.

The first column is number 0 on the desktop, main FoxPro window or a user-defined window. Columns are numbered from left to right. For printers, the first column is 1 and the physical page size determines the maximum column number.

When @ ... SAY output is directed to a user-defined window, the row and column coordinates are relative to the user-defined window, not the desktop in FoxPro for MS-DOS or the main FoxPro window in FoxPro for Windows.

In FoxPro for Windows, a position in the main FoxPro window or in a user-defined window is determined by the font of the main FoxPro window or the user-defined window. Most fonts Microsoft Windows uses can be displayed in a wide variety of sizes, and some are proportionally spaced. A row corresponds to the height of the current font; a column corresponds to the average width of a letter in the current font. See the Font topic in the FoxPro online help for additional information about fonts.

To facilitate precise positioning of output in the main FoxPro window or in a user-defined window, you can use decimal fractions for row and column coordinates in FoxPro for Windows.

<expr>

<expr> is evaluated and displayed or printed starting at <row, column>. It can be a user-defined function (UDF).

FUNCTION <expC1> | PICTURE <expC2>

You can include the FUNCTION clause, the PICTURE clause or both to control how <expr> is displayed or printed.

FUNCTION codes can be included in a PICTURE clause. In this case, the PICTURE clause must start with @. Also, a PICTURE clause can contain FUNCTION codes, PICTURE codes or both. Since a FUNCTION clause affects the entire expression it can contain only FUNCTION codes.

FUNCTION Code	Purpose
B	Left-justifies numeric data within the display region.
C	CR is displayed after a positive number to indicate a credit. Can be used with numeric data only.
D	Uses the current SET DATE format.
E	Edits date type data as a BRITISH date.
T	Trim leading and trailing blanks from <expr>.
X	DB is displayed after negative numbers to indicate a debit. Use only with numeric data.
Z	<expr> is displayed as all blanks if its numeric value is 0. Use only with numeric data.
(	Encloses negative numbers in parentheses. Use only with numeric data.
!	Converts alphabetic characters to upper-case. Use with character data only.
^	Displays numeric data using scientific notation. Use with numeric data only.
\$	Displays data in currency format. The currency symbol appears before or after the field value depending on the current setting of SET CURRENCY. Use with numeric data only.

A PICTURE expression can include any characters, but only the characters listed below actively participate in display and printing.

PICTURE Code	Purpose
X	Allows any character.
Y	Allows logical Y, y, N and n only. Converts y and n to Y and N, respectively.
!	Converts lower-case letters to upper-case letters.
\$	Displays the current currency symbol specified by SET CURRENCY. By default, the symbol is placed immediately before or after the field. However, the currency symbol and its placement (SET CURRENCY), the separator character (SET SEPARATOR) and the decimal character (SET POINT) can all be changed.
*	Asterisks are displayed in front of the numeric value. Use with a dollar sign \$ for check protection.
.	Specifies the decimal point position.
,	Use to separate digits to the left of the decimal point.

Example

The following example combines two FUNCTION codes to format a numeric value. The \$\$ codes create a floating dollar sign and the C code places CR after the number.

```
CLEAR
@ 2, 2 SAY 1.15 FUNCTION 'C$$'
$1.15 CR
```

SIZE <expN1>, <expN2>

The SIZE clause lets you control the length and height of a @ ... SAY display or print region.

FoxPro creates a region one row high by default. Include the optional SIZE clause to create a region that is more than one row high. The height of the region in rows is specified by <expN1>, and the width in columns by <expN2>. The @ ... SAY output word wraps at the width specified by <expN1> for <expN2> rows.

In FoxPro for Windows, the @ ... SAY font determines the size of the editing region. The @ ... SAY font is specified with the FONT clause. If the FONT clause is omitted, the @ ... SAY output uses the font of its parent window (the main FoxPro window or a user-defined window).

FONT <expC3> [, <expN3>] — FoxPro for Windows only

The character expression <expC3> is the name of the font, and the numeric expression <expN3> is the font size. For example, the following command displays the @ ... SAY text in 16 point Roman font:

```
@ 2, 2 SAY 'Font clause example' FONT 'ROMAN', 16
```

If you include the FONT clause but omit the font size <expN3>, a 10 point font is used.

If the font you specify is not available, Windows substitutes a font with similar font characteristics.

STYLE <expC4> — FoxPro for Windows only

In FoxPro for Windows, include the STYLE clause to specify a font style for @ ... SAY output. The styles that are available for a font are determined by Windows. If the font style you specify is not available, Windows substitutes a font style with similar characteristics.

The font style is specified with <expC4>. If the STYLE clause is omitted, the standard font style is used.

Character	Font Style
B	Bold
I	Italic
N	Normal
O	Outline
Q	Opaque
S	Shadow
—	Strikeout
T	Transparent
U	Underline

You can include more than one character to specify a combination of font styles. For example, the following command specifies Bold Italic:

```
@ 2, 2 SAY 'Font clause example' STYLE 'BI'
```

COLOR SCHEME <expN4>

| COLOR <color pair list>

If you do not include a COLOR clause, the color of @ ... SAY output is determined by the color scheme for the desktop or the main FoxPro window; if @ ... SAY output is directed to a user-defined window, the window's color scheme determines the color of @ ... SAY output.

Only the first color pair in a color scheme or color pair list affects the color of @ ... SAY output.

The color of @ ... SAY output can be specified by including the number of an existing color scheme in the COLOR SCHEME clause or a set of color pairs in the COLOR clause.

A color scheme is a set of 10 predefined color pairs. The color pairs in a color scheme can be changed with SET COLOR OF SCHEME. In FoxPro for MS-DOS the color pairs in a color scheme can also be changed in the Color Picker.

A color pair is a set of two letters separated by a forward slash. The first color letter specifies the foreground color and the second letter specifies the background color.

For example, this color pair specifies a red foreground on a white background:

R/W

For a list of colors and their corresponding color letters, see the SET COLOR commands section in this chapter or the Color Table by Color Pair topic in the online help facility.

In FoxPro for Windows, a color pair can also be specified with a set of 6 RGB (Red Green Blue) color values separated by commas. The first 3 color values specify the foreground color and the second 3 color values specify the background color. The color values can range from 0 through 255.

In FoxPro for Windows, the R/W color pair in the example above can also be specified with this RGB color pair:

RGB(255,0,0,255,255,255)

Example

```
@ 2, 2 SAY 'This is red on white' COLOR R/W
@ 4, 2 SAY 'This is Color Scheme 16' COLOR SCHEME 16
IF _WINDOWS    && FoxPro for Windows.
    @ 6, 2 SAY 'This is red on white' COLOR RGB(255,0,0,255,255,225)
ENDIF
```

@ ... SAY — BMPs and OLE Objects

Purpose	Displays BMPs and OLE objects or calls the OLE server to execute the OLE object verb
Syntax	<pre>@ <row, column> SAY <file> BITMAP <general field> [CENTER] [ISOMETRIC STRETCH] [SIZE <expN1>, <expN2>]</pre> or <pre>@ <row, column> SAY <general field> VERB <expN3> <expC1></pre>
Remarks	FoxPro for Windows only
See Also	APPEND GENERAL, CREATE SCREEN, MODIFY GENERAL, SET DEVICE, SET PALETTE, SHOW GETS

Description Use this command to display .BMPs (Bitmap pictures) and OLE (Object Linking and Embedding) objects in the main FoxPro window or a user-defined window. .BMPs and OLE objects can also be directed to the printer with @ ... SAY.

If you issue SET DEVICE TO SCREEN, output appears in the main FoxPro window or an active user-defined window. SET DEVICE TO SCREEN is the startup default. If you issue SET DEVICE TO PRINTER, output is directed to the printer.

The second syntax form for this command is used to call the OLE server to service a verb for an OLE object. An OLE verb is a powerful tool for manipulating OLE objects. Use the VERB clause to execute commands associated with the OLE object. For example, you can start the application that created the OLE object, or edit or print an OLE object.



For additional information about OLE objects, see the Object Linking and Embedding chapter in the FoxPro for Windows *Developer's Guide*.

Clauses **<row, column>**

Row and column are numeric expressions with values 0 or greater that determine where the .BMP or OLE object appears. The first row is number 0 in the main FoxPro window or a user-defined window. Rows are numbered from top to bottom. The first column is number 0 in the main FoxPro window or a user-defined window. Columns are numbered from left to right. For printers, the first row and column is 1. The physical page size and the number of lines per page determines the maximum row number.

A position in the main FoxPro window or in a user-defined window is determined by the font of the main FoxPro window or the user-defined window. Most fonts Microsoft Windows uses can be displayed in a wide variety of sizes, and some are proportionally spaced. A row corresponds to the height of the current font; a column corresponds to the average width of a letter in the current font.

To facilitate precise positioning of a .BMP or an OLE object in the main FoxPro window or in a user-defined window, you can use decimal fractions for row and column coordinates.

When a .BMP or OLE object is displayed in a user-defined window, the row and column coordinates are relative to the user-defined window, not the main FoxPro window.

<file> BITMAP | <general field>

The .BMP or OLE object specified with <file> or <general field> is displayed starting at the position specified by the row and column coordinates. .BMPs are stored in files on disk. OLE objects are stored in fields of the general data type.

If you use the Screen Builder to create your data entry screens, you may not have to use this form of @ ... SAY at all. The Screen Builder automatically generates the @ ... SAYs.

To display a .BMP (bitmap picture) in the main FoxPro window or in a user-defined window, specify the name of the file containing the picture in <file> and include BITMAP. The BITMAP key word must follow the .BMP file name. The .BMP file name must be delimited with single or double quotation marks or square brackets. If the file name is stored in a memory variable or an array element, you can include the memory variable or array element without the delimiters.

If the .BMP file resides in a directory other than the current default directory, include a path name with the file name.

Example

```
CLEAR  
@ 2, 2 SAY '\windows\arcade.bmp' BITMAP
```


CENTER

Include the **CENTER** key word to center a .BMP or OLE object within the area specified by the **SIZE** clause. If the **SIZE** clause is omitted, the .BMP or OLE object is centered in the rectangular region bounded by the row and column coordinates and the lower right corner of the main FoxPro window or user-defined window.

ISOMETRIC | STRETCH

If the area specified with the **SIZE** clause is too small to contain a .BMP or OLE object, the picture is clipped to fit in the area. You can prevent a picture or OLE object from being clipped by including **ISOMETRIC** or **STRETCH**.

If **ISOMETRIC** is included, the picture or object is scaled so that its proportions are maintained. The picture is shrunk or expanded to fit within the specified area and isn't distorted.

If **STRETCH** is included, the picture or object is scaled horizontally, vertically or in both directions to fit the picture or object within the area specified by the **SIZE** clause. The picture or object's proportions are not maintained.

SIZE <expN1>, <expN2>

You can specify the size of a .BMP or OLE object with the **SIZE** clause. If the **SIZE** clause is omitted, a .BMP or an OLE object is displayed at its default size.

The numeric expression <expN1> specifies the height of the .BMP or OLE object. The height of the .BMP or OLE object is a multiple of the height of the font in the active window. For example, if <expN1> is 2, the height of the .BMP or OLE object is twice the height of the font in the active window.

The numeric expression <expN2> specifies the width of the .BMP or OLE object. The width of the .BMP or OLE object is a multiple of the width of the font in the active window. For example, if <expN2> is 2, the width of the .BMP or OLE object is twice the width of the font in the active window.

If the area specified with the **SIZE** clause is too small to contain a .BMP or OLE object, the picture or object is clipped. Include **ISOMETRIC** or **STRETCH** to prevent a .BMP from being clipped.

VERB <expN3> | <expC1>

Include the VERB clause to execute a command verb in the registration database for an OLE object.



Run REGEDIT /V to see what command verbs are available for each installed OLE server. The command verbs and their corresponding numbers are displayed.

The command verb can be either a numeric expression <expN1> or a character expression <expC1>. Command verb numbers start at 0.

For example, if the OLE object is a sound and the third command verb (<expN1> = 2) in the registration database is PLAY, issuing either

```
@ 2, 2 SAY sounds VERB 2
```

or

```
@ 2, 2 SAY sounds VERB 'PLAY'
```

plays the sound contained in the general field named SOUNDS. The OLE object contained in SOUNDS isn't displayed.

The verb EDIT is supported for most OLE objects.

For additional information about OLE objects, see the Object Linking and Embedding chapter in the *FoxPro for Windows Developer's Guide*.

@ ... GET

Purpose	Creates an editing region
Syntax	<pre>@ <row, column> GET <memvar> <field> [FUNCTION <expC1>] [PICTURE <expC2>] [FONT <expC3> [, <expN1>]] [STYLE <expC4>] [DEFAULT <expr1>] [ENABLE DISABLE] [MESSAGE <expC5>] [[OPEN] WINDOW <window name>] [RANGE [<expr2>] [, <expr3>]] [SIZE <expN2>, <expN3>] [VALID <expl1> <expN4> [ERROR <expC6>]] [WHEN <expl2>] [COLOR SCHEME <expN5> COLOR <color pair list>]</pre>
Remarks	FONT and STYLE clauses and support for RGB color pair — FoxPro for Windows only OPEN WINDOW included for backward compatibility — use @ ... EDIT instead
See Also	@ ... SAY, @ ... EDIT, CREATE SCREEN, _CUROBJ, FONTMETRIC(), OBJNUM(), READ, SET COLOR, SET CURRENCY, SET DEVICE, SHOW GETS, SHOW OBJECT, VARREAD(), User-Defined Functions

Description Use this command to create an editing region for the contents of a memory variable, array element or field. Use READ or READ CYCLE to activate @ ... GET editing regions.

You can combine @ ... SAY and @ ... GET into a single command. If both the SAY and GET clauses are included, specify a single set of coordinates <row, column> where the @ ... SAY output begins. A space is automatically inserted between the @ ... SAY output and the @ ... GET editing region. In FoxPro for Windows, @ ... GET editing regions are surrounded by a black border if the BORDER clause is included in READ.

If you use the Screen Builder to create your data entry screens, you may not have to use @ ... GET or @ ... SAY at all. The Screen Builder automatically generates the @ ... GETs or @ ... SAYs.

<row, column>

Row and column are numeric expressions with values 0 or greater that determine where the @ ... GET editing region is displayed. The first row is number 0 on the desktop, main FoxPro window or a user-defined window. Rows are numbered from top to bottom.

The first column is number 0 on the desktop, main FoxPro window or a user-defined window. Columns are numbered from left to right.

When the editing region is placed in a user-defined window, the row and column coordinates are relative to the user-defined window, not the desktop (FoxPro for MS-DOS) or the main FoxPro window (FoxPro for Windows).

In FoxPro for Windows, a position in the main FoxPro window or in a user-defined window is determined by the font of the main FoxPro window or the user-defined window. Most fonts Microsoft Windows uses can be displayed in a wide variety of sizes, and some are proportionally spaced. A row corresponds to the height of the current font; a column corresponds to the average width of a letter in the current font.

To facilitate precise positioning of an @ ... GET editing region in the main FoxPro window or in a user-defined window, you can use decimal fractions for row and column coordinates in FoxPro for Windows.

<memvar> | <field>

@ ... GET creates an editing region for the memory variable or array element specified in <memvar> or the field specified in <field>.

@ ... GET can be used to create an editing region for a memo field. When you use @ ... GET with a memo field, the word Memo is displayed. When READ is issued and the memo field is selected, place the cursor on the word Memo and press Ctrl+Home, Ctrl+PgUp or Ctrl+PgDn to open the memo editing window. You can also double-click on Memo to open the editing window.

To exit the memo editing window and save your editing changes in FoxPro for MS-DOS, click the window's close box or press Ctrl+W. Press Escape to discard your changes.

To exit the memo editing window and save your changes in FoxPro for Windows, choose **Close** from the window's **Control** menu or press Ctrl+W. Press Escape to discard your changes.

A better method for editing memo fields is using @ ... EDIT instead of @ ... GET. @ ... EDIT creates a text editing window with a scroll bar, and the contents of the memo field are displayed when @ ... EDIT is issued.



FUNCTION <expC1> | PICTURE <expC2>

When creating a text editing region with @ ... GET, you can include the FUNCTION clause, the PICTURE clause or both to create an editing mask. These clauses contain special codes that control how the memory variable, array element or field is displayed and edited.

FUNCTION codes can be included in a PICTURE clause. In this case, the PICTURE clause must start with @. Also, a PICTURE clause can contain FUNCTION codes, PICTURE codes or both. Since a FUNCTION clause affects the entire expression it can contain only FUNCTION codes.

FUNCTION Code	Purpose
A	Allows alphabetic characters only (no spaces or symbols).
B	Left-justifies numeric data within the output field.
D	Uses the current SET DATE format.
E	Edits date type data as a BRITISH date.
I	Centers text within a field.
J	Right-justifies text within a field.
K	Selects an entire field for editing when the cursor is moved to the field.
L	Displays leading zeros (instead of spaces) in numeric output. Use with numeric data only.
M <list>	Creates multiple preset choices. The list is a comma-delimited collection of items. Individual items within the list cannot contain embedded commas. If <memvar> or <field> initially do not contain one of the items in the list when READ is issued, the first item in the list is displayed. To scroll through the list, press the Spacebar or type the first letter of an item. To choose one of the items and move to the next control, press Enter. Use only with character data.
R	Displays a format mask in an @ ... GET editing region. These mask characters are not stored to the field when you exit the @ ... GET editing region. Use only with character or numeric data.

FUNCTION Code	Purpose
S<n>	Limits the display width to n characters. You can scroll within the region with the cursor control keys. Use only with character data.
T	Trims leading and trailing blanks from <memvar> or <field>.
Z	Displays <memvar> or <field> as blank if its numeric value is 0. Use with numeric data only.
!	Converts alphabetic characters to upper-case. Use with character data only.
^	Displays numeric data using scientific notation. Use with numeric data only.
\$	Displays data in a currency format. The currency symbol appears before or after the value depending on the current setting of SET CURRENCY. If CURRENCY is SET LEFT, the \$ function code cannot be used. Use with numeric data only.

PICTURE Codes

A PICTURE expression can include any characters, but only the characters listed below actively participate in display and editing.

PICTURE Code	Purpose
A	Allows alphabetic characters only.
L	Allows logical data only.
N	Allows letters and digits only.
X	Allows any character.
Y	Allows logical Y, y, N and n only. Converts y and n to Y and N, respectively.
9	Allows only digits in character data. Allows digits and signs in numeric data.
#	Allows digits, blanks and signs.
!	Converts lower-case letters to upper-case letters.

PICTURE Code	Purpose
\$	Displays the current currency symbol specified by SET CURRENCY. By default, the symbol is placed immediately before or after the field. However, the currency symbol and its placement (SET CURRENCY), the separator character (SET SEPARATOR) and the decimal character (SET POINT) can all be changed. Can only be used in @ ... GET when SET CURRENCY is LEFT.
*	Asterisks are displayed in front of a numeric value. Use with a dollar sign \$ for check protection.
.	Specifies the decimal point position.
,	Use to separate digits to the left of the decimal point.

FONT <expC3> [, <expN1>] — FoxPro for Windows only

The character expression <expC3> is the name of the font and the numeric expression <expN1> is the font size.

For example, this short program example displays the CONTACT field for editing. In FoxPro for Windows the text in the editing region is displayed in 16 point Roman font:

```

CLOSE DATABASES
USE customer
DO CASE
    CASE _WINDOWS
        @ 2, 2 GET contact FONT 'ROMAN',16
    CASE _DOS
        @ 2, 2 GET contact
ENDCASE
READ

```

If you include the FONT clause but omit the font size <expN3>, a 10 point font is used.

If the FONT clause is omitted and the editing region is placed in the main FoxPro window, the main FoxPro window font is used. If the FONT clause is omitted and the editing region is placed in a user-defined window, the user-defined window font is used.

If the font you specify is not available, Windows substitutes a font with font characteristics similar to the requested font.

STYLE <expC4> — FoxPro for Windows only

In FoxPro for Windows, include the STYLE clause to specify a font style for an @ ... GET editing region. The styles that are available for a font are determined by Windows. If the font style you specify is not available, Windows substitutes a font style with similar characteristics.

The font style is specified with <expC4>. If the STYLE clause is omitted, the standard font style is used.

Character	Font Style
B	Bold
I	Italic
N	Normal
O	Outline
Q	Opaque
S	Shadow
—	Strikeout
T	Transparent
U	Underline

If you include T to create a transparent editing region, the background color is ignored by the region.

You can include more than one character to specify a combination of font styles. For example, the following short program create an @ ... GET text editing region. The CONTACT field appears in the editing region appears in Bold Italic in FoxPro for Windows.

```
CLOSE DATABASES
CLEAR
USE customer

DO CASE
CASE _WINDOWS
    @ 2, 2 GET contact STYLE 'BI'
CASE _DOS
    @ 2, 2 GET contact
ENDCASE

READ
```


DEFAULT <expr1>

If you specify a memory variable for the @ ... GET editing region that doesn't exist, it is automatically created and initialized if you include DEFAULT. However, an array element isn't created if you specify an array element in a DEFAULT clause. The DEFAULT clause is ignored if the memory variable already exists or you specify a field.



If the DEFAULT clause isn't included and the memory variable <memvar> doesn't exist, the error message "Variable not found" is displayed.

The DEFAULT expression <expr1> determines the type of memory variable created and its initial value.

ENABLE | DISABLE

Including DISABLE prevents access to an @ ... GET editing region. The editing region is displayed in the disabled colors and cannot be selected.

By default, @ ... GET editing regions are enabled. You can include ENABLE as a reminder in a program that a GET editing region can be accessed.



If all the GETs in user-defined window are disabled, the window won't remain on top. If all the GETs in the current READ are disabled, the READ is terminated.

MESSAGE <expC5>

The MESSAGE clause character expression <expC5> is displayed when the @ ... GET editing region is selected. In FoxPro for MS-DOS the message is centered on the last line of the desktop and temporarily cancels any SET MESSAGE expression.

In FoxPro for Windows, the message is placed in the Windows-style status bar. If the Windows-style status bar has been turned off with SET STATUS BAR OFF, the message is placed on the last line of the main FoxPro window.

[OPEN] WINDOW <window name>

Include the WINDOW clause to edit a memo field in a user-defined window. The user-defined window must first be created with DEFINE WINDOW. The word Memo is displayed when you issue @ ... GET WINDOW.

To open the memo editing window, double-click Memo or place the cursor on the word Memo and press Ctrl+Home, Ctrl+PgUp or Ctrl+PgDn.

If OPEN is included, the memo editing window is automatically opened when READ or READ CYCLE is issued.

To exit the window and save your editing changes in FoxPro for MS-DOS, click the window's close box (if available) or press Ctrl+W. Press Escape to discard your changes.

To exit the window and save your editing changes in FoxPro for Windows, choose **Close** from the window's **Control** menu (if available) or press Ctrl+W. Press Escape to discard your changes.



A better method for editing memo fields is using @ ... EDIT instead of @ ... GET. @ ... EDIT creates a text editing window with a scroll bar, and the contents of the memo field are displayed when @ ... EDIT is issued.

RANGE [<expr2>] [, <expr3>]

Use the RANGE clause with character, date and numeric data to specify a range of acceptable values. If the value you enter in the @ ... GET editing region isn't within the specified range, a message showing the correct range is displayed. To override the default range message, use ON READERROR.

The lower boundary of the range is specified with <expr2>, the upper boundary with <expr3>. <expr2> and <expr3> must be character, numeric or date expressions that correspond to the data in the memory variable, array element or field. Either <expr2> or <expr3> can be omitted, but not both. If one boundary is omitted, the data you enter is checked against the specified boundary only.



The range isn't checked if you press Enter without changing the memory variable, array element or field.

SIZE <expN2>, <expN3>

SIZE lets you control the length and height of an @ ... GET text editing region. By default, a text editing region is one row high. The size of the region is determined by the length of the memory variable, array element or field or a PICTURE clause.

The height of the text editing region in rows is specified with <expN2> and the width in columns is specified with <expN3>.

In FoxPro for Windows, the editing region font determines the size of the editing region. The editing region font is specified with the FONT clause. If the FONT clause is omitted, the editing region uses the font of its parent window (the main FoxPro window or a user-defined window).

VALID <expL1> | <expN4>

Use VALID to validate input. When you attempt to exit the GET editing region, the VALID expression is evaluated.

A VALID clause greatly simplifies data validation when used with a user-defined function (UDF). If a UDF is called within a VALID clause in @ ... GET, the UDF should return a logical or numeric value.



Unlike the RANGE clause, a VALID clause is always executed when you exit the GET editing region unless you press Escape. The RANGE clause is only executed when a change is made to the memory variable, array element or a field.

<expL1>

If <expL1> evaluates to a logical true (.T.), the input is considered correct and the editing region is exited.

If <expL1> evaluates to false (.F.), the value you entered is considered incorrect and a message is displayed directing you to reenter the data after pressing the Spacebar.

If a UDF validation routine performs a replacement on a field and the validation routine then returns false, the field replacement occurs but the previous field value is restored when you return from the UDF to the GET editing region.

<expN4>

A VALID clause that includes a numeric expression is used to specify which object is activated after you exit the GET editing region. Objects are @ ... GET input fields, check boxes, lists, popups, spinners, text editing regions and each individual button in a set of push, radio and invisible buttons. The numeric expression <expN4> has one of the three effects:

- When <expN4> is 0, the cursor remains in the GET editing region. The MESSAGE <expC5> and ERROR <expC6> error messages are suppressed. A special error message routine can be written as part of a UDF called by VALID.

If a UDF validation routine performs a replacement on a field and the validation routine returns 0, the field replacement occurs but the previous field value is restored when you return to the GET editing region.

- When <expN4> is positive, <expN4> specifies the number of objects to advance. For example, when the cursor is positioned on a GET editing region and VALID returns 1, the next object is activated. If <expN4> is greater than the number of remaining objects, the READ is terminated (unless READ CYCLE is issued to activate the objects).
- When <expN4> is negative, <expN4> specifies the number of objects to move back. For example, when the cursor is positioned on a GET editing region and VALID returns -1, the previous object is activated. If <expN4> moves back past the first object, the READ is terminated (unless READ CYCLE is issued to activate the objects).

ERROR <expC6>

ERROR <expC6> lets you specify a custom error message displayed when a VALID clause evaluates to false (.F.). FoxPro displays <expC6> in place of the default error message.

WHEN <expL2>

WHEN allows or prohibits access to a GET editing region based on the value of <expL2>, which must be true (.T.) before the GET editing region can be accessed. If WHEN is specified and <expL2> is false (.F.), the GET editing region cannot be accessed and the next object is activated.

**COLOR SCHEME <expN5>
| COLOR <color pair list>**

If you do not include a COLOR clause, the colors of an @ ... GET editing region are determined by the color scheme for the desktop or the main FoxPro window; if an @ ... GET editing region is placed in a user-defined window, the window's color scheme determines the editing region's colors.

Only the second color pair in a color scheme or color pair list affects the color of an @ ... GET editing region.

The color of an @ ... GET editing region can be specified by including the number of an existing color scheme in the COLOR SCHEME clause or a set of color pairs in the COLOR clause.

A color scheme is a set of 10 predefined color pairs. The color pairs in a color scheme can be changed with SET COLOR OF SCHEME. In FoxPro for MS-DOS the color pairs in a color scheme can also be changed in the Color Picker.

A color pair is a set of two letters separated by a forward slash. The first color letter specifies the foreground color and the second letter specifies the background color.

For example, this color pair specifies a red foreground on a white background:

R/W

For a list of colors and their corresponding color letters, see the SET COLOR commands section in this chapter or the Color Table by Color Pair topic in the online help facility.

In FoxPro for Windows, a color pair can also be specified with a set of 6 RGB (Red Green Blue) color values separated by commas. The first 3 color values specify the foreground color and the second 3 color values specify the background color. The color values can range from 0 through 255.

In FoxPro for Windows, the R/W color pair in the example above can also be specified with this RGB color pair:

RGB(255,0,0,255,255,255)

Only color pairs 2, 5, 6 and 10 in a color scheme or color pair list affect the @ ... GET editing region colors.

@ ... GET

Color Pair Number	Check Box Attribute
2	@ ... GET editing region
5	Message
6	Selected @ ... GET editing region
10	Disabled @ ... GET editing region

Example

The following example demonstrates how you can combine @ ... SAY and @ ... GET into a single command and the COLOR, PICTURE, RANGE and WHEN clauses. Two tables/.DBFs are opened and a relation is set between the CUSTOMER and OFFICES tables/.DBFs to extract information from both tables/.DBFs.

```
SET TALK OFF
CLEAR
CLOSE DATABASES
```

```
USE customer IN 0          && Parent
USE offices  IN 0 ORDER ono && Child
SET RELATION TO ono INTO offices && Set the relation
```

```
@ 1, 0 TO 18, 79 DOUBLE
@ 3, 13 SAY 'Company: '   GET company COLOR gr+/b, r/w
@ 5, 13 SAY 'Contact: '   GET contact COLOR gr+/b, r/w
@ 7, 13 SAY 'Address: '   GET address COLOR gr+/b, r/w
@ 9, 13 SAY 'City: '      GET city COLOR gr+/b, r/w
@ 11, 13 SAY 'State: '    GET state PICTURE '!!' COLOR gr+/b, r/w
@ 11, 28 SAY 'Zip: '      GET zip COLOR gr+/b, r/w
@ 15, 13 SAY 'Ytdsales: '
@ 15, 23 GET offices.ytdsales PICTURE '999999.99' ;
    RANGE 100,999999 ;
    MESSAGE 'Enter ytdsales from 100 to 999999.'
READ
CLEAR
```

Controls

What They Are

The next eight sections discuss the controls available in FoxPro. *Controls* supply or request information and can designate, confirm or cancel actions. They can appear on the desktop in FoxPro for MS-DOS, in the main FoxPro window in FoxPro for Windows, and in user-defined windows and dialogs. Check boxes, radio buttons and popups are examples of commonly used controls.

If you use the Screen Builder to create your data entry screens, you may not have to use the commands to create controls at all. The Screen Builder automatically generates the commands that create the controls.

Most controls are created with different variations of @ ... GET. Text editing regions are created with @ ... EDIT. Issue READ or READ CYCLE to activate controls.

The FoxPro interface uses controls extensively. The *FoxPro User's Guide* and the *FoxPro Getting Started* manuals describe controls and their operation.

Control Summary

Here is a summary of each FoxPro control and its use. FoxPro controls include check boxes, invisible buttons, list, popups, push buttons, radio buttons, spinners and text editing regions.

Check Boxes

On the screen, a check box is a box with descriptive text to the right.

A check box in FoxPro for MS-DOS looks like this:

☐ **Titles**

A check box in FoxPro for Windows looks like this:

☒ **Titles**

A check box is used to toggle between two states such as true (.T.) and false (.F.) or yes and no.

In FoxPro for Windows, a .BMP (bitmap picture) can be used as a check box.

Invisible Buttons

Invisible buttons are rectangular regions of the desktop, the main FoxPro window or a user-defined window that you can select and choose. Characters and pictures (in FoxPro for Windows) are often placed over the “hot” invisible button regions, allowing you to create custom designed buttons.

Use @ ... SAY to place characters and pictures over the invisible buttons.

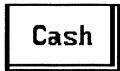
Lists

A list is a vertical list of options inside a box, often with a scroll bar to the right. Lists are often used to display the directories, files or fields from which you can choose.

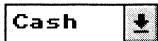
Popups

A popup is a rectangle containing a set of options. When you select a popup, FoxPro displays a list of options, only one of which you can choose.

Popups in FoxPro for MS-DOS look like this:



Popups in FoxPro for Windows look like this:



Push Buttons

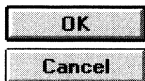
In FoxPro for MS-DOS, a push button is descriptive text that you can choose.

Push buttons in FoxPro for MS-DOS look like this:

< OK >
< Cancel >

In FoxPro for Windows, a push button is a raised rectangle with text on top, and .BMPs can be used as push buttons.

Push buttons in FoxPro for Windows look like this:



Typically, a push button is used to trigger an action.

Radio Buttons

As their name suggests, radio buttons are similar to the buttons of a car radio — choosing a button makes your choice current and releases your previous choice. A bullet in a button indicates that it is the current choice.

Radio buttons are typically used to display several options, of which you can only choose one of the options.

Radio buttons in FoxPro for MS-DOS look like this:

(.) **None**
 () **Single**
 () **Double**

Radio buttons in FoxPro for Windows look like this:

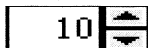
☐ **None**
☒ **Single**
☐ **Double**

In FoxPro for Windows, .BMPs can be used as radio buttons.

Spinners — FoxPro for Windows only

A *spinner* is a control that lets you “spin” through a set of numeric values displayed in a text box. Clicking on the up and down arrows changes the numeric value. You can also enter a value directly into the spinner’s text box. The value in the spinner is stored to a memory variable, an array element or a field.

Spinners look like this:



Text Editing Regions

A text editing region is a rectangular region in which you can modify or enter text. The text you enter is stored to a memory variable, an array element or a field.

All standard FoxPro editing features, such as cutting, copying and pasting, are available in a text editing region. Unlike other FoxPro controls, this command uses EDIT in its syntax instead of GET.

Text editing regions are typically used to update and modify memo fields and long character fields.

How They Work

The @ ... GET command creates most controls (@ ... EDIT creates text editing regions). The FUNCTION and PICTURE clauses of @ ... GET determine which type of control is created. The FUNCTION and PICTURE clauses also contain the prompts associated with most kinds of controls.

Many controls trigger an action. The action to be taken is specified by the VALID clause.

Controls are activated by READ or READ CYCLE. Depending on the options you specify, READ or READ CYCLE can either terminate or continue when a control is chosen.

Hot Keys

You can define hot keys for check boxes, popups, push buttons and radio buttons. *Hot keys* provide a shortcut method of choosing a button or option. A hot key is a key that you can press to immediately choose the desired item. The hot key usually corresponds to first letter in a prompt or option; the letter is highlighted (FoxPro for MS-DOS) or underlined (FoxPro for Windows).

A hot key is created by placing a back slash and a less than sign (\<) before the hot key character. Place any other special characters (\!, \\", ...) before the \< hot key designator. For example:

```
\!\<Hot key
```

Disabled Controls

To make a control (or part of it) unavailable for use, you can disable it. Disabled controls, buttons and options appear in disabled colors and cannot be selected or chosen.

Syntax Summary

The following table summarizes the different syntax, options, and defaults for the various controls. Refer to the individual commands for information on these controls.

Control	FUNCTION/ PICTURE Code	Additional PICTURE/FUNCTION Code Options	Default	Special Characters
Check box	*C	N - Non-terminating T - Terminating	Non-terminating	\< - Hot key \\ - Disable
Invisible button	*I	N - Non-terminating T - Terminating H - Horizontal V - Vertical	Non-terminating, Vertical	\\ - Disable
List	&	N - Non-terminating T - Terminating	Terminating	N/A
Popup	^	N - Non-terminating T - Terminating	Non-terminating	\< - Hot key \ - Disable
Push button	*	N - Non-terminating T - Terminating H - Horizontal V - Vertical	Terminating, Vertical	\< - Hot key \\ - Disable \! - Default \? - Escape
Radio button	*R	N - Non-terminating T - Terminating H - Horizontal V - Vertical	Non-terminating, Vertical	\< - Hot Key \\ - Disable
Spinner	N/A	See the Picture and Function tables in the Spinner topic	Terminating	N/A
Text editing region	N/A	I - Centered J - Right-justified	Left-justified	N/A

@ ... GET — Check Boxes

Purpose	Creates a check box or a picture check box
Syntax	<pre>@ <row, column> GET <memvar> <field> FUNCTION <expC1> PICTURE <expC2> [FONT <expC3> [, <expN1>]] [STYLE <expC4>] [DEFAULT <expr>] [SIZE <expN2>, <expN3>] [ENABLE DISABLE] [MESSAGE <expC5>] [VALID <expL1> <expN4>] [WHEN <expL2>] [COLOR SCHEME <expN5> COLOR <color pair list>]</pre>
Remarks	FONT and STYLE clauses, picture check boxes and support for RGB color pairs — FoxPro for Windows only
See Also	@ ... SAY, @ ... GET, Controls, _CUROBJ, OBJNUM(), READ, SET KEYCOMP, SHOW GET, User-Defined Functions

Description With this variation of @ ... GET you can create a check box or a picture check box. A check box is used to toggle between two states, such as true (.T.) and false (.F.) or yes and no.

If you use the Screen Builder to create your data entry screens, you may not have to use this command at all. The Screen Builder automatically generates the commands that create check boxes and picture check boxes.

A check box is box with descriptive text to its right. The string of text, called a *prompt*, indicates what the check box controls. The text of the prompt is specified by the FUNCTION or PICTURE clause. When a condition is true, an X is displayed. Only one check box may be created in a single @ ... GET. Issue READ or READ CYCLE to activate a check box.

You can create a picture check box in FoxPro for Windows. A .BMP (bitmap picture) on a button replaces the check box and the check box prompt.

A check box in FoxPro for MS-DOS looks like this:

[X] Titles

In FoxPro for MS-DOS, pressing the Spacebar or Enter or clicking the check box toggles the check box from one state to the other.

A check box in FoxPro for Windows looks like this:

☒ Titles

In FoxPro for Windows, pressing the Spacebar or Enter or clicking the check box toggles the check box from one state to the other if KEYCOMP is set to DOS. If KEYCOMP is set to WINDOWS, which is the default, you can toggle the check box state by pressing the Spacebar or clicking on the check box. Press Enter to move to the next object.

Clauses

<row, column>

Row and column are numeric expressions with values 0 or greater that determine where the check box appears.

The first row is number 0 on the desktop, main FoxPro window or a user-defined window. In FoxPro for MS-DOS, row 0 is the row the FoxPro system menu bar occupies. See SET SYSMENU in this chapter for information about manipulating the system menu bar so you can place output on row 0. In FoxPro for Windows, row 0 is the row immediately under the FoxPro system menu bar. Rows are numbered from top to bottom.

The first column is number 0 on the desktop, main FoxPro window or a user-defined window. Columns are numbered from left to right.

When a check box is placed in a user-defined window, the row and column coordinates are relative to the user-defined window, not the desktop in FoxPro for MS-DOS or the main FoxPro window in FoxPro for Windows.

In FoxPro for Windows, a position in the main FoxPro window or in a user-defined window is determined by the font of the main FoxPro window or the user-defined window. Most fonts Microsoft Windows uses can be displayed in a wide variety of sizes, and some are proportionally spaced. A row corresponds to the height of the current font; a column corresponds to the average width of a letter in the current font. See the Font topic in the FoxPro online help for additional information about fonts.

To facilitate precise positioning of a check box in the main FoxPro window or in a user-defined window, you can use decimal fractions for row and column coordinates in FoxPro for Windows.

<memvar> | <field>

When you check or uncheck a check box, your choice is stored to a memory variable, an array element or a field, which you specify with <memvar> or <field>. The <memvar> or <field> must be of numeric or logical type.

When a check box is initially displayed on the screen or in a window, the box is checked if <memvar> or <field> contains a non-zero numeric value or a true (.T.) value. The box isn't checked if <memvar> or <field> is zero or false (.F.).

READ or READ CYCLE activates the check box. The state of the check box when the READ is terminated determines the value stored to the memory variable, array element or field — 1 or .T. if checked and 0 or .F. if unchecked.

FUNCTION <expC1> | PICTURE <expC2>

When creating a check box, you must include the FUNCTION clause, the PICTURE clause or both. There is no advantage for any of the three methods. The FUNCTION clause or the PICTURE clause contains the check box specification code *C.

The FUNCTION clause character expression <expC1> must begin with *C. To create the prompt, include a space after *C followed by the text of the prompt. For example, this clause creates a check box with the prompt Titles:

```
... FUNCTION '*C Titles' ...
```

The PICTURE clause character expression <expC2> uses the same syntax as the FUNCTION clause character expression, except the PICTURE clause expression must begin with @ followed by *C. For example, this clause creates a check box with the prompt Titles:

```
... PICTURE '@*C Titles' ...
```

You can also include *both* the FUNCTION and PICTURE clauses to create a check box. If both are included, the FUNCTION character expression <expC1> must contain *C to create the check box. The PICTURE character expression <expC2> must include the prompt. For example:

```
... FUNCTION '*C' PICTURE 'Titles' ...
```

The following examples illustrate the various forms of syntax you can use to create a check box. In all the examples the check box is placed in the second row and column. The state of the box (checked or unchecked) is stored to the memory variable MCHOICE. Each example creates the same check box.

FUNCTION clause only:

```
STORE 1 TO mchoice
@ 2,2 GET mchoice FUNCTION '*C Titles'
READ
```

```
STORE 1 TO mchoice
STORE '*C Titles' TO mprompt
@ 2,2 GET mchoice FUNCTION mprompt
READ
```

PICTURE clause only:

```
STORE 1 TO mchoice
@ 2,2 GET mchoice PICTURE '@*C Titles'
READ
```

```
STORE 1 TO mchoice
@ 2,2 GET mchoice PICTURE '@*C' + ' Titles'
READ
```

FUNCTION and PICTURE clauses:

```
STORE 1 TO mchoice
@ 2,2 GET mchoice FUNCTION '*C' PICTURE ' Titles'
READ
```

Picture Check Boxes

In FoxPro for Windows, a check box can also be a .BMP (bitmap picture) displayed on a button. A picture check box mimics the *behavior* of a check box with a text prompt. The .BMP is displayed but the check box isn't displayed. Like checking a check box, choosing the picture toggles the value of <memvar> or <field> between .T. and .F. or 1 and 0.

To use a .BMP in a check box, add B to the check box specification code. The FUNCTION and PICTURE clause specification codes that create a check box are *C or @*C, respectively. To create a check box with picture prompts, use the codes *CB and @*CB.



The pictures aren't clipped, reduced or enlarged to fit the check box button. Use the SIZE clause to adjust the size of the check box button to accommodate the picture.

Picture Masks and Check Boxes

In FoxPro for Windows, a picture check box has three states: up, down and disabled. FoxPro automatically controls the appearance a picture check box when it is in each of these three states, but you can override the default appearance by using a picture mask. A mask is a monochrome .BMP file with an .MSK extension. FoxPro automatically looks for a mask for a .BMP file in the same directory where the .BMP file is located. The mask must have the same base name as the .BMP file and an extension of .MSK.

A mask is used to control the transparent areas of a picture check box. By default, the white areas are transparent. If a mask is present, the white areas of the mask, not the .BMP, are transparent.

In most cases, a mask isn't necessary. If you don't need anything in your picture to appear white and your .BMP has a white background, the picture check box will appear as desired in the up, down and disabled states.

When a picture check box has a mask and is in the up or down state, any white areas in the .BMP appear transparent, allowing the color of the button face to show through. However, you can maintain the white color of certain areas.

Suppose that you have a .BMP of a dog on a white background and white eyes, and the button face is red. You want the background of the button to appear red but you want the dog's eyes to be white, not red. Make a mask that is the same size as the .BMP but includes only two colors, black and white. Leave the background of the mask white but make the dog — including his eyes — completely black. When the button is displayed, the background is red to match the button face but the dog's eyes are white.

When a picture check box is disabled, any white areas in the .BMP appear transparent so the color of the button face shows through. Any non-white areas appear dark grey. If the button has a .MSK, all white areas in the mask are transparent so the color of the button face show through, and all black areas appear dark grey.

PICTURE and FUNCTION Options N and T

Two options can be placed after the specification code *C in the FUNCTION or PICTURE clause to specify whether the READ is terminated when a check box is chosen:

Option	Description
N	Do not terminate the READ when the box is chosen. This is the default behavior.
T	Terminate the READ when the box is chosen.

Check Boxes with Special Features

You can assign a hot key to the check box or disable the check box by including special characters when defining the prompt.

Hot Keys

A hot key allows you to immediately choose or change the state of the check box. To assign a hot key, place a backslash and a less than sign (\<) before the desired character of the check box prompt. The hot key is a highlighted character (FoxPro for MS-DOS) or an underlined character (FoxPro for Windows).



A hot key doesn't choose the check box if the current object is an @ ... GET input field, a text editing region, a popup or a list.

If the current object is an @ ... GET input field or a text editing region, pressing the hot key enters the corresponding letter in the field or the text editing region. If the current object is a popup or a scrollable list, pressing the hot key selects the first option in the popup or list whose prompt begins with the letter corresponding to the hot key.



In FoxPro for Windows, if the current object is an @ ... GET input field, text editing region, popup or list and KEYCOMP is set to WINDOWS, you can press Alt and the hot key to move to the check box and choose it. If you create a picture check box, you cannot create a hot key for the picture check box.

The following example creates a check box with the prompt Titles and assigns the hot key T to it:

```
STORE 1 TO mchoice
@ 2,2 GET mchoice FUNCTION '*C' PICTURE ' \<Titles'
READ
```

Disabled Check Boxes

A disabled check box cannot be selected or chosen and is displayed in disabled colors. To disable a check box, place two backslashes (\\) before the check box prompt or use the DISABLE clause. The following example disables the check box created earlier:

```
STORE 1 TO mchoice1, mchoice2, mchoice3
@ 2,2 GET mchoice1 FUNCTION '*C \\Titles'
@ 4,2 GET mchoice2 FUNCTION '*C Titles' DISABLE
@ 6,2 GET mchoice3 FUNCTION '*C Titles'
READ
```

FONT <expC3> [, <expN1>] — FoxPro for Windows only

Include FONT to specify a font and font size for the check box prompt. The character expression <expC3> is the name of the font, and the numeric expression <expN1> is the font size. For example, the following command displays the prompt in 16 point Roman font

```
@ 2,2 GET mchoice FUNCTION '*C Titles' DEFAULT 1 ;
    FONT 'ROMAN',16
```

If you include the FONT clause but omit the font size <expN1>, a 10 point font is used.

If the FONT clause is omitted and the check box is placed in the main FoxPro window, the main FoxPro window font is used. If the FONT clause is omitted and the check box is placed in a user-defined window, the user-defined window font is used.

If the font you specify is not available, Windows substitutes a font with similar font characteristics.

STYLE <expC5> — FoxPro for Windows only

In FoxPro for Windows, include the **STYLE** clause to specify a font style for a check box. The styles that are available for a font are determined by Windows. If the font style you specify is not available, Windows substitutes a font style with similar characteristics.

The font style is specified with <expC5>. If the **STYLE** clause is omitted, the standard font style is used.

Character	Font Style
B	Bold
I	Italic
N	Normal
O	Outline
Q	Opaque
S	Shadow
—	Strikeout
T	Transparent
U	Underline

You can include more than one character to specify a combination of font styles. For example, the following clause specifies Bold Italic:

```
STYLE 'BI'
```

If **T** is included to create a transparent check box, the check box is filled with the background color. If **Q** is included, the check box and the check box prompt is filled with the background color.

DEFAULT <expr>

When you check or uncheck a check box, the state of the box is saved in a memory variable, an array element or a field. If you specify a memory variable that doesn't exist, it is automatically created and initialized if the **DEFAULT** clause is included. However, an array element isn't created if you specify an array element in a **DEFAULT** clause. The **DEFAULT** clause is ignored if the memory variable already exists or you specify a field.



If the DEFAULT clause is omitted and the memory variable `<memvar>` doesn't exist, the error message "Variable not found" is displayed.

The DEFAULT expression `<expr>` determines the type of memory variable created and its initial value. It must be a numeric or logical expression. Here are examples of DEFAULT clauses for check boxes:

```
@ 2,2 GET mchoice FUNCTION '*C Titles' DEFAULT .T.  
READ  
  
STORE .T. TO button  
@ 2,2 GET mchoice FUNCTION '*C Titles' DEFAULT button  
READ
```

SIZE `<expN2>`, `<expN3>`

The numeric expression `<expN2>` specifies the height of a check box. For check boxes in FoxPro for MS-DOS, this expression is ignored because a check box is always one line high. However, you must include `<expN2>` if you include `<expN3>` to specify the width of the prompt.

In FoxPro for Windows, you can specify a height for the check box. If `<expN2>` is 1, the focus rectangle around the check box prompt will be clipped. In this case, include a larger value for `<expN2>`.

By default, the width of a check box is determined by the length of the prompt text. You can use `<expN3>` to make the check box prompt wider than the default. The numeric expression `<expN3>` specifies the width (in columns) of a check box.

In FoxPro for Windows, the check box font determines the size of the check box. The check box font is specified with the FONT clause. If the FONT clause is omitted, the check box uses the font of its parent window (the main FoxPro window or a user-defined window).

ENABLE | DISABLE

By default, a check box is enabled when READ is issued. You can prevent a check box from being activated when READ is issued by including DISABLE. A disabled check box cannot be selected and is displayed in disabled colors. Use SHOW GET ENABLED to enable a disabled check box.

MESSAGE <expC5>

The MESSAGE clause character expression <expC5> is displayed when a check box is selected. The message is centered by default on the last line of the desktop in FoxPro for MS-DOS; the message location can be specified with SET MESSAGE.

In FoxPro for Windows, the message is placed in the Windows-style status bar. If the Windows-style status bar has been turned off with SET STATUS BAR OFF, the message is centered on the last line of the main FoxPro window.

VALID <expL1> | <expN4>

You can include an optional VALID expression <expL1> or <expN4> that is evaluated when a check box is *chosen* (checked or unchecked). That is, VALID isn't evaluated when you select (move to) the check box, but when you actually choose the check box.

Typically, <expL1> or <expN4> is a user-defined function. With user-defined functions you can select, enable or disable other objects, open a Browse window, open another data entry screen or move to a new record. CLEAR READ can be included in a user-defined function to terminate the READ.

<expL1>

When a logical value <expL1> is returned to the VALID clause, the logical value is ignored and the check box remains the active control. However, you can specify a UDF that returns a logical value to the VALID clause and activates another object.

Example

The following example demonstrates how a VALID clause UDF is executed when a check box is chosen. The check box remains the current control when the VALID clause UDF is executed because a logical false (.F.) is returned by the UDF.

```
CLEAR
STORE 1 TO mchoice1, mchoice2, mchoice3
@ 2,2  GET mchoice2
@ 4,2  GET mchoice1 FUNCTION '*C \<Titles' ;
      VALID showmes( ) ;
      MESSAGE 'Execute the VALID. ' + ;
        'Select this check box using {Spacebar}.'
@ 6,2  GET mchoice3
READ CYCLE  && Press {Esc} to exit the READ

FUNCTION showmes
WAIT WINDOW NOWAIT 'The VALID clause was executed.'
RETURN .F.
```

<expN4>

A VALID clause that includes a numeric expression <expN4> is used to specify which object is activated after a check box is chosen. Objects are @ ... GET input fields, check boxes, lists, popups, spinners, text editing regions and each individual button in a set of push, radio and invisible buttons.

The numeric expression has one of three effects:

- When <expN4> is 0, the check box remains the active control.
- When <expN4> is positive, <expN4> specifies the number of objects to advance. For example, when the check box is selected and VALID returns 1, the next object is activated. If <expN4> is greater than the number of objects remaining, the READ is terminated (unless READ CYCLE is issued to activate the objects).
- When <expN4> is negative, <expN4> specifies the number of objects to move back. For example, when the check box is selected and VALID returns -1, the previous object is activated. If <expN4> moves back past the first object, the READ is terminated (unless READ CYCLE is issued to activate the objects).

Example

The following example demonstrates how a VALID clause UDF can activate another object by returning a numeric value. When the VALID clause UDF is executed, the UDF returns 2. The second object after the check box is activated. Since READ CYCLE was issued, the READ isn't terminated even though there are less than two objects remaining.

```
CLEAR
STORE 1 TO mchoice1, mchoice2
STORE 0 TO m.choice3
@ 2,2 GET mchoice1
@ 4,2 GET mchoice2
@ 6,2 GET m.choice3 FUNCTION '*C \<Titles' ;
    VALID showmes( ) ;
    MESSAGE 'Execute the VALID. ' + ;
        'Select this check box using {Spacebar}.'
READ CYCLE      && Press {Esc} to exit the READ

FUNCTION showmes
WAIT WINDOW NOWAIT 'The VALID clause was executed.'
RETURN 2
```

WHEN <expL2>

The optional WHEN clause allows or prohibits selection of a check box based on the logical value of <expL2>, which must evaluate to a logical true (.T.) before a check box can be selected. If <expL2> evaluates to a logical false (.F.), the check box cannot be selected and is skipped over if placed between other objects.

Example

In the following example, the check box WHEN clause returns a logical (.F.). The check box is skipped even though it isn't disabled.

```
CLEAR
STORE 1 TO mchoice1, mchoice2
STORE 0 TO mchoice3
@ 2,2 GET mchoice1
@ 4,2 GET mchoice2 FUNCTION '*C \<Titles' ;
    WHEN showmes( )
@ 6,2 GET mchoice3
READ CYCLE          && Press {Esc} to exit the READ

FUNCTION showmes
WAIT WINDOW NOWAIT 'The WHEN clause was executed.'
RETURN .F.
```

COLOR SCHEME <expN5>**| COLOR <color pair list>**

If you do not include a COLOR clause, check box colors are determined by the color scheme for the desktop or the main FoxPro window; if a check box is placed in a user-defined window, the window's color scheme determines the check box colors.

The color of a check box can be specified by including the number of an existing color scheme in the COLOR SCHEME clause or a set of color pairs in the COLOR clause.

A color scheme is a set of 10 predefined color pairs. The color pairs in a color scheme can be changed with SET COLOR OF SCHEME. In FoxPro for MS-DOS the color pairs in a color scheme can also be changed in the Color Picker.

A color pair is a set of two letters separated by a forward slash. The first color letter specifies the foreground color and the second letter specifies the background color.

For example, this color pair specifies a red foreground on a white background:

```
R/W
```

For a list of colors and their corresponding color letters, see the SET COLOR commands section in this chapter or the Color Table by Color Pair topic in the online help facility.

In FoxPro for Windows, a color pair can also be specified with a set of 6 RGB (Red Green Blue) color values separated by commas. The first 3 color values specify the foreground color and the second 3 color values specify the background color. The color values can range from 0 through 255.

In FoxPro for Windows, the R/W color pair in the example above can also be specified with this RGB color pair:

```
RGB(255,0,0,255,255,255)
```

Only color pairs 5, 6, 7, 9 and 10 in a color scheme or color pair list affect the check box colors. The color of picture check boxes is controlled by Windows

Color Pair Number	Check Box Attribute
5	Message
6	Selected check box prompt — FoxPro for MS-DOS only
7	Hot keys — FoxPro for MS-DOS only
9	Enabled check box prompt
10	Disabled check box prompt

The following example shows how to override the color scheme of the desktop or main FoxPro window with another predefined color scheme:

```
ACTIVATE SCREEN  
STORE 1 TO mchoice  
@ 2,2 GET mchoice FUNCTION '*C Titles' COLOR SCHEME 4  
READ
```


The following example creates check boxes with the prompt Titles, the hot key T and the following color characteristics:

- A selected box is shown with a bright white prompt on a blue background (W+/B). In FoxPro for Windows, a focus rectangle is used to indicate the current object.
- Hot key characters are shown in red on a blue background (R/B) in FoxPro for MS-DOS. In FoxPro for Windows, the hot key character is underlined.
- An enabled check box is shown with a yellow prompt on a blue background (GR+/B).
- A disabled check box is shown with a gray prompt on a blue background (W/B).

When you skip a color pair, you must include a comma where the color pair is omitted.

Here are the commands:

```
CLEAR
SET COLOR TO W/B
STORE 1 TO mchoice
@ 2,2 GET mchoice FUNCTION '*C \<Titles';
  COLOR ,,,,W+/B,R/B,,GR+/B,W/B
@ 4,2 GET mchoice FUNCTION '*C \\\Titles';
  COLOR ,,,,W+/B,R/B,,GR+/B,W/B
@ 6,2 GET mchoice FUNCTION '*C \<Titles';
  COLOR ,,,,W+/B,R/B,,GR+/B,W/B
READ
```

@ ... GET — Invisible Buttons

Purpose	Creates invisible buttons
Syntax	<pre>@ <row, column> GET <memvar> <field> FUNCTION <expC1> PICTURE <expC2> [DEFAULT <expN1>] [SIZE <expN2>, <expN3> [, <expN4>]] [ENABLE DISABLE] [MESSAGE <expC5>] [VALID <expL1> <expN5>] [WHEN <expL2>] [COLOR SCHEME <expN6> COLOR <color pair list>]</pre>
Remarks	RGB color pairs — FoxPro for Windows only
See Also	@ ... SAY, @ ... GET, Controls, _CUROBJ, OBJNUM(), READ, SHOW GETS, User-Defined Functions

Description This variation of @ ... GET creates *invisible buttons* which are rectangular regions of the desktop (FoxPro for MS-DOS), the main FoxPro window (FoxPro for Windows) or a user-defined window that you can select. You can use @ ... SAY to place characters on the rectangular button regions. When you select an invisible button, the characters in the button are highlighted. Issue READ or READ CYCLE to activate the buttons.

If you use the Screen Builder to create your data entry screens, you may not have to use this command at all. The Screen Builder automatically generates the commands that create invisible buttons.

Clauses **<row, column>**

The first button in a set of invisible buttons is placed at the location specified by <row, column>. The row and column coordinates are numeric expressions. Row and column values can range from 0 through the maximum number of rows and columns on the desktop (FoxPro for MS-DOS), the main FoxPro window (FoxPro for Windows), or a user-defined window. Rows are numbered from top to bottom, and columns are numbered from left to right.

In FoxPro for Windows the position and size of invisible buttons in the main FoxPro window or in a user-defined window is determined by the font of the main FoxPro window or the user-defined window. Most Windows fonts can be displayed in a wide variety of sizes and some are proportionally spaced. Thus, positions and sizes depend on the current font. A row corresponds to the height of the current font. A column corresponds to the average width of the current font.

To facilitate precise positioning of invisible buttons in the main FoxPro window or in a user-defined window, you can use decimal fractions for row and column coordinates in FoxPro for Windows.

<memvar> | <field>

When you choose an invisible button, a number corresponding to your choice is stored to a memory variable, an array element or a field. For example, if you create four invisible buttons and choose the third button, 3 is stored to the memory variable, array element or field named by <memvar> or <field>. The initial value of <memvar> or <field> determines which button is initially selected.

FUNCTION <expC1> | PICTURE <expC2>

When creating invisible buttons, you must include the FUNCTION clause, the PICTURE clause or both. There is no advantage to any of the three methods. The FUNCTION or PICTURE clause contains the invisible button specification code *I.

The FUNCTION clause character expression <expC1> must begin with *I. To create more than one invisible button, include a semicolon for each additional button. For example, this clause creates three invisible buttons:

```
... FUNCTION '*I ;;' ...
```

Note that the clause contains only two semicolons.

The PICTURE clause character expression <expC2> uses the same syntax as the FUNCTION character expression, except the PICTURE clause expression must begin with @ followed by *I. For example, this clause creates three invisible buttons:

```
... PICTURE '@*I ;;' ...
```

You can also include *both* the FUNCTION and PICTURE clauses to create invisible buttons. If both are included, FUNCTION <expC1> must contain *I to create the invisible buttons and can also include a space followed by semicolons to create *additional* invisible buttons. PICTURE <expC2> can include semicolons to create additional invisible buttons.

Examples The following examples illustrate the various forms of syntax you can use to create invisible buttons. Five invisible buttons are created and a number corresponding to each button chosen is stored to MCHOICE. The first button is placed in the second row and column. Each button is two rows high and four columns wide, and the buttons are separated by a single row.

FUNCTION clause only:

```
STORE 1 TO mchoice
@ 2,2 GET mchoice FUNCTION '*I ;;;;' SIZE 2, 4, 1
READ
```

```
STORE 1 TO mchoice
STORE '*I ;;;;' TO mfunc
@ 2,2 GET mchoice FUNCTION mfunc SIZE 2, 4, 1
READ
```

PICTURE clause only:

```
STORE 1 TO mchoice
@ 2,2 GET mchoice PICTURE '@*I ;;;;' SIZE 2, 4, 1
READ
```

```
STORE 1 TO mchoice
@ 2,2 GET mchoice PICTURE '@*I' + ' ;;;;' SIZE 2, 4, 1
READ
```

FUNCTION and PICTURE clauses:

```
STORE 1 TO mchoice
@ 2,2 GET mchoice FUNCTION '*I' PICTURE ' ;;;;' SIZE 2, 4, 1
READ
```

```
STORE 1 TO mchoice
@ 2,2 GET mchoice FUNCTION '*I ;;' PICTURE ';;' SIZE 2, 4, 1
READ
```

PICTURE and FUNCTION Options N, T, H and V

Additional options can be combined with the *I specification code to modify the behavior (N and T options) and appearance (H and V options) of invisible buttons. These options are described below:

Option	Description
N	Do not terminate the READ when an invisible button is chosen. This is the default behavior.
T	Terminate the READ when an invisible button is chosen.
H	Position the invisible buttons in a horizontal row.
V	Position the invisible buttons in a vertical row. This is the default position.

You can combine the H and V options with the T and N options in the following ways: NH, NV, TH and TV. For example, the following clause creates a vertical set of buttons and doesn't cause the READ to terminate when one of the buttons is selected:

```
... FUNCTION '*INV ...'
```

Disabled Invisible Buttons

You can disable an invisible button so that it cannot be selected. When a button is disabled, it is displayed in disabled colors. To disable an invisible button, place two backslashes (\\) before the semicolon corresponding to the button in the FUNCTION or PICTURE clause.

In the following example, five invisible buttons are created. The second and fourth buttons are disabled by placing two backslashes before the second and fourth semicolons respectively. If you are using FoxPro for Windows, set the font for the main FoxPro window to 9 point FoxFont to draw the box characters correctly.

```
CLEAR
STORE 1 TO mchoice

*** Draw boxes for the invisible-button regions. ***
@ 4,2 SAY CHR(218) + CHR(196) + CHR(196) + CHR(191)
@ 5,2 SAY CHR(192) + CHR(196) + CHR(196) + CHR(217)
@ 7,2 SAY CHR(218) + CHR(196) + CHR(196) + CHR(191)
@ 8,2 SAY CHR(192) + CHR(196) + CHR(196) + CHR(217)
@ 10,2 SAY CHR(218) + CHR(196) + CHR(196) + CHR(191)
@ 11,2 SAY CHR(192) + CHR(196) + CHR(196) + CHR(217)
@ 13,2 SAY CHR(218) + CHR(196) + CHR(196) + CHR(191)
@ 14,2 SAY CHR(192) + CHR(196) + CHR(196) + CHR(217)
@ 16,2 SAY CHR(218) + CHR(196) + CHR(196) + CHR(191)
@ 17,2 SAY CHR(192) + CHR(196) + CHR(196) + CHR(217)

@ 4,2 GET mchoice FUNCTION '*I ;\\;;\\;' SIZE 2, 4, 1
READ
```

DEFAULT <expN1>

When you choose an invisible push button, your choice is saved in the memory variable, array element or field you specify. If you specify a memory variable that doesn't exist, it is automatically created and initialized if you include the DEFAULT. However, an array element isn't created if you specify an array element in a DEFAULT clause. The DEFAULT clause is ignored if the memory variable already exists or you specify a field.



If the DEFAULT clause isn't included and the memory variable specified with <memvar> doesn't exist, the error message "Variable not found" is displayed.

The DEFAULT expression <expN1> determines the initial value of <memvar>. Here are examples of DEFAULT clauses for invisible buttons:

```
@ 2,2 GET mchoice FUNCTION '*I ;;;;' DEFAULT 1
READ
STORE 3 TO mbutton
@ 2,2 GET mchoice FUNCTION '*I ;;;;' DEFAULT mbutton
READ
```

SIZE <expN2>, <expN3> [, <expN4>]

The SIZE clause determines the size and spacing of invisible buttons. The numeric expressions <expN2> and <expN3> control the height and width of each button respectively. The default for the height and width is 0.

By default, no space is placed between vertical buttons, and a single column is placed between horizontal buttons. The spacing between invisible buttons is determined by <expN4>. If you create vertical buttons, <expN4> designates the number of rows between buttons. If you create horizontal buttons, <expN4> designates the number of columns between buttons.

ENABLE | DISABLE

Invisible buttons are enabled by default when READ is issued. You can prevent a set of buttons from being activated when READ is issued by including DISABLE. Disabled invisible push buttons cannot be chosen and are displayed in disabled colors. To disable individual invisible buttons instead of the entire set of buttons, see Disabled Invisible Buttons, discussed earlier in this section. Use SHOW GET ENABLED to enable a set of disabled invisible buttons.

MESSAGE <expC5>

The MESSAGE clause character expression <expC5> is displayed when an invisible button is selected. In FoxPro for MS-DOS the message is centered on the last line of the desktop by default. The message location can be changed with SET MESSAGE.

In FoxPro for Windows, the message is placed in the Windows-style status bar. If the Windows-style status bar has been turned off with SET STATUS BAR OFF, the message is placed on the last line of the main FoxPro window.

VALID <expL1> | <expN5>

You can include an optional VALID expression <expL1> or <expN5> that is evaluated when an invisible button is *chosen*. That is, VALID isn't evaluated when select an invisible button, but when you actually choose a button.

Typically, <expL1> or <expN5> is a user-defined function. With a user-defined function you can select, enable or disable other objects, open a Browse window, open another data entry screen or move to a new record. CLEAR READ can be included in the user-defined function to terminate the READ.

<expL1>

When a logical value is returned to the VALID clause, the logical value is ignored and the invisible buttons remain the current control. However, you can specify a UDF that returns a logical value to the VALID clause and activates another object.

<expN5>

A VALID clause that includes a numeric expression is used to specify which object is activated after an invisible button is chosen. Objects are @ ... GET input fields, check boxes, lists, popups, spinners, text editing regions and each individual button in a set of push, radio and invisible buttons.

The numeric expression <expN5> has one of three effects:

- When <expN5> = 0, the chosen invisible button remains the active button.
- When <expN5> is positive, <expN5> indicates the number of objects to advance. For example, when you're positioned on the last button in a set of invisible buttons and VALID returns 1, the next object is activated. If <expN5> is greater than the number of objects remaining, the READ is terminated (unless READ CYCLE is issued to activate the objects).
- When <expN5> is negative, <expN5> indicates the number of objects to move back. For example, when you're positioned on the first button in a set of invisible buttons and VALID returns -1, the previous @ ... GET input field or control is activated. If <expN5> moves back past the first object, the READ is terminated (unless READ CYCLE is issued to activate the objects).

WHEN <expL2>

The WHEN clause allows or prohibits selection of the invisible buttons based on the logical value of <expL2>, which must evaluate to a logical true (.T.) before any of the invisible buttons can be selected. If <expL2> evaluates to false (.F.), none of the invisible buttons can be selected and are skipped if placed between other objects.

**COLOR SCHEME <expN6>
| COLOR <color pair list>**

If you do not include a COLOR clause, invisible button colors are determined by the color scheme for the desktop or the main FoxPro window; if invisible buttons are placed in a user-defined window, the window's color scheme determines the invisible button colors.

The colors of invisible buttons can be specified by including the number of an existing color scheme in the COLOR SCHEME clause or a set of color pairs in the COLOR clause.

A color scheme is a set of 10 predefined color pairs. The color pairs in a color scheme can be changed with SET COLOR OF SCHEME. In FoxPro for MS-DOS the color pairs in a color scheme can also be changed in the Color Picker.

A color pair is a set of two letters separated by a forward slash. The first color letter specifies the foreground color and the second letter specifies the background color.

For example, this color pair specifies a red foreground on a white background:

R/W

For a list of colors and their corresponding color letters, see the SET COLOR commands section in this chapter or the Color Table by Color Pair topic in the online help facility.

In FoxPro for Windows, a color pair can also be specified with a set of 6 RGB (Red Green Blue) color values separated by commas. The first 3 color values specify the foreground color and the second 3 color values specify the background color. The color values can range from 0 through 255.

In FoxPro for Windows, the R/W color pair in the example above can also be specified with this RGB color pair:

RGB(255,0,0,255,255,255)

Color pair 6 determines the color of the selected invisible button.

Example

The following example demonstrates invisible buttons. @ ... SAY places four sets of characters (hearts, diamonds, clubs and spades) on the desktop or the main FoxPro window. Four invisible buttons are created that line up with the characters. When an invisible button is chosen, the SHOWCARD routine is executed that indicates which button has been chosen. If you are using FoxPro for Windows, set the font for the main FoxPro window to 9 point FoxFont to draw the box characters correctly.

```
STORE 0 TO mchoice
ACTIVATE SCREEN
CLEAR
@ 2,2 SAY REPLICATE(CHR(3),2)  && Hearts
@ 3,2 SAY REPLICATE(CHR(3),2)
@ 2,10 SAY REPLICATE(CHR(4),2)  && Diamonds
@ 3,10 SAY REPLICATE(CHR(4),2)
@ 2,18 SAY REPLICATE(CHR(5),2)  && Clubs
@ 3,18 SAY REPLICATE(CHR(5),2)
@ 2,26 SAY REPLICATE(CHR(6),2)  && Spades
@ 3,26 SAY REPLICATE(CHR(6),2)
@ 1,1,4,4 BOX
@ 1,9,4,12 BOX
@ 1,17,4,20 BOX
@ 1,25,4,28 BOX

*** The next line creates four invisible buttons and ***
*** executes the procedure SHOWCARD when a button is chosen ***

@ 1,1 GET mchoice PICTURE '@*IH ;;;' SIZE 4, 4, 4 ;
  VALID SHOWCARD( ) MESSAGE 'Pick a card, any card!' ;
  COLOR ,,,,R/W
READ CYCLE

PROCEDURE SHOWCARD  && procedure executed when a button is chosen
@ 6,1 CLEAR
DO CASE
  CASE _CUROBJ = 1      && First button chosen
    @ 6,1 SAY 'Hearts'
  CASE _CUROBJ = 2      && Second button chosen
    @ 6,9 SAY 'Diamonds'
  CASE _CUROBJ = 3      && Third button chosen
    @ 6,17 SAY 'Clubs'
  CASE _CUROBJ = 4      && Fourth button chosen
    @ 6,25 SAY 'Spades'
ENDCASE
RETURN .T.
```

@ ... GET — Lists

Purpose	Creates a list
Syntax	<pre> @ <row, column> GET <memvar> <field> FROM <array> [RANGE <expN1> [, <expN2>]] POPUP <popup name> [FUNCTION <expC1>] [PICTURE <expC2>] [FONT <expC3> [, <expN3>]] [STYLE <expC4>] [DEFAULT <expr>] [SIZE <expN4>, <expN5>] [ENABLE DISABLE] [MESSAGE <expC5>] [VALID <expL1> <expN6>] [WHEN <expL2>] [COLOR SCHEME <expN7> COLOR <color pair list>] </pre>
Remarks	FONT and STYLE clauses and RGB color pairs — FoxPro for Windows only
See Also	@ ... SAY, @ ... GET, Controls, _CUROBJ, DEFINE POPUP, OBJNUM(), SHOW GETS, User-Defined Functions

Description This variation of @ ... GET creates a list. A *list* is a set of items from which you can choose one item. To choose an item from a list, select the item and press Enter or double-click on the item.

If you use the Screen Builder to create your data entry screens, you may not have to use this command at all. The Screen Builder automatically generates the commands that create lists.

The list is displayed inside a box, often with a scroll bar to the right. The scroll bar lets you move quickly through the items with the mouse and provides a visual indication of your position in the list. Another quick way to move around in the list is to press the Home key to go to the first item, or the End key to go to the last item. This method works even if the list does not have a scroll bar.

The items in the list are obtained from an array or a popup. Include FROM <array> to build the list from an array. Include POPUP <popup name> to build the list from a popup created with DEFINE POPUP.

Issue READ or READ CYCLE to activate the list.

Clauses **<row, column>**

The upper-left corner of the list is placed at the location designated by <row, column>. The row and column coordinates are numeric expressions that can range from 0 through the maximum number of rows and columns on the desktop (FoxPro for MS-DOS), the main FoxPro window (FoxPro for Windows), or a user-defined window. Rows are numbered from top to bottom, and columns are numbered from left to right.

In FoxPro for Windows the position and size of the list in the main FoxPro window or in a user-defined window are determined by the font of the main FoxPro window or the user-defined window. Most fonts Microsoft Windows uses can be displayed in a wide variety of sizes, and some are proportionally spaced; thus, positions and sizes depend on the current font. A row corresponds to the height of the current font. A column corresponds to the average width of a letter in the current font.

To facilitate precise positioning of the list in the main FoxPro window or in a user-defined window, you can use decimal fractions for row and column coordinates in FoxPro for Windows.

<memvar> | <field>

When you choose an item from a list, a value corresponding to your choice is stored to the memory variable or array element <memvar>, or the field <field>. <memvar> or <field> must be of numeric or character type. If <memvar> or <field> is numeric, the chosen item's position in the list is stored. If <memvar> or <field> is character, the chosen item's prompt is stored.

Initial Option Selection

When a list is displayed, the value of <memvar> or <field> determines which list option (if any) is initially selected. For example, if the value of <memvar> or <field> is 4, the fourth option in the list is selected when the list is activated by READ. If <memvar> or <field> doesn't correspond to any of the options in the list (the value is less than 1 or greater than the number of options), no option is initially selected.

FROM <array>

The FROM <array> clause creates a list from an array. If the array is one-dimensional, the contents of the first array element is the first item in the list, the contents of the second array element is the second item and so on.

If the array is two-dimensional, the elements in the first *column* of the array are used to create the list items. The first element in the first column is the first item in the list, the second element in the first column is the second item and so on.

RANGE <expN1> [, <expN2>]

List items by default start with the contents of the first array element. You can designate a different starting element in the array by including RANGE <expN1>. For example, if the array is one-dimensional and <expN1> is 3, the third element in the array is the first item in the list, the fourth element is the second item, and so on.

An element's position number in a two-dimensional array is determined by counting along rows. For example, suppose you create the following 3-by-3 array:

```
a b c
d e f
g h i
```

Elements a, b, and c are in position numbers 1, 2, and 3. Elements d, e and f are in position numbers 4, 5, and 6, and so on. If a two-dimensional array is used, only elements in the *same column* as array element <expN1> become items in the list. For example, if <expN1> is 2, the contents of elements b, e, and h are the list items. If <expN1> is 5, only the contents of elements e and h are included.

If you include a starting element <expN1>, you can also specify the number of elements in the list by including <expN2>. If <expN2> isn't included, the contents of all array elements from the starting element <expN1> through the last element in the column are items in the list.

If SHOW GETS is issued, the RANGE clause is re-evaluated. If the value of <expN1> or <expN2> has changed, the list is updated to reflect the changes.

The contents of a list can be dynamically changed. You can insert and remove items by modifying the array. The ACOPY(), ADEL(), ADIR(), AELEMENT(), AFIELDS(), AINS(), ALLEN(), ASCAN(), ASORT() and ASUBSCRIPT() functions facilitate the manipulation of arrays.

POPUP <popup name>

The list can also be built from a popup created with `DEFINE POPUP`. Each popup item is used to create an item in the list.

To create a list from a popup, first create the popup with `DEFINE POPUP`. Include in the `POPUP <popup name>` clause the name of the popup created with `DEFINE POPUP`.

You can create a popup (and thus, a list) containing records from a field in a table/.DBF (PROMPT FIELDS), files available on disk (PROMPT FILES) or the names of the fields in a .DBF (PROMPT STRUCTURE).

Example

The following example demonstrates how to create a list from a popup. `DEFINE POPUP` is used to create a popup containing the names of .DBF files available on disk. The .DBF names appear as options in the list. `MARGIN` is included to provide an additional space for the mark character. The `SCROLL` option places a scroll bar to the right of the list.

```
CLEAR
SET TALK OFF
STORE 1 TO mchoice

DEFINE POPUP scrollopts FROM 0, 0 PROMPT FILES LIKE *.DBF ;
    MARGIN SCROLL

@ 2,2 GET mchoice POPUP scrollopts SIZE 8, 20

READ && Activate the list.
```

FUNCTION <expC1> | PICTURE <expC2>

To terminate the `READ` when an item is chosen from the list, include `FUNCTION '&T'` or `PICTURE '@&T'`. Include `FUNCTION '&N'` or `PICTURE '@&N'` to prevent the `READ` from terminating when an item is chosen from the list. For example:

```
... FUNCTION '&T' ...
... PICTURE '@&T' ...
```

If a `FUNCTION` or `PICTURE` clause isn't included, the `READ` isn't terminated when an item is chosen.

FONT <expC3> [, <expN3>] — FoxPro for Windows only

The character expression <expC3> is the name of the font, and the numeric expression <expN3> is the font size. For example, the following clause can be used to display the items in the list in 16 point Roman font:

```
FONT 'ROMAN', 16
```

If you include the FONT clause but omit the font size <expN3>, a 10 point font is used.

If the FONT clause is omitted and the list is placed in the main FoxPro window, the main FoxPro window font is used. If the FONT clause is omitted and the list is placed in a user-defined window, the user-defined window font is used.

If the font you specify is not available, Windows substitutes a font with similar font characteristics.

STYLE <expC4> — FoxPro for Windows only

In FoxPro for Windows, include the STYLE clause to specify a font style for the items in the list. The styles that are available for a font are determined by Windows. If the font style you specify is not available, Windows substitutes a font style with similar characteristics.

The font style is specified with <expC4>. If the STYLE clause is omitted, the standard font style is used.

Character	Font Style
B	Bold
I	Italic
N	Normal
O	Outline
S	Shadow
—	Strikeout
U	Underline

You can include more than one character to specify a combination of font styles. For example, the following clause specifies Bold Italic:

```
STYLE 'BI'
```

DEFAULT <expr>

When you choose an item from the list, your choice is saved in a memory variable, array element or field you specified. If you specify a memory variable that doesn't exist, it is automatically created and initialized if you include the DEFAULT clause. However, an array element isn't created if you specify an array element in a DEFAULT clause. The DEFAULT clause is ignored if the memory variable already exists or you specify a field.



If the DEFAULT clause is omitted and the memory variable <memvar> doesn't exist, the error message "Variable not found" is displayed.

The DEFAULT expression <expr> determines the type of memory variable created and its initial value. <expr> must be of numeric or character type.

SIZE <expN4>, <expN5>

The width of the list is, by default, determined by the width of the longest item text in the list. The number of items in the popup or array by default determines the number of items displayed in the list. You can optionally specify the length and width of the list by including SIZE. The length of the list in rows is specified by <expN4>, and the width of the list in columns is specified by <expN5>.

If there are more items than can be displayed in the list at one time, a scroll bar is automatically placed to the right of the list items.

In FoxPro for Windows, the list's font determines the size of the editing region. The list's font is specified with the FONT clause. If the FONT clause is omitted, the list uses the font of its parent window (the main FoxPro window or a user-defined window).

ENABLE | DISABLE

Lists are by default enabled when READ is issued. You can prevent a list from being activated when READ is issued by including DISABLE. A disabled list cannot be selected and is displayed in disabled colors. Use SHOW GET ENABLED to enable a disabled list.

MESSAGE <expC5>

The MESSAGE clause character expression <expC5> is displayed when a list is selected. In FoxPro for MS-DOS the message is centered on the last line of the desktop and the message location can be changed with SET MESSAGE.

In FoxPro for Windows, the message is placed in the Windows-style status bar. If the Windows-style status bar has been turned off with SET STATUS BAR OFF, the message is placed on the last line of the main FoxPro window.

VALID <expL1> | <expN6>

You can include an optional VALID expression <expL1> or <expN6> that is evaluated when an option is *chosen* from the list. That is, VALID isn't evaluated when you select an item, but when you actually choose an item by selecting it and pressing Enter or double-clicking on the item.

Typically, <expL1> or <expN6> is a user-defined function. With a user-defined function you can select, enable or disable other @ ... GET input fields or objects, open a Browse window, open another data entry screen or move to a new record. CLEAR READ can be included in the user-defined function to terminate the READ.

<expL1>

When a logical value is returned to the VALID clause, the logical value is ignored and the list remains the current control. However, you can specify a UDF that returns a logical value to the VALID clause and then activates another object.

<expN6>

A VALID clause that includes a numeric expression is used to specify which object is activated after an item in the list is chosen. Objects are @ ... GET input fields, check boxes, lists, popups, spinners, text editing regions and each individual button in a set of push, radio and invisible buttons.

The numeric expression <expN6> has one of these effects:

- When <expN6> = 0, the list remains the active control.
- When <expN6> is positive, <expN6> indicates the number of objects to advance. For example, when the list is selected and VALID returns 1, the next object is activated. If <expN6> is greater than the number of objects remaining, the READ is terminated (unless READ CYCLE is issued to activate the objects).
- When <expN6> is negative, <expN6> specifies the number of objects to move back. For example, when you're positioned on a list and VALID returns -1, the previous object is activated. If <expN6> moves back past the first object, the READ is terminated (unless READ CYCLE is issued to activate the objects).

WHEN <expL2>

The WHEN clause allows or prohibits selection of a list based on the logical value of <expL2> which must evaluate to a logical true (.T.) before any of the list can be selected. If <expL2> evaluates to a logical false (.F.), the list cannot be selected and is skipped over if placed between other objects.

COLOR SCHEME <expN7>

! COLOR <color pair list>

If you do not include a COLOR clause, the list's colors are determined by the color scheme for the desktop or the main FoxPro window; if a list is placed in a user-defined window, the window's color scheme determines the list's colors.

The colors of a list can be specified by including the number of an existing color scheme in the COLOR SCHEME clause or a set of color pairs in the COLOR clause.

A color scheme is a set of 10 predefined color pairs. The color pairs in a color scheme can be changed with SET COLOR OF SCHEME. In FoxPro for MS-DOS the color pairs in a color scheme can also be changed in the Color Picker.

A color pair is a set of two letters separated by a forward slash. The first color letter specifies the foreground color and the second letter specifies the background color.

For example, this color pair specifies a red foreground on a white background:

R/W

For a list of colors and their corresponding color letters, see the SET COLOR commands section in this chapter or the Color Table by Color Pair topic in the online help facility.

In FoxPro for Windows, a color pair can also be specified with a set of 6 RGB (Red Green Blue) color values separated by commas. The first 3 color values specify the foreground color and the second 3 color values specify the background color. The color values can range from 0 through 255.

In FoxPro for Windows, the R/W color pair in the example above can also be specified with this RGB color pair:

RGB(255,0,0,255,255,255)

Color Pair Number	List Attribute
1	Disabled option
2	Enabled option
3	Border and scroll bar*
5	Message
6	Selected list item
9	Enabled list
10	Disabled list

* Controls drawn in background color.
Bar drawn in dim of foreground color.

This example uses the array created in the previous example to override the color scheme of the desktop or main FoxPro window with another predefined color scheme:

```
ACTIVATE SCREEN
CLEAR
SET TALK OFF
STORE 1 TO mchoice

DEFINE POPUP scrollopts FROM 0,0 PROMPT FILES LIKE *.DBF ;
    MARGIN SCROLL COLOR SCHEME 4

@ 2,2 GET mchoice POPUP scrollopts SIZE 8, 20
READ && Activate the list
```

The following example, which uses the array created in an example above, defines a list with the following color characteristics:

- A selected list item is shown in bright white on a blue background (W+/B).
- An enabled list is shown with yellow options on a blue background (GR+/B).
- A disabled list is shown with white options on a blue background (W/B).

When you skip a color pair, you must include a comma where the color pair would have been.

Here are the commands:

```
CLEAR
SET TALK OFF
STORE 1 TO mchoice

DEFINE POPUP scrollopts FROM 0,0 PROMPT FILES LIKE *.DBF ;
    MARGIN SCROLL COLOR ,GR+/B,,,W+/B,,,W/B

@ 2,2 GET mchoice POPUP scrollopts SIZE 8, 20
READ && Activate the list
```

Example

```

SET TALK OFF
DEFINE WINDOW example FROM 4,3 TO 21,76 TITLE ' List Example ' ;
    FLOAT SHADOW SYSTEM COLOR SCHEME 8

*** Fill the array cityarray with city data ***

SELECT DISTINCT city FROM customer INTO ARRAY cityarray
SELECT customer

*** Define a popup that contains the structure of customer ***

DEFINE POPUP popstru PROMPT STRUCTURE SCROLL MARGIN MARK CHR(16)

*** Define a popup containing data from the company field ***

DEFINE POPUP popfield PROMPT FIELD company SCROLL MARGIN MARK
CHR(16)
ACTIVATE WINDOW example
@ 1,3 SAY 'Structure Popup:'

*** Get information using predefined popstru popup ***

@ 2,2 GET liststructure POPUP popstru SIZE 11, 20;
    DEFAULT FIELD(1,'customer') WHEN refresh( );
    VALID dispitem(liststructure) COLOR SCHEME 9
@ 13,3 SAY liststructure SIZE 1, 18
@ 1,26 SAY 'Field Popup:'

*** Get information using predefined popfield popup ***

@ 2,25 GET listfield POPUP popfield SIZE 11, 20 ;
    DEFAULT company WHEN refresh( ) VALID dispitem(listfield);
    COLOR SCHEME 9
@ 13,26 SAY listfield SIZE 1, 18
@ 1,50 SAY 'Array of City Names:'

*** Get information using predefined array ***

@ 2,49 GET arrayitem FROM cityarray SIZE 11, 20 ;
    DEFAULT cityarray(1) WHEN refresh( ) VALID dispitem(arrayitem);
    COLOR SCHEME 9
@ 13,50 SAY arrayitem SIZE 1,18
@ 14,63 GET ok FUNCTION '*t \!OK' DEFAULT 1 SIZE 1, 6
READ CYCLE SHOW popshow( )  && Activate gets
RELEASE WINDOW example
RELEASE POPUPS popstru, popfield

```

```
FUNCTION refresh
```

```
*** Refresh window information without calling subroutine ***
```

```
SHOW GETS OFF
```

```
FUNCTION dispitem
```

```
*** Display which item has been chosen ***
```

```
PARAMETER item
```

```
WAIT WINDOW 'You have chosen: ' + ALLTRIM(item) NOWAIT
```

```
FUNCTION popshow
```

```
*** Display current items for each list ***
```

```
@ 13,3 SAY liststructure SIZE 1, 18
```

```
@ 13,26 SAY listfield SIZE 1, 18
```

```
@ 13,50 SAY arrayitem SIZE 1, 18
```

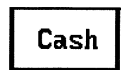
@ ... GET — Popups

Purpose	Creates a popup
Syntax	<pre> @ <row, column> GET <memvar> <field> FUNCTION <expC1> PICTURE <expC2> [FONT <expC3> [, <expN1>]] [STYLE <expC4>] [DEFAULT <expr>] [FROM <array>] [RANGE <expN2> [, <expN3>]] [SIZE <expN4>, <expN5>] [ENABLE DISABLE] [MESSAGE <expC5>] [VALID <expL1> <expN6>] [WHEN <expL2>] [COLOR SCHEME <expN7> [, <expN8>] COLOR <color pair list>] </pre>
Remarks	FONT and STYLE clauses and support for RGB color pairs — FoxPro for Windows only
See Also	@ ... SAY, @ ... GET, Controls, _CUROBJ, READ, SET KEYCOMP, SHOW GET, OBJNUM(), User-Defined Functions

Description This variation of @ ... GET creates a popup. When selected, the popup opens, displaying a set of options from which you can choose one. The set of options is specified in the FUNCTION and/or PICTURE clause containing the specification code for a popup — a caret (^).

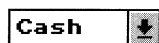
If you use the Screen Builder to create your data entry screens, you may not have to use this command at all. The Screen Builder automatically generates the commands that create popups.

A popup looks like this in FoxPro for MS-DOS:



To open a popup in FoxPro for MS-DOS, press Enter or the Spacebar or click on the popup. You can open a popup in the same manner in FoxPro for Windows if KEYCOMP is set to DOS.

A popup looks like this in FoxPro for Windows:



To open a popup in FoxPro for Windows when KEYCOMP is set to WINDOWS (the default setting), press the Spacebar, Alt+Up Arrow, Alt+Down Arrow or click on the popup.

Clauses **<row, column>**

The upper-left corner of the popup is placed at the location designated by <row, column>. The row and column coordinates are numeric expressions. Row and column values can range from 0 through the maximum number of rows and columns on the desktop (FoxPro for MS-DOS), the main FoxPro window (FoxPro for Windows), or a user-defined window. Rows are numbered from top to bottom, and columns are numbered from left to right.

In FoxPro for Windows the position and size of a popup in the main FoxPro window or in a user-defined window are determined by the font of the main FoxPro window or the user-defined window. Most fonts Microsoft Windows uses can be displayed in a wide variety of sizes, and some are proportionally spaced; thus, positions and sizes depend on the current font. A row corresponds to the height of the current font. A column corresponds to the average width of a letter in the current font.

To facilitate precise positioning of a popup in the main FoxPro window or in a user-defined window, you can use decimal fractions for row and column coordinates in FoxPro for Windows.

<memvar> | <field>

When you choose an option from the popup, your choice is stored to the memory variable or array element specified with <memvar> or a field specified with <field>. If you press Escape to exit the popup, the value of <memvar> or <field> isn't changed.

<memvar> or <field> must be of numeric or character type. If <memvar> or <field> is character, the prompt of the item you choose is stored to <memvar>. If <memvar> or <field> is numeric, the number representing the position of the item in the popup is stored to <memvar> or <field>.

Initial Option Selection

When a popup is displayed, the value of `<memvar>` or `<field>` determines which popup option (if any) is initially displayed on the face of the popup.

If `<memvar>` or `<field>` is numeric, the option in the specified numeric position in the list of options is initially displayed on the face of the popup. For example, if the memory variable, array element or field contains 1, the first popup option you specify is initially displayed. If the value of `<memvar>` or `<field>` doesn't correspond to any of the popup options (the value is less than 1 or greater than the number of options) the face of the popup is initially blank.

If `<memvar>` or `<field>` is of character type, a case-sensitive comparison is made between `<memvar>` or `<field>` and each popup option. When the comparison is made, all special characters and any leading or trailing spaces are ignored. If a match is found, the matching option is initially displayed on the face of the popup. If a match isn't found, the character value of `<memvar>` or `<field>` is initially displayed on the face of the popup and is added as a temporary option at the end of the list of options.

FUNCTION `<expC1>` | PICTURE `<expC2>`

When creating popups, you must include the FUNCTION clause, the PICTURE clause or both. There is no advantage to any of the three methods. The FUNCTION or PICTURE clause contains the popup specification code `^` and the set of popup options.

Options can also be specified in the FROM `<array>` clause. If the options are specified there, you do not need to include them in the FUNCTION or PICTURE clause.

However, if you include FROM `<array>`, `^` must still be included in the FUNCTION or PICTURE clause.

The FUNCTION character expression `<expC1>` must begin with `^`. To create the options that appear in the popup list, include a space after `^` followed by a list of options separated by semicolons. The following example creates the option prompts Cash, Charge, Net 30 and Net 60:

```
... FUNCTION '^ Cash;Charge;Net 30;Net 60' ...
```

The PICTURE character expression `<expC2>` uses the same syntax as the FUNCTION character expression except the PICTURE expression must begin with `@` followed by `^`.

For example, the following PICTURE clause creates the option prompts Cash, Charge, Net 30 and Net 60:

```
... PICTURE '@^ Cash;Charge;Net 30;Net 60' ...
```

You can also include *both* the FUNCTION and PICTURE clauses to create popups. If both are included, the FUNCTION character expression <expC1> must contain ^ to create the popup, and can also include a space and the popup options. The PICTURE character expression <expC2> can include prompts to create additional options.

The following examples illustrate the various forms of syntax you can use to create a popup. In all the examples, a popup is placed in the second row and column. It contains four options with the prompts Cash, Charge, Net 30 and Net 60. It is initialized by the memory variable MCHOICE. When you choose one of the options from the popup, your choice is stored to MCHOICE.

FUNCTION clause only:

```
STORE 1 TO mchoice
@ 2,2 GET mchoice FUNCTION '^ Cash;Charge;Net 30;Net 60'
READ
```

```
STORE 1 TO mchoice
STORE '^ Cash;Charge;Net 30;Net 60' TO mfunc
@ 2,2 GET mchoice FUNCTION mfunc
READ
```

PICTURE clause only:

```
STORE 1 TO mchoice
@ 2,2 GET mchoice PICTURE '@^ Cash;Charge;Net 30;Net 60'
READ

STORE 1 TO mchoice
@ 2,2 GET mchoice PICTURE '@^' + ' Cash;Charge;Net 30;Net 60'
READ
```

FUNCTION and PICTURE clauses:

```
STORE 1 TO mchoice
@ 2,2 GET mchoice FUNCTION '^' ;
    PICTURE ' Cash;Charge;Net 30;Net 60'
READ

STORE 1 TO mchoice
@ 2,2 GET mchoice FUNCTION '^ Cash;Charge' ;
    PICTURE ' ;Net 30;Net 60'
READ
```

PICTURE and FUNCTION Options N and T

Place one of the following two options after the caret ^ in the FUNCTION or PICTURE clause to specify whether or not the READ is terminated when an option is chosen from the popup.

Option	Description
N	Do not terminate the READ when an option is chosen. This is the default behavior.
T	Terminate the READ when an option is chosen.

Popup Options with Special Features

When defining a prompt, you can assign special characteristics to a popup option. For example, you can assign a hot key to the option or disable the option by including special characters.

Hot Keys

In FoxPro for MS-DOS, a hot key is a highlighted letter in a prompt that you can type to immediately choose an option on an open popup. To assign a hot key, place a backslash and a less than sign (\<) before the desired character of the popup option.

In FoxPro for Windows, a hot key is an underlined letter on a prompt that you can type to immediately choose an option on the popup. The popup doesn't have to be open to choose an option with a hot key if KEYCOMP is set to WINDOWS.

The following example assigns the hot keys C and R to the **Cash** and **Charge** options, respectively:

```
STORE 1 TO mchoice
@ 2,2 GET mchoice FUNCTION '^ \<Cash;Cha\<rge;Net 30;Net 60'
READ
```

Disabled Options

You can disable an option so it can't be chosen. Disabled options are shown in disabled colors. To disable a popup option, place a backslash (\) before the option. To disable the entire popup, include the DISABLE key word.

The **Charge** option is disabled in this example:

```
STORE 1 TO mchoice
@ 2,2 GET mchoice FUNCTION '^ Cash;\Charge;Net 30;Net 60'
READ
```

FONT <expC3> [, <expN1>] — FoxPro for Windows only

The character expression <expC3> is the name of the font, and the numeric expression <expN1> is the font size. For example, the following clause can be used to display the options in the popup in 16 point Roman font:

```
FONT 'ROMAN', 16
```

If you include the FONT clause but omit the font size <expN1>, a 10 point font is used.

If the FONT clause is omitted and the popup is placed in the main FoxPro window, the main FoxPro window font is used. If the FONT clause is omitted and the popup is placed in a user-defined window, the user-defined window font is used.

If the font you specify is not available, Windows substitutes a font with similar font characteristics.

STYLE <expC4> — FoxPro for Windows only

In FoxPro for Windows, include the STYLE clause to specify a font style for the options in the popup. The styles that are available for a font are determined by Windows. If the font style you specify is not available, Windows substitutes a font style with similar characteristics.

The font style is specified with <expC4>. If the STYLE clause is omitted, the standard font style is used.

Character	Font Style
B	Bold
I	Italic
N	Normal
O	Outline
S	Shadow
—	Strikeout
U	Underline

You can include more than one character to specify a combination of font styles. For example, the following specifies Bold Italic:

```
STYLE 'BI'
```

DEFAULT <expr>

When you choose an option from a popup, your choice is saved in the memory variable, array element or field you specify. If you specify a memory variable that doesn't exist, it is automatically created and initialized if you include the DEFAULT clause. However, an array element isn't created if you specify an array element in a DEFAULT clause. The DEFAULT clause is ignored if the memory variable already exists or you specify a field.



If the DEFAULT clause isn't included and the memory variable <memvar> doesn't exist, the error message "Variable not found" is displayed.

The DEFAULT expression <expr> determines the type of memory variable created and its initial value. <expr> must be a numeric or character expression. Here are examples of DEFAULT clauses for popups:

```
@ 2,2 GET mchoice FUNCTION '^ Cash;Charge;Net 30;Net 60' ;
    DEFAULT 'Cash'
READ
```

```
@ 2,2 GET mchoice FUNCTION '^ Cash;Charge;Net 30;Net 60' DEFAULT 3
READ
```

FROM <array>

The FROM <array> clause can be included to create the popup options from a predefined array. If you create popup options with FROM <array>, you must still specify @^ in the PICTURE clause or ^ in the FUNCTION clause. For example:

```
... FROM marray PICTURE '@^' ...
... FROM marray FUNCTION '^' ...
```

If FROM <array> is included, any option prompts specified in the PICTURE or FUNCTION clause are ignored.

The options that appear in the popup are taken from the elements of the array you specify with <array>. If the array is one-dimensional, the first element in the array creates the first option in the popup, the second array element creates the second option and so on.

If the array is two-dimensional, only the elements in the first column of the array create the options in the popup. The first element in the array creates the first option in the popup, the second element in the first column creates the second option and so on.

RANGE <expN2> [, <expN3>]

By default, popup options start with the first array element. You can designate a different starting element in the array by including the RANGE value <expN2>. For example, if the array is one-dimensional and <expN2> is 3, the third element in the array creates the first option in the popup, the fourth element creates the second option and so on.

An element's position number in a two-dimensional array is determined by counting along rows. For example, suppose you create the following 3-by-3 array:

```
a b c  
d e f  
g h i
```

Elements a, b, and c are in position numbers 1, 2, and 3. Elements d, e and f are in position numbers 4, 5, and 6 and so on. With two-dimensional arrays, only elements in the *same column* as array element <expN2> create the options on the popup. For example, if <expN2> is 2, the contents of elements b, e, and h are used to create the popup options. If <expN2> is 5, only the contents of elements e and h are used.

If you include a starting element <expN2>, you can also specify the number of elements used to create options by including the numeric expression <expN3>. The number of options appearing in the popup is determined by <expN3>. If <expN3> isn't included, the contents of all array elements in the specified column from the starting element <expN2> through the last element in the column are the popup options.

If the array is two-dimensional, <expN3> designates the number of elements used from the array column containing the starting element <expN2>. For example, if <expN2> is 2 and <expN3> is 3, the second array element and the next two elements in the *same array column* create options on the popup.

The contents of a popup can be dynamically changed. You can insert and delete options by modifying the array. When SHOW GETS is issued, the RANGE clause is re-evaluated. If the value of the element numbered <expN2> or <expN3> has changed, the popup is updated to reflect the change.

The ACOPY(), ADEL(), ADIR(), AELEMENT(), AFIELDS(), AINS(), ALLEN(), ASCAN(), ASORT() and ASUBSCRIPT() functions facilitate the manipulation of arrays.

Example The following program example demonstrates how a popup can be dynamically changed. Two radio buttons, **Vegetable** and **Fruits**, can be chosen. Another set of radio buttons specify the color (red, yellow or green) of the vegetables or fruits. The popup is modified and refreshed as different buttons are chosen.

```

CLEAR
SET TALK OFF
STORE 1 TO color, fruitorveg, popchoice, start
DIMENSION poparray(4,3)
STORE 'Apples'      TO poparray(1,1)
STORE 'Bananas'     TO poparray(1,2)
STORE 'Limes'       TO poparray(1,3)
STORE 'Strawberries' TO poparray(2,1)
STORE 'Lemons'     TO poparray(2,2)
STORE 'Grapes'      TO poparray(2,3)
STORE 'Radishes'    TO poparray(3,1)
STORE 'Corn'        TO poparray(3,2)
STORE 'Lettuce'     TO poparray(3,3)
STORE 'Beets'       TO poparray(4,1)
STORE 'Squash'      TO poparray(4,2)
STORE 'Celery'      TO poparray(4,3)
@ 9,2 SAY 'Color:'
@ 11,2 SAY 'Type:'
@ 9,10 GET color FUNCTION '*RH \<Red;\<Yellow;\<Green' ;
      SIZE 1, 9, 1 VALID PROC1( )
@ 11,10 GET fruitorveg FUNCTION '*RH \<Fruit;\<Vegetable' ;
      SIZE 1, 12, 1 VALID PROC2( )
@ 4,14 GET popchoice FUNCTION '^' FROM poparray RANGE start,2
@ 14,17 GET ok FUNCTION '*T OK' DEFAULT 1 SIZE 1, 8
@ 10,0,10,41 BOX
@ 8,0,12,41 BOX
READ CYCLE

PROCEDURE PROC1
DO CASE
CASE COLOR = 1
  IF fruitorveg = 1
    start = 1
  ELSE
    start = 7
  ENDIF
ENDIF

```

```
CASE COLOR = 2
  IF fruitorveg = 1
    start = 2
  ELSE
    start = 8
  ENDIF
CASE COLOR = 3
  IF fruitorveg = 1
    start = 3
  ELSE
    start = 9
  ENDIF
ENDCASE
STORE 1 TO popchoice
SHOW GETS
RETURN .T.

PROCEDURE PROC2
IF fruitorveg = 1
  DO CASE
    CASE color = 1
      STORE 1 TO start
    CASE color = 2
      STORE 2 TO start
    CASE color = 3
      STORE 3 TO start
  ENDCASE
ELSE
  DO CASE
    CASE color = 1
      STORE 7 TO start
    CASE color = 2
      STORE 8 TO start
    CASE color = 3
      STORE 9 TO start
  ENDCASE
ENDIF
STORE 1 TO popchoice
SHOW GETS
RETURN .T.
```


SIZE <expN4> [, <expN5>]

By default, the width of the popup is determined by the length of the longest option text in columns. You can optionally specify the width of the popup in columns by specifying <expN5>. For popups, the first SIZE argument <expN4> is ignored, because the height of a popup is determined by the number of options. You can include any numeric value for <expN4>.

In FoxPro for Windows, the popup font determines the size of the popup. The popup font is specified with the FONT clause. If the FONT clause is omitted, the popup uses the font of its parent window (the main FoxPro window or a user-defined window).

ENABLE | DISABLE

Popups are enabled by default when READ or READ CYCLE is issued. You can prevent a popup from being activated when READ or READ CYCLE is issued by including DISABLE. A disabled popup cannot be selected and is displayed in disabled colors. To disable individual popup options instead of the entire popup, see Disabled Options earlier in this section. Use SHOW GET ENABLED to enable a disabled popup.

MESSAGE <expC5>

The MESSAGE clause character expression <expC5> is displayed when the popup is selected. In FoxPro for MS-DOS the message is centered by default on the last line of the desktop. The message location can be changed with SET MESSAGE.

In FoxPro for Windows, the message is placed in the Windows-style status bar. If the Windows-style status bar has been turned off with SET STATUS BAR OFF, the message is placed on the last line of the main FoxPro window.

VALID <expL1> | <expN6>

You can include an optional VALID expression <expL1> or <expN6> that is evaluated when a popup option is *chosen*. That is, VALID isn't evaluated when you select the popup, but when you choose an option in the popup.

Typically, <expL1> or <expN6> is a user-defined function. With a user-defined function you can select, enable or disable other objects; open a Browse window, open another data entry screen or move to a new record. CLEAR READ may be included in the user-defined function to terminate the READ.

<expL1>

When a logical value is returned to the VALID clause, the logical value is ignored and the popup remains the current control. However, you can specify a UDF that returns a logical value to the VALID clause and activates another object.

<expN6>

A VALID clause that includes a numeric expression is used to specify which object is activated after an option in the popup is chosen. Objects are @ ... GET input fields, check boxes, lists, popups, spinners, text editing regions and each individual button in a set of push, radio and invisible buttons.

The expression <expN6> has one of these effects:

- When <expN6> = 0, the popup remains the active control.
- When <expN6> is positive, <expN6> specifies the number of objects to advance. For example, when the popup is the current object and VALID returns 1, the next object is activated. If <expN6> is greater than the number of objects remaining, the READ is terminated (unless READ CYCLE is issued to activate the objects).
- When <expN6> is negative, <expN6> specifies the number of objects to move back. For example, if the popup is the current object and VALID returns -1, the previous object is activated. If <expN6> moves back past the first object, the READ is terminated (unless READ CYCLE is issued to activate the objects).

WHEN <expL2>

The WHEN clause allows or prohibits selection of a popup based on the logical value of <expL2>, which must evaluate to a logical true (.T.) before a popup can be selected. If <expL2> evaluates to a logical false (.F.), the popup cannot be selected and is skipped over if placed between other objects.

COLOR SCHEME <expN7> [, <expN8>]

COLOR <color pair list>

If you do not include a COLOR clause, popup colors are determined by the color scheme for the desktop or the main FoxPro window; if a popup is placed in a user-defined window, the window's color scheme determines the popup's colors.

The colors of a popup can be specified by including the number of an existing color scheme in the COLOR SCHEME clause or a set of color pairs in the COLOR clause.

A color scheme is a set of 10 predefined color pairs. The color pairs in a color scheme can be changed with SET COLOR OF SCHEME. In FoxPro for MS-DOS the color pairs in a color scheme can also be changed in the Color Picker.

A color pair is a set of two letters separated by a forward slash. The first color letter specifies the foreground color and the second letter specifies the background color.

For example, this color pair specifies a red foreground on a white background:

R/W

For a list of colors and their corresponding color letters, see the SET COLOR commands section in this chapter or the Color Table by Color Pair topic in the online help facility.

In FoxPro for Windows, a color pair can also be specified with a set of 6 RGB (Red Green Blue) color values separated by commas. The first 3 color values specify the foreground color and the second 3 color values specify the background color. The color values can range from 0 through 255.

In FoxPro for Windows, the R/W color pair in the example above can also be specified with this RGB color pair:

RGB(255,0,0,255,255,255)

By default, the colors of a popup and its options are derived from the color scheme of the desktop, main FoxPro window or the active user-defined window and color scheme 2.

The first color scheme <expN7> determines the colors of the face of the popup when the popup is selected, enabled or disabled. It also determines the color of the message. The second color scheme <expN8> determines the color of the popup options and the border of box containing the options.

The following table lists the color pairs in each color scheme and what each color pair controls in a popup.

Popup Attribute		
Color Pair Number	Color Scheme <expN7> or Output window	Color Scheme <expN8> or Color Scheme 2
1	Not used	Disabled options
2	Not used	Enabled options
3	Not used	Border
5	Message	Not used
6	Selected popup	Selected option
7	Not used	Hot keys
9	Enabled popups	Not used
10	Disabled popups	Not used

A color pair list only control colors of the face of the popup when the popup is selected, enabled or disabled and the color of the message (popup color attributes controlled by color scheme <expN7>). For complete color control of a popup, specify two color schemes.

Example

This example shows how to override the current color scheme with other predefined color schemes:

```
STORE 1 TO mchoice
@ 2,2 GET mchoice FUNCTION '^ Cash;Charge;Net 30;Net 60' ;
  COLOR SCHEME 4, 5
READ
```

The following example uses a popup to get a field for the browse order from the CUSTOMER table/.DBF:

```

SET TALK OFF
DEFINE WINDOW popupex FROM 5,10 TO 15,55 ;
    FLOAT DOUBLE COLOR SCHEME 5
CLOSE DATABASES
USE customer
DIMENSION dbftags(256)
dbftags(1) = 'Record #'
FOR i = 2 to 256
    IF empty(tag(i-1))
        i = i - 1
        DIMENSION dbftags(i)
        EXIT
    ELSE
        dbftags(i) = tag(i-1)
    ENDIF
ENDFOR
dbforder = 'Record #'

ACTIVATE WINDOW popupex
@ 3,7 GET dbforder PICTURE '@^' FROM dbftags SIZE 3, 14
@ 0,2 TO 7,43
@ 0,5 SAY ' Customer Browse Order: '
@ 2,27 GET okcancel PICTURE '@*VT \!OK;\?Cancel' ;
    SIZE 1, 10, 2 DEFAULT 0
READ CYCLE
RELEASE WINDOW popupex

IF okcancel = 1
    IF dbforder = "Record #"
        SET ORDER TO
    ELSE
        SET ORDER TO (dbforder)
    ENDIF
    WAIT WINDOW 'Order set to '+dbforder NOWAIT
    BROWSE WIDTH 10 NOWAIT
ELSE
    WAIT WINDOW 'Order not set' NOWAIT
ENDIF

```

@ ... GET — Push Buttons

Purpose	Creates a set of push buttons or picture push buttons
Syntax	<pre> @ <row, column> GET <memvar> <field> FUNCTION <expC1> PICTURE <expC2> [FONT <expC3> [, <expN1>]] [STYLE <expC4>] [DEFAULT <expr>] [SIZE <expN2>, <expN3> [, <expN4>]] [ENABLE DISABLE] [MESSAGE <expC5>] [VALID <expL1> <expN5>] [WHEN <expL2>] [COLOR SCHEME <expN6> COLOR <color pair list>] </pre>
Remarks	FONT and STYLE clauses and support for RGB color pairs — FoxPro for Windows only
See Also	@ ... SAY, @ ... GET, Controls, _CUROBJ, OBJNUM(), READ, SET KEYCOMP, SHOW GETS, User-Defined Functions

Description This variation of @ ... GET creates push buttons or picture push buttons. A single button or a group of buttons can be created. In FoxPro for MS-DOS, a push button is shown as a string of text between right and left angle brackets. For example:

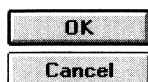
```

< OK >
< Cancel >

```

If you use the Screen Builder to create your data entry screens, you may not have to use this command at all. The Screen Builder automatically generates the commands that create push buttons and picture push buttons.

In FoxPro for Windows, a push button is shown as a string of text surrounded by a rectangle:



The string of text, often called a *prompt*, is specified in the FUNCTION or PICTURE clause. Typically, a push button is used to trigger an action. The triggered action is specified in a VALID clause. A push button is activated by issuing READ or READ CYCLE.

You can create picture push buttons in FoxPro for Windows. .BMPs (bitmap pictures) on buttons replace the push button prompts.

Clauses

<row, column>

The first button in a set of push buttons is placed at the location designated by <row, column>. The row and column coordinates are numeric expressions. Row and column values can range from 0 through the maximum number of rows and columns on the desktop (FoxPro for MS-DOS), the main FoxPro window (FoxPro for Windows), or a user-defined window. Rows are numbered from top to bottom, and columns are numbered from left to right.

In FoxPro for Windows the position and size of push buttons in the main FoxPro window or in a user-defined window are determined by the font of the main FoxPro window or the user-defined window. Most fonts Microsoft Windows uses can be displayed in a wide variety of sizes, and some are proportionally spaced; thus, positions and sizes depend on the current font. A row corresponds to the height of the current font. A column corresponds to the average width of a letter in the current font.

To facilitate precise positioning of a push buttons in the main FoxPro window or in a user-defined window, you can use decimal fractions for row and column coordinates in FoxPro for Windows.

<memvar> | <field>

When you choose a push button, your choice is stored to the memory variable or array element <memvar> or to the field <field>. <memvar> or <field> must be of numeric or character type.

If <memvar> or <field> is of numeric type, a number corresponding to the push button choice is stored to the memory variable, array element or field specified with <memvar> or <field>. For example, if you create four push buttons and choose the third button, 3 is stored. If <memvar> or <field> is of character type, the prompt of the push button you choose is stored to the memory variable, array element or field specified with <memvar> or <field>.

FUNCTION <expC1> | PICTURE <expC2>

When creating push buttons you must include the FUNCTION clause, the PICTURE clause or both. There is no advantage to any of the three methods. The FUNCTION or PICTURE clause contains the push button specification code an asterisk * and the text for the individual push button prompts.

The FUNCTION character expression <expC1> must begin with *. To create the push button prompts, include a space after the asterisk followed by a list of the prompts separated by semicolons. One button is created for each prompt. For example, the following FUNCTION clause creates push buttons with the prompts **OK** and **Cancel**:

```
... FUNCTION '* OK;Cancel' SIZE 1, 8 ...
```

The PICTURE character expression <expC2> uses the same syntax as the FUNCTION character expression except the PICTURE character expression must begin with @ followed by an asterisk (*). For example, the following PICTURE clause creates push buttons with the prompts **OK** and **Cancel**:

```
... PICTURE '@* OK;Cancel' SIZE 1, 8 ...
```

You can also include *both* the FUNCTION and PICTURE clauses to create push buttons. If both are included, the FUNCTION character expression <expC1> must contain * to create the push buttons followed by a space and the push button prompts. The PICTURE character expression <expC2> can include prompts to create additional push buttons.

The following examples illustrate the various forms of syntax you can use to create push buttons. Two push buttons are created: **OK** and **Cancel**. The first button (**OK**) is placed in the second row and second column. A number corresponding to the button chosen is stored to the memory variable MCHOICE.

FUNCTION clause only:

```
STORE 1 TO mchoice
@ 2,2 GET mchoice FUNCTION '* OK;Cancel' SIZE 1, 8
READ
```

```
STORE 1 TO mchoice
STORE '* OK;Cancel' TO mprompts
@ 2,2 GET mchoice FUNCTION mprompts SIZE 1, 8
READ
```


PICTURE clause only:

```
STORE 1 TO mchoice
@ 2,2 GET mchoice PICTURE '@* OK;Cancel' SIZE 1, 8
READ

STORE 1 TO mchoice
@ 2,2 GET mchoice PICTURE '@*' + ' OK;Cancel' SIZE 1, 8
READ
```

FUNCTION and PICTURE clauses:

```
STORE 1 TO mchoice
@ 2,2 GET mchoice FUNCTION '*' ;
    PICTURE ' OK;Cancel' SIZE 1, 8
READ
```

Push Buttons with Picture Prompts

In FoxPro for Windows, the prompt for a push button can also be the name of a bitmap picture file with a .BMP extension. When a push button uses a .BMP for a prompt, the push button mimics the *behavior* of a push button with a text prompt. For example, the .BMP is displayed on a button but the push button isn't displayed. Like choosing a push button, choosing the picture prompt stores the push button's value to <memvar> or <field>.

To use a .BMP in a push button, add B to the push button specification code. The FUNCTION and PICTURE clause specification codes that create a push button are * and @*, respectively. To create a push button with picture prompts, use the codes *B and @*B.



The pictures aren't clipped, reduced or enlarged to fit the push button. Use the SIZE clause to adjust the size of the push buttons to accommodate the pictures.

Picture Masks and Push Buttons

In FoxPro for Windows, a picture push button has three states: up, down and disabled. FoxPro automatically controls the appearance a picture push button when it is in each of these three states, but you can override the default appearance by using a picture mask. A mask is a monochrome .BMP file with an .MSK extension. FoxPro automatically looks for a mask for a .BMP file in the same directory where the .BMP file is located. The mask must have the same base name as the .BMP file and an extension of .MSK.

A mask is used to control the transparent areas of a picture push button. By default, the white areas are transparent. If a mask is present, the white areas of the mask, not the .BMP, are transparent.

In most cases, a mask isn't necessary. If you don't need anything in your picture to appear white or your .BMP has a white background, the picture push button will appear as desired in the up, down and disabled states.

When a picture push button has a mask and is in the up or down state, any white areas in the .BMP appear transparent, allowing the color of the button face to show through. However, you can maintain the white color of certain areas. Suppose that you have a .BMP of a dog on a white background and white eyes, and the button face is red. You want the background of the button to appear red but you want the dog's eyes to be white, not red. Make a mask that is the same size as the .BMP but includes only two colors, black and white. Leave the background of the mask white but make the dog — including his eyes — completely black. When the button is displayed, the background is red to match the button face but the dog's eyes are white.

When a picture push button is disabled, any white areas in the .BMP appear transparent so the color of the button face shows through. Any non-white areas appear dark grey. If the button has a .MSK, all white areas in the mask are transparent so the color of the button face show through, and all black areas appear dark grey.

PICTURE and FUNCTION Options N, T, H and V

Additional options can be combined with the * specification code to modify the behavior (N and T options) and appearance (H and V options) of push buttons.

Option	Description
N	Do not terminate the READ when a push button is chosen.
T	Terminate the READ when a push button is chosen. This is the default behavior.
H	Position the push buttons in a horizontal row.
V	Position the push buttons in a vertical row. This is the default position.

You can combine the H and V options with the T and N options in the following ways: NH, NV, TH and TV. For example, the following clause creates a horizontal row of buttons and doesn't cause the READ to terminate when one of the buttons is chosen:

```
... FUNCTION '*NH ...'
```

Push Buttons with Special Features

You can assign a hot key to a button, disable a button, or create a default or escape button. To assign these special features to a push button, you must include special characters when defining the prompt. The special characters are removed when the prompt is stored to <memvar> or <field>.

Hot Keys

In FoxPro for MS-DOS, a hot key is a highlighted letter in the push button prompt that you can type to immediately choose a push button. Pressing the hot key selects the button and chooses it. To assign a hot key, place a backslash and a less than sign (\<) before the desired character of the push button prompt.

A hot key doesn't choose the push button if the current object is a GET field, a text editing region, a popup or a list.

If the current object is an @ ... GET input field or a text editing region, pressing the hot key enters the character in the field or the text editing region. If the current object is a popup or a list, pressing the hot key selects the first option on the popup or list whose prompt begins with the hot key character.

In FoxPro for Windows, a hot key is an underlined letter in the push button prompt that you can type to immediately choose the button. If the current object is an @ ... GET input field, a text editing region, a popup or a list and KEYCOMP is set to WINDOWS, you can press the Alt key and the hot key to choose the push button.

The following example creates the **OK** and **Cancel** buttons with hot keys O and C, respectively:

```
STORE 1 TO mchoice
@ 2,8 GET mchoice FUNCTION '* \<OK;\<Cancel'
READ
```

Disabled Push Buttons

You can disable a push button so it can't be selected or chosen. Disabled push buttons are shown in disabled colors. To disable a single push button, place two backslashes (\\) before the button's prompt. To disable a set of push buttons, include the DISABLE key word.

The **OK** push button is disabled in the following example:

```
STORE 1 TO mchoice
@ 2,2 GET mchoice FUNCTION '* \\OK;Cancel'
READ
```

Default Push Buttons

A default push button is typically used to exit a data entry screen, dialog or routine and accept any changes made in the screen, dialog or routine. To create a default push button, place a backslash and an exclamation point (\!) before the push button's prompt.

The following example specifies the **OK** button as the default push button:

```
STORE 1 TO mchoice
@ 2,2 GET mchoice FUNCTION '* \!OK;Cancel'
READ
```

In FoxPro for MS-DOS, the default push button is enclosed in double angle brackets to distinguish it from other push buttons. The default push button is automatically chosen when you press Ctrl+Enter or Ctrl+W. You can specify only one default push button for each READ.

In FoxPro for Windows, the default push button is surrounded by a thick border. Press Enter, Ctrl+Enter or Ctrl+W to choose the push default button.

The KEYCOMP setting determines the behavior of default push buttons in FoxPro for Windows in the following ways:

- If you set KEYCOMP DOS, the behavior of the default push button is the same as the behavior of a default push button in FoxPro for MS-DOS.
- If you set KEYCOMP to WINDOWS, the default push button changes as you move between buttons. The push button specified with \! is the default button when objects besides push buttons are active. When a push button is active, it becomes the default push button.

Escape Push Buttons

An escape push button is automatically chosen when you press the Escape key. An escape button is typically used to exit a data entry screen, dialog or routine and discard any changes made in the screen, dialog or routine. You can specify only one escape push button for each READ.

In the absence of an escape push button, pressing Escape when editing an @ ... GET input field aborts the editing and restores the original value to the field. If the field has a VALID procedure, the procedure isn't executed. When an escape push button is present, pressing Escape saves the contents of the current @ ... GET input field contents and executes the VALID procedure.

To create an escape push button, place a backslash and a question mark (\?) before the push button's prompt.

The following example specifies the **Cancel** push button as the escape push button:

```
STORE 1 TO mchoice
@ 2,2 GET mchoice FUNCTION '* OK;\?Cancel'
READ
```

Push Buttons with Combined Characteristics

You can create a push button with more than one special feature. In the following example, the **OK** button is the default button and is assigned O as its hot key character. The **Cancel** button is the escape button and is assigned C as its hot key character. The hot key designators immediately precede the desired hot key character.

```
STORE 1 TO mchoice
@ 2, 2 GET mchoice FUNCTION '*  \!\<OK;\?\<Cancel'
READ
```

FONT <expC3> [, <expN1>] — FoxPro for Windows only

The character expression <expC3> is the name of the font, and the numeric expression <expN1> is the font size. For example, the following clause can be used to display the push button prompts in 16 point Roman font:

```
FONT 'ROMAN', 16
```

If you include the FONT clause but omit the font size <expN1>, a 10 point font is used.

If the FONT clause is omitted and the push buttons are placed in the main FoxPro window, the main FoxPro window font is used. If the FONT clause is omitted and the push buttons are placed in a user-defined window, the user-defined window font is used.

If the font you specify is not available, Windows substitutes a font with similar font characteristics.

STYLE <expC4> — FoxPro for Windows only

In FoxPro for Windows, include the STYLE clause to specify a font style for the push button prompts. The styles that are available for a font are determined by Windows. If the font style you specify is not available, Windows substitutes a font style with similar characteristics.

The font style is specified with <expC4>. If the STYLE clause is omitted, the standard font style is used.

Character	Font Style
B	Bold
I	Italic
N	Normal
O	Outline
S	Shadow
—	Strikeout
U	Underline

You can include more than one character to specify a combination of font styles. For example, the following clause specifies Bold Italic:

```
STYLE 'BI'
```

DEFAULT <expr>

When you choose a push button, your choice is saved in the memory variable, array element or field you specify. If you specify a memory variable that doesn't exist, it is automatically created and initialized if you include DEFAULT. However, an array element isn't created if you specify an array element in a DEFAULT clause. The DEFAULT clause is ignored if the memory variable already exists or you specify a field.



If the DEFAULT clause isn't included and the memory variable you specify doesn't exist, the error message "Variable not found" is displayed.

The DEFAULT expression <expr> determines the type of memory variable created and its initial value. <expr> must be of numeric or character type. The following are examples of DEFAULT clauses for push buttons:

```
@ 2,2 GET mchoice FUNCTION '* OK;Cancel' DEFAULT 'OK'
READ
```

```
@ 2,2 GET mchoice FUNCTION '* OK;Cancel' DEFAULT 2
READ
```

```
STORE 'OK' TO button
```

```
@ 2,2 GET mchoice FUNCTION '* OK;CANCEL' DEFAULT button
READ
```

SIZE <expN2>, <expN3> [, <expN4>]

The numeric expression <expN2> specifies the height in rows of the push buttons. In FoxPro for MS-DOS, a push button is always one line high, so the numeric expression <expN2> is ignored.

By default, the width of each individual button is determined by the length of the push button prompt text. The numeric expression <expN3> specifies the width (in columns) of each push button. A push button will never be sized smaller than its prompt.

By default, no rows are placed between vertical buttons and a single column is placed between horizontal buttons. The spacing between push buttons is specified with <expN4>. If you create vertical push buttons, <expN4> specifies the number of rows between the buttons. If you create horizontal buttons, <expN4> specifies the number of columns between the buttons.

In FoxPro for Windows, the push button font determines the size of the push buttons. The push button font is specified with the FONT clause. If the FONT clause is omitted, the push buttons use the font of the parent window (the main FoxPro window or a user-defined window).

The following example demonstrates how the SIZE clause controls the button spacing:

```
CLEAR
@ 2,2 GET mchoice FUNCTION '* OK;Cancel' ;
    DEFAULT 1 SIZE 2, 10, 1
@ 2,16 GET mchoice FUNCTION '* OK;Cancel' ;
    DEFAULT 1 SIZE 2, 10, 2
READ
```

ENABLE | DISABLE

Push buttons by default are enabled when READ or READ CYCLE is issued. You can prevent a set of buttons from being selected when READ or READ CYCLE is issued by including DISABLE.

Disabled push buttons cannot be selected and are displayed in the disabled colors. To disable individual push buttons instead of the entire set, see Disabled Buttons earlier in this section. Use SHOW GET ENABLED to enable a set of disabled push buttons.

MESSAGE <expC5>

The MESSAGE clause character expression <expC5> is displayed when a push button is selected. In FoxPro for MS-DOS, the message is centered by default on the last line of the desktop. The message location can be changed with SET MESSAGE.

In FoxPro for Windows, the message is placed in the Windows-style status bar. If the Windows-style status bar has been turned off with SET STATUS BAR OFF, the message is placed on the last line of the main FoxPro window.

VALID <expL1> | <expN5>

You can include an optional VALID expression <expL1> or <expN5> that is evaluated when a push button is *chosen*. That is, VALID isn't evaluated when you select a button, but when you actually choose a button by pressing Enter, Spacebar or clicking the button.

Typically, <expL1> or <expN5> is a user-defined function. With a user-defined function you can select, enable or disable other objects, open a Browse window, open another data entry screen; or move to a new record. CLEAR READ can be included in the user-defined function to terminate the READ.

<expL1>

When a logical value is returned to the VALID clause, the logical value is ignored and the push buttons remain the active control. However, you can specify a UDF that returns a logical value to the VALID clause and activates another object.

<expN5>

A VALID clause that includes a numeric expression is used to specify which object is activated after a push button is chosen. Objects are @ ... GET input fields, check boxes, lists, popups, spinners, text editing regions and each individual button in a set of push, radio and invisible buttons.

The expression <expN5> has one of three effects:

- When <expN5> = 0, the push button you choose remains the active button.
- When <expN5> is positive, <expN5> indicates the number of objects to advance. For example, when the last button in a set of push buttons is selected and VALID returns 1, the next object is activated. If <expN5> is greater than the number of objects remaining, the READ is terminated (unless READ CYCLE is issued to activate the objects).
- When <expN5> is negative, <expN5> indicates the number of objects to move back. For example, when the first button in a set of push buttons is selected and VALID returns -1, the previous object is activated. If <expN5> moves back past the first object, the READ is terminated (unless READ CYCLE is issued to activate the objects).

Example

The following is an example of what you can do with a user-defined function. Five buttons are displayed vertically. The button choices are used to position the record pointer. The N option is included so the READ isn't terminated when you choose a button. When you choose a button, the user-defined function GOREC is executed.

```
CLOSE DATABASES
USE customer

SET TALK OFF
DEFINE WINDOW one FROM 3,5 TO 18,20 FLOAT DOUBLE COLOR SCHEME 5
ACTIVATE WINDOW one
@ 1,2 GET mchoice FUNCTION '*NV Next;Prior;Top;Bottom;Quit';
    SIZE 2, 10, 1 VALID GOREC( ) DEFAULT 1
READ CYCLE
CLEAR WINDOW

PROCEDURE gorec
DO CASE
    CASE mchoice = 1
        SKIP
        IF EOF( )
            SKIP -1
        ENDIF
    CASE mchoice = 2
        SKIP -1
        IF BOF( )
            SKIP
        ENDIF
    CASE mchoice = 3
        GO TOP
    CASE mchoice = 4
        GO BOTTOM
    CASE mchoice = 5
        CLEAR READ
ENDCASE
@ 12,6 SAY ALLTRIM(STR(RECNO( )))
RETURN
```

WHEN <expL2>

The WHEN clause allows or prohibits selection of a set of push buttons based on the logical value of <expL2> which must evaluate to a logical true (.T.) before the push buttons can be selected. If <expL2> evaluates to a logical false (.F.), the push buttons cannot be selected and are skipped over if placed between other objects.

**COLOR SCHEME <expN6>
| COLOR <color pair list>**

If you do not include a COLOR clause, push button colors are determined by the color scheme for the desktop or the main FoxPro window; if push buttons are placed in a user-defined window, the window's color scheme determines the push button colors.

The colors of push buttons can be specified by including the number of an existing color scheme in the COLOR SCHEME clause or a set of color pairs in the COLOR clause.

A color scheme is a set of 10 predefined color pairs. The color pairs in a color scheme can be changed with SET COLOR OF SCHEME. In FoxPro for MS-DOS the color pairs in a color scheme can also be changed in the Color Picker.

A color pair is a set of two letters separated by a forward slash. The first color letter specifies the foreground color and the second letter specifies the background color.

For example, this color pair specifies a red foreground on a white background:

R/W

For a list of colors and their corresponding color letters, see the SET COLOR commands section in this chapter or the Color Table by Color Pair topic in the online help facility.

In FoxPro for Windows, a color pair can also be specified with a set of 6 RGB (Red Green Blue) color values separated by commas. The first 3 color values specify the foreground color and the second 3 color values specify the background color. The color values can range from 0 through 255.

In FoxPro for Windows, the R/W color pair in the example above can also be specified with this RGB color pair:

RGB(255,0,0,255,255,255)

In FoxPro for Windows, the COLOR clause does not affect the color of push buttons. Only the message color (if the message is not displayed in the status bar) is affected by the COLOR clause.

The following table lists the color pairs and what each color pair in the list controls.

Color Pair Number	Push Button Attribute
5	Message
6	Selected button prompt — FoxPro for MS-DOS only
7	Hot keys — FoxPro for MS-DOS only
9	Enabled button prompt — FoxPro for MS-DOS only
10	Disabled button prompt — FoxPro for MS-DOS only

The following example creates **OK** and **Cancel** push buttons with the hot keys O and C respectively. In addition, the buttons have the following color characteristics:

- The selected button is shown with a bright white prompt on a blue background (W+/B).
- Hot key characters are shown in red on a blue background (R/B).
- Enabled buttons are shown with a yellow prompt on a blue background (GR+/B).
- Disabled buttons are shown with a white prompt on a blue background (W/B).

When you skip a color pair, you must include a comma where the color pair is omitted.

Here are the commands:

```
STORE 1 TO mchoice
@ 2,2 GET mchoice FUNCTION '*' \<OK;\<CANCEL;\<DISABLED';
  COLOR , , , , W+/B, R/B, , GR+/B, W/B
READ
```

@ ... GET — Radio Buttons

Purpose	Creates a set of radio buttons or picture radio buttons
Syntax	<pre> @ <row, column> GET <memvar> <field> FUNCTION <expC1> PICTURE <expC2> [FONT <expC3> [, <expN1>]] [STYLE <expC4>] [DEFAULT <expr>] [SIZE <expN2>, <expN3> [, <expN4>]] [ENABLE DISABLE] [MESSAGE <expC5>] [VALID <expL1> <expN5>] [WHEN <expL2>] [COLOR SCHEME <expN6>] COLOR <color pair list>] </pre>
Remarks	FONT and STYLE clauses and support for RGB color pairs — FoxPro for Windows only
See Also	@ ... SAY, @ ... GET, _CUROBJ, Controls, OBJNUM(), READ, SET KEYCOMP, SHOW GETS, User-Defined Functions

Description This variation of @ ... GET creates a set of radio buttons. As their name suggests, *radio buttons* are similar to the buttons of a car radio – choosing a button makes your choice current and releases your previous choice. A bullet next to a radio button indicates that it is the current choice.

If you use the Screen Builder to create your data entry screens, you may not have to use this command at all. The Screen Builder automatically generates the commands that create radio buttons and picture radio buttons.

In FoxPro for MS-DOS, a set of radio buttons looks like this:

```

(•) None
( ) Single
( ) Double

```

In FoxPro for Windows, a set of radio buttons looks like this:

```

○ None
● Single
○ Double

```

The string of text to the right of each button is called a *prompt*. The text of the prompt is specified with the FUNCTION or PICTURE clause. Issue READ or READ CYCLE to activate the buttons.

You can create picture radio buttons in FoxPro for Windows. .BMPs (bitmap pictures) replace the radio button prompts.

Clauses <row, column>

The first button in a set of radio buttons is placed at the location designated by <row, column>. The row and column coordinates are numeric expressions. Row and column values can range from 0 through the maximum number of rows and columns on the desktop (FoxPro for MS-DOS), the main FoxPro window (FoxPro for Windows), or a user-defined window. Rows are numbered from top to bottom, and columns are numbered from left to right.

In FoxPro for Windows the position and size of radio buttons in the main FoxPro window or in a user-defined window are determined by the font of the main FoxPro window or the user-defined window. Most fonts Microsoft Windows uses can be displayed in a wide variety of sizes, and some are proportionally spaced; thus, positions and sizes depend on the current font. A row corresponds to the height of the current font. A column corresponds to the average width of a letter in the current font.

To facilitate precise positioning of radio buttons in the main FoxPro window or in a user-defined window, you can use decimal fractions for row and column coordinates in FoxPro for Windows.

GET <memvar> | <field>

When you choose a radio button, your choice is stored to the memory variable or array element <memvar> or to the field <field>. <memvar> or <field> must be of numeric or character type.

If <memvar> or <field> is of numeric type, a number corresponding to your radio button choice is stored to the memory variable, array element or field specified with <memvar> or <field>. For example, if you create three radio buttons and choose the second button, 2 is stored. If <memvar> or <field> is of character type, the prompt of the radio button you choose is stored to the memory variable, array element or field specified with <memvar> or <field>.

Initial Radio Button Selection

When a set of radio buttons is displayed, the value of `<memvar>` or `<field>` determines which radio button (if any) is initially selected.

If `<memvar>` or `<field>` is numeric, the radio button corresponding to the numeric value is initially selected. For example, if `<memvar>` or `<field>` is 1, the first radio button you define is initially selected. If `<memvar>` or `<field>` doesn't correspond to any of the buttons (the value is less than 1 or greater than the number of radio buttons) no button is initially selected.

If `<memvar>` or `<field>` is of character type, a case-sensitive comparison is made between `<memvar>` or `<field>` and each radio button prompt. When the comparison is made, all special characters and any leading or trailing spaces are removed from the button prompts. If a match is found, the matching button is initially selected. If a match can't be found, no button is initially selected.

FUNCTION `<expC1>` | PICTURE `<expC2>`

When creating a set of radio buttons you must include the `FUNCTION` clause, the `PICTURE` clause or both. There is no advantage to any of the three methods. The `FUNCTION` or `PICTURE` clause contains the radio button specification code `*R` and the text for the individual radio button prompts.

The `FUNCTION` character expression `<expC1>` must begin with `*R`. To create the radio button prompts, include a space after `*R` followed by a list of the prompts separated by semicolons. One button is created for each prompt. For example, the following `FUNCTION` clause creates three radio buttons with the prompts `None`, `Single` and `Double`:

```
... FUNCTION '*R None;Single;Double' ...
```

The `PICTURE` character expression `<expC2>` uses the same syntax as the `FUNCTION` expression except the `PICTURE` expression must begin with `@` followed by `*R`. For example, the following `PICTURE` clause creates three radio buttons with the prompts `None`, `Single` and `Double`:

```
... PICTURE '@*R None;Single;Double' ...
```

You can also include *both* the `FUNCTION` and `PICTURE` clauses to create radio buttons. If both are included, the `FUNCTION` character expression `<expC1>` must contain `*R` to create the radio buttons and can also include a space and some radio button prompts. The `PICTURE` character expression `<expC2>` can include prompts to create additional radio buttons.

Examples The following examples illustrate the various forms of syntax you can use to create a set of radio buttons. The radio buttons begin in the second row and second column. Three radio buttons are created with the prompts None, Single and Double. A number corresponding to the button chosen is stored to the memory variable MCHOICE.

FUNCTION clause only:

```
STORE 1 TO mchoice
@ 2,2 GET mchoice FUNCTION '*R None;Single;Double'
READ
```

```
STORE 1 TO mchoice
STORE '*R None;Single;Double' TO mfunc
@ 2,2 GET mchoice FUNCTION mfunc
READ
```

PICTURE clause only:

```
STORE 1 TO mchoice
@ 2,2 GET mchoice PICTURE '@*R None;Single;Double'
READ
```

```
STORE 1 TO mchoice
@ 2,2 GET mchoice PICTURE '@*R None;Single;Double'
READ
```

FUNCTION and PICTURE clause:

```
STORE 1 TO mchoice
@ 2,2 GET mchoice FUNCTION '*R' PICTURE ' None;Single;Double'
READ
```

```
STORE 1 TO mchoice
@ 2,2 GET mchoice FUNCTION '*R None;Single' PICTURE ' ;Double'
READ
```

Radio Buttons with Picture Prompts

In FoxPro for Windows, the prompt for a radio button can also be the name of a bitmap picture file with a .BMP extension. When a radio button uses a .BMP for a prompt, the radio button mimics the *behavior* of a radio button with a text prompt. For example, the .BMP is displayed but the radio button isn't displayed. Like choosing a radio button, choosing the picture prompt stores the radio button's value to <memvar> or <field>.

To use a .BMP in a radio button, add B to the radio button specification code. The FUNCTION and PICTURE clause specification codes that create a radio button are *R and @*R, respectively. To create a radio button with picture prompts, use the codes *RB and @*RB.



The pictures aren't clipped, reduced or enlarged to fit the radio button. Use the `SIZE` clause to adjust the size of the radio buttons to accommodate the pictures.

Picture Masks and Radio Buttons

In FoxPro for Windows, a picture radio button has three states: up, down and disabled. FoxPro automatically controls the appearance a picture radio button when it is in each of these three states, but you can override the default appearance by using a picture mask. A mask is a monochrome .BMP file with an .MSK extension. FoxPro automatically looks for a mask for a .BMP file in the same directory where the .BMP file is located. The mask must have the same base name as the .BMP file and an extension of .MSK.

A mask is used to control the transparent areas of a picture radio button. By default, the white areas are transparent. If a mask is present, the white areas of the mask, not the .BMP, are transparent.

In most cases, a mask isn't necessary. If you don't need anything in your picture to appear white or your .BMP has a white background, the picture radio button will appear as desired in the up, down and disabled states.

When a picture radio button has a mask and is in the up or down state, any white areas in the .BMP appear transparent, allowing the color of the button face to show through. However, you can maintain the white color of certain areas. Suppose that you have a .BMP of a dog on a white background and white eyes, and the button face is red. You want the background of the button to appear red but you want the dog's eyes to be white, not red. Make a mask that is the same size as the .BMP but includes only two colors, black and white. Leave the background of the mask white but make the dog — including his eyes — completely black. When the button is displayed, the background is red to match the button face but the dog's eyes are white.

When a picture radio button is disabled, any white areas in the .BMP appear transparent so the color of the button face shows through. Any non-white areas appear dark grey. If the button has a .MSK, all white areas in the mask are transparent so the color of the button face show through, and all black areas appear dark grey.

PICTURE and FUNCTION Options N, T, H and V

Additional options can be combined with the *R specification code to modify the behavior (N and T options) and appearance (H and V options) of radio buttons.

Option	Description
N	Do not terminate the READ when a radio button is chosen. This is the default behavior.
T	Terminate the READ when a radio button is chosen.
H	Position the radio buttons in a horizontal row.
V	Position the radio buttons in a vertical row. This is the default position.

You can combine the H and V options with the T and N options in the following ways: NH, NV, TH and TV. For example, the following clause creates a vertical row of radio buttons and doesn't cause the READ to terminate when a button is chosen:

```
... FUNCTION '*RNV ...'
```

Radio Buttons with Special Features

You can assign a hot key to a radio button or disable a radio button. To assign these special features to a radio button, you must include special characters when defining the prompt. The special characters are removed when the prompt is stored to <memvar> or <field>.

Hot Keys

In FoxPro for MS-DOS, a hot key is a highlighted letter in a radio button prompt that you can type to immediately choose the radio button. Pressing the hot key selects the radio button and chooses it. To assign a hot key, place a backslash and a less than sign (\<) before the desired character of the radio button prompt.

A hot key doesn't choose the radio button if the current object is a GET field, a text editing region, a popup or a list.

If the current object is an @ ... GET input field or a text editing region, pressing the hot key enters the character in the field or the text editing region. If the current object is a popup or a list, pressing the hot key selects the first option in the popup or list whose prompt begins with the hot key character.

In FoxPro for Windows, a hot key is an underlined letter in the radio button prompt that you can type to immediately choose the radio button. If the current object is an @ ... GET input field, a text editing region, a popup or a list and KEYCOMP is set to WINDOWS, you can press the Alt key and the hot key to choose the radio button.

The following example creates three radio buttons with the prompts None, Single and Double and assigns the hot keys N to None, S to Single and D to Double:

```
STORE 1 TO mchoice
@ 2,2 GET mchoice FUNCTION '*R' PICTURE '\<None;\<Single;\<Double'
READ
```

Disabled Radio Buttons

You can disable a radio button so it can't be selected or chosen. Disabled buttons are shown in disabled colors. To disable a single radio button, place two backslashes (\\) before the button's prompt. To disable a set of radio buttons, use DISABLE which is discussed later in this section.

The radio button with the prompt Single is disabled in this example:

```
STORE 1 TO mchoice
@ 2,2 GET mchoice FUNCTION '*R' PICTURE 'None;\\Single;Double'
READ
```

FONT <expC3> [, <expN1>] — FoxPro for Windows only

The character expression <expC3> is the name of the font, and the numeric expression <expN1> is the font size. For example, the following clause can be used to display the radio button prompts in 16 point Roman font:

```
FONT 'ROMAN' , 16
```

If you include the FONT clause but omit the font size <expN1>, a 10 point font is used. If the FONT clause is omitted and the radio buttons are placed in the main FoxPro window, the main FoxPro window font is used. If the FONT clause is omitted and the radio buttons are placed in a user-defined window, the user-defined window font is used.

If the font you specify is not available, Windows substitutes a font with similar font characteristics.

STYLE <expC4> — FoxPro for Windows only

In FoxPro for Windows, include the STYLE clause to specify a font style for the radio button prompts. The styles that are available for a font are determined by Windows. If the font style you specify is not available, Windows substitutes a font style with similar characteristics.

The font style is specified with <expC4>. If the STYLE clause is omitted, the standard font style is used.

Character	Font Style
B	Bold
I	Italic
N	Normal
O	Outline
Q	Opaque
S	Shadow
—	Strikeout
T	Transparent
U	Underline

You can include more than one character to specify a combination of font styles. For example, the following clause specifies Bold Italic:

```
STYLE 'BI'
```

DEFAULT <expr>

When you choose a radio button, your choice is saved in the memory variable, array element or field you specify. If you specify a memory variable that doesn't exist, it is automatically created and initialized if you include the DEFAULT clause. However, an array element isn't created if you specify an array element in a DEFAULT clause. The DEFAULT clause is ignored if the memory variable already exists or you specify a field.



If the DEFAULT clause isn't included and <memvar> doesn't exist, the error message "Variable not found" is displayed.

The DEFAULT expression <expr> determines the type of memory variable created and its initial value. <expr> must be of numeric or character type. Here are examples of DEFAULT clauses for radio buttons:

```
@ 2,2 GET mchoice FUNCTION '*R' + ;
      'None;Single;Double' DEFAULT 'Single'
READ

@ 2,2 GET mchoice FUNCTION '*R None;Single;Double';
      DEFAULT 2
READ
```

SIZE <expN2>, <expN3> [, <expN4>]

The size <expN2> clause specifies the height in rows of the radio buttons. In FoxPro for MS-DOS a radio button is always one line high, so the numeric expression <expN2> is ignored.

By default, the width of each individual button is determined by the length of the radio button prompt text. <expN3> specifies the width (in columns) of each radio button.

By default, no rows are placed between vertical buttons and a single column is placed between horizontal buttons. The spacing between radio buttons in rows is specified with <expN4>.

In FoxPro for Windows, the radio button font determines the size of the editing region. The radio button font is specified with the FONT clause. If the FONT clause is omitted, the radio buttons use the font of the parent window (the main FoxPro window or a user-defined window).

This example demonstrates how the SIZE clause controls the button spacing:

```
CLEAR
@ 2,2 GET mchoice1 FUNCTION '*R'+ '\<None;\<Single;\<Double' ;
    DEFAULT 'None' SIZE 1, 12, 1
@ 2,16 GET mchoice2 FUNCTION '*R'+ '\<None;\<Single;\<Double' ;
    DEFAULT 'None' SIZE 1, 12, 3
READ
```

ENABLE | DISABLE

Radio buttons are by default enabled when READ or READ CYCLE is issued. You can prevent a set of radio buttons from being selected when READ or READ CYCLE is issued by including DISABLE.

Disabled radio buttons cannot be selected and are displayed in the disabled colors. To disable individual radio buttons instead of an entire set, see Disabled Buttons earlier in this section. Use SHOW GET ENABLED to enable a set of disabled radio buttons.

MESSAGE <expC5>

The MESSAGE clause character expression <expC5> is displayed when a radio button is selected. In FoxPro for MS-DOS the message is by default centered on the last line of the desktop. The message location can be changed with SET MESSAGE.

In FoxPro for Windows, the message is placed in the Windows-style status bar. If the Windows-style status bar has been turned off with SET STATUS BAR OFF, the message is placed on the last line of the main FoxPro window.

VALID <expL1> | <expN5>

You can include an optional VALID expression <expL1> or <expN5> that is evaluated when a radio button is *chosen*. That is, VALID isn't evaluated when you select a radio button, but when you actually choose the radio button by pressing Enter, Spacebar or clicking on the radio button.

Typically, <expL1> or <expN5> is a user-defined function. With a user-defined functions let you can select, enable or disable other objects, open a Browse window, open another data entry screen or move to a new record. CLEAR READ can be included in the user-defined function to terminate the READ.

<expL1>

When a logical value is returned to the VALID clause, the logical value is ignored and the radio buttons remain the active control. However, you can specify a UDF that returns a logical value to the VALID clause and activates another object.

<expN5>

A VALID clause that includes a numeric expression is used to specify which object is activated after a radio button is chosen. Objects are @ ... GET input fields, check boxes, lists, popups, spinners, text editing regions and each individual button in a set of push, radio and invisible buttons.

The expression <expN5> has one of three effects:

- When <expN5> = 0, the chosen radio button remains the active button.
- When <expN5> is positive, <expN5> indicates the number of objects to advance. For example, when the last radio button in a set of radio buttons is selected and VALID returns 1, the next object is activated. If <expN5> is greater than the number of remaining objects, the READ is terminated (unless READ CYCLE is issued to activate the objects).
- When <expN5> is negative, <expN5> indicates the number of objects to move back. For example, when the first radio button in a set of radio buttons is selected and VALID returns -1, the previous object is activated. If <expN5> moves back past the first object, the READ is terminated (unless READ CYCLE is issued to activate the objects).

WHEN <expL2>

The WHEN clause allows or prohibits selection of a set of radio button based on the logical value of <expL2>, which must evaluate to a logical true (.T.) before the radio buttons can be selected. If <expL2> evaluates to a logical false (.F.), the radio buttons cannot be selected and are skipped over if placed between other objects.

COLOR SCHEME <expN6>

| COLOR <color pair list>

If you do not include a COLOR clause, radio button colors are determined by the color scheme for the desktop or the main FoxPro window; if radio buttons are placed in a user-defined window, the window's color scheme determines the radio button colors.

The colors of radio buttons can be specified by including the number of an existing color scheme in the COLOR SCHEME clause or a set of color pairs in the COLOR clause.

A color scheme is a set of 10 predefined color pairs. The color pairs in a color scheme can be changed with SET COLOR OF SCHEME. In FoxPro for MS-DOS the color pairs in a color scheme can also be changed in the Color Picker.

A color pair is a set of two letters separated by a forward slash. The first color letter specifies the foreground color and the second letter specifies the background color.

For example, this color pair specifies a red foreground on a white background:

R/W

For a list of colors and their corresponding color letters, see the SET COLOR commands section in this chapter or the Color Table by Color Pair topic in the online help facility.

In FoxPro for Windows, a color pair can also be specified with a set of 6 RGB (Red Green Blue) color values separated by commas. The first 3 color values specify the foreground color and the second 3 color values specify the background color. The color values can range from 0 through 255.

In FoxPro for Windows, the R/W color pair in the example above can also be specified with this RGB color pair:

RGB(255,0,0,255,255,255)

The following table lists the color pairs and what each color pair in the list controls.

Color Pair Number	Radio Button Attribute
5	Message
6	Selected radio prompt
7	Hot keys
9	Enabled radio prompt
10	Disabled radio prompt

This example shows how to override one color scheme with another predefined color scheme:

```
STORE 1 TO mchoice
@ 2,2 GET mchoice FUNCTION '*R None;Single;Double';
  COLOR SCHEME 4
READ
```

The following example creates **None**, **Single** and **Double** radio buttons with the hot keys N for **None** and S for **Single**. In addition, the buttons have the following color characteristics:

- The selected button is shown with a bright white prompt on a blue background (W+/B).
- Hot key characters are shown in red on a blue background (R/B).
- Enabled buttons are shown with a yellow prompt on a blue background (GR+/B).
- Disabled buttons are shown with a white prompt on a blue background (W/B).

When you skip a color pair, you must include a comma where the color pair is omitted.

Here are the commands:

```
STORE 1 TO mchoice
@ 2,2 GET mchoice FUNCTION '*R \<None;\<Single;\Double';
  COLOR ,,,,W+/B,R/B,,GR+/B,W/B
READ
```

Example

```
CLOSE DATABASES
SET TALK OFF
CLEAR
USE customer

*** Define and activate window for GET information. ***
DEFINE WINDOW one FROM 3, 5 TO 13, 33 FLOAT DOUBLE COLOR SCHEME 5
ACTIVATE WINDOW one

*** Define a set of radio buttons. ***
@ 1,2 GET mchoice FUNCTION '*RNV Next;Prior;Top;Bottom';
    SIZE 1, 10, 1 DEFAULT 'Next'

*** Define a set of push buttons to terminate the READ. ***
@ 3,15 GET okcancel FUNCTION '*V \!OK;\?Cancel' DEFAULT 1;
    SIZE 1, 10, 1
READ CYCLE

*** If cancel was selected, ***
IF okcancel = 2
    WAIT WINDOW 'Cancel Selected' NOWAIT
ELSE

*** If OK is selected, move record pointer to users choice. ***
DO CASE
    CASE mchoice = 'Next'
        SKIP
        IF EOF( )
            SKIP -1
        ENDIF
    CASE mchoice = 'Prior'
        IF !BOF( )
            SKIP -1
        ENDIF
    CASE mchoice = 'Top'
        GO TOP
    CASE mchoice = 'Bottom'
        GO BOTTOM
ENDCASE

*** Display users choice ***
WAIT WINDOW mchoice + ' Selected' NOWAIT
ENDIF
CLEAR WINDOW
```

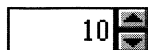
@ ... GET — Spinners

Purpose	Creates a spinner
Syntax	<pre> @ <row, column> GET <memvar> <field> SPINNER <expN1> [, <expN2> [, <expN3>]] [FUNCTION <expC1>] [PICTURE <expC2>] [FONT <expC3> [, <expN4>]] [STYLE <expC4>] [DEFAULT <expN5>] [SIZE <expN6>, <expN7>] [ENABLE DISABLE] [MESSAGE <expC5>] [RANGE [<expN8>] [, <expN9>]] [VALID <expL1> <expN10> [ERROR <expC6>]] [WHEN <expL2>] [COLOR SCHEME <expN11> COLOR <color pair list>] </pre>
Remarks	FoxPro for Windows only
See Also	@ ... SAY, @ ... GET, Controls, _CUROBJ, OBJNUM(), READ, SHOW GET, User-Defined Functions

Description This command creates a spinner in FoxPro for Windows. A *spinner* is a control that lets you “spin” through a set of numeric values displayed in a text box.

If you use the Screen Builder to create your data entry screens, you may not have to use this command at all. The Screen Builder automatically generates the commands that create spinners.

A spinner looks like this:



Clicking the up and down arrows changes the numeric value. You can also enter a value directly into the spinner text box. When the value you want is displayed in the text box, press Enter or Tab to move to the next object.

Clauses **<row, column>**

The spinner is placed at the location designated by <row, column>. The row and column coordinates are numeric expressions. Row and column values can range from 0 through the maximum number of rows and columns on the desktop (FoxPro for MS-DOS), the main FoxPro window (FoxPro for Windows), or a user-defined window. Rows are numbered from top to bottom, and columns are numbered from left to right.

In FoxPro for Windows the position and size of the spinner in the main FoxPro window or in a user-defined window are determined by the font of the main FoxPro window or the user-defined window. Most fonts Microsoft Windows uses can be displayed in a wide variety of sizes, and some are proportionally spaced; thus, positions and sizes depend on the current font. A row corresponds to the height of the current font. A column corresponds to the average width of a letter in the current font.

To facilitate precise positioning of a spinner in the main FoxPro window or in a user-defined window, you can use decimal fractions for row and column coordinates in FoxPro for Windows.

GET <memvar> | <field>

The numeric value contained in <memvar> or <field> is the initial value displayed in the spinner before READ or READ CYCLE is issued.

When you choose a value in the spinner, your choice is stored to the numeric memory variable or array element <memvar> or to the numeric field <field>.

SPINNER <expN1> [, <expN2> [, <expN3>]]

The values displayed in the spinner are specified with <expN1>, <expN2> and <expN3>.

The value displayed in the spinner text box is incremented or decremented by the value specified with <expN1>. For example, if <expN1> is 2 and you click the up or down arrow, the value in the text box increases or decreases by 2.

The following example defines a spinner for the memory variable MCHOICE. The spinner increments by 1 with a lower limit of -5 and an upper limit of 24. The initial value displayed is 3.

```
@ 2,2 GET mchoice SPINNER 1, -5, 24 DEFAULT 3
```

The minimum value that can be displayed in the spinner is specified with <expN2>, and the maximum value that can be displayed is specified with <expN3>. However, using the keyboard you can enter a value in the text box that lies outside the range specified with <expN2> and <expN3>. Use VALID or RANGE to control the what values can be entered in the text box with the keyboard.

FUNCTION <expC1> | PICTURE <expC2>

When creating spinners, you can include the FUNCTION clause, the PICTURE clause or both. There is no advantage to any of the three methods. The FUNCTION and PICTURE clauses control how <mem-var> or <field> is displayed and edited in the text box.

The following table shows how the FUNCTION clause <expC1> determines how values are displayed in the spinner text box.

<expC1>	Purpose
B	Left justifies the value in the text box.
I	Centers the value in the text box.
J	Right justifies the value in the text box.
K	Selects the entire value for editing when the cursor is moved to the spinner.
L	Displays leading zeros before the value.
Z	Displays the value as blank if it is 0.
^	Displays the value using scientific notation.
\$	Displays the value in a currency format. The currency symbol appears to the left or right of the value depending on the setting of SET CURRENCY.

The PICTURE clause character expression <expC2> determines what values can be entered in the text box and how values are displayed. The PICTURE clause expression <expC2> operates on individual characters in the spinner. You should include enough mask characters to allow entry of the highest value you anticipate in addition to any signs or symbols.

<expC2>	Purpose
9	Allows you to enter digits and signs (±).
#	Allows you to enter digits, blanks and signs (±).
\$	Displays the current currency symbol (specified with SET CURRENCY).
*	Asterisks are displayed to the left of the value.
.	A period specifies the decimal point position.
,	Commas can be included to separate digits to the left of the decimal point.

FONT <expC3> [, <expN4>]

The character expression <expC3> is the name of the font, and the numeric expression <expN4> is the font size. If <expN4> is omitted, a 10 point font is used. For example, the following command displays the text in the spinner in 16 point Roman font:

```
@ 2, 2 GET mchoice SPINNER 1, 1, 10 DEFAULT 3 FONT 'ROMAN', 16
```

If you include the FONT clause but omit the font size <expN4>, a 10 point font is used.

If the FONT clause is omitted and the spinner is placed in the main FoxPro window, the main FoxPro window font is used. If the FONT clause is omitted and the spinner is placed in a user-defined window, the user-defined window font is used.

If the font you specify is not available, Windows substitutes a font with similar font characteristics.

STYLE <expC4> — FoxPro for Windows only

In FoxPro for Windows, include the STYLE clause to specify a font style for the spinner. The styles that are available for a font are determined by Windows. If the font style you specify is not available, Windows substitutes a font style with similar characteristics.

The font style is specified with <expC4>. If the STYLE clause is omitted, the standard font style is used.

Character	Font Style
B	Bold
I	Italic
N	Normal
O	Outline
S	Shadow
—	Strikeout
U	Underline

You can include more than one character to specify a combination of font styles. For example, the following command displays the text in the spinner in Bold Italic:

```
@ 2, 2 GET mchoice SPINNER 1, 1, 10 DEFAULT 3 STYLE 'BI'
```

DEFAULT <expN5>

When you exit a spinner, the value in the spinner is saved to the memory variable, array element or field you specify. If you specify a memory variable that doesn't exist, it is automatically created and initialized to <expN5> if you include DEFAULT. However, an array element isn't created if you specify an array element in a DEFAULT clause. The DEFAULT clause is ignored if the memory variable already exists or you specify a field.



If the DEFAULT clause isn't included and <memvar> doesn't exist, the error message "Variable not found" is displayed.

SIZE <expN6>, <expN7>

The SIZE <expN6> expression specifies the height of the spinner. The height of the spinner is a multiple of the height of the current font or the font you specify when you create the spinner. For example, if <expN6> is 2, the height of the spinner is twice the height of the current font or the font you specify.

The numeric expression <expN7> specifies the width of the spinner. The width of the spinner is a multiple of the average width of a letter in the current font or the font you specify when you create the spinner. For example, if <expN7> is 2, the width of the spinner is twice the width of the average width of the current font or the font you specify.

In FoxPro for Windows, the spinner font determines the size of the spinner. The spinner font is specified with the FONT clause. If the FONT clause is omitted, the spinner uses the font of its parent window (the main FoxPro window or a user-defined window).

ENABLE | DISABLE

Spinners are enabled when READ or READ CYCLE is issued. You can prevent a spinner from being selected by including DISABLE. The spinner is disabled when READ or READ CYCLE is issued.

Disabled spinners cannot be selected and are displayed in the disabled colors. Use SHOW GET ENABLE to enable a disabled spinner.

MESSAGE <expC5>

The optional MESSAGE clause character expression <expC5> is displayed when the spinner is selected. The message is placed in the Windows-style status bar. If the Windows-style status bar has been turned off with SET STATUS BAR OFF, the message is placed on the last line of the main FoxPro window.

RANGE [<expN8>] [, <expN9>]

Use the RANGE clause to specify a range of acceptable spinner values. If the value entered in the spinner text box isn't within the specified range, a message showing the correct range is displayed. To override the default range message, create a routine using ON READERROR.

The lower boundary of the range is specified with <expN8>, the upper boundary with <expN9>. Either <expN8> or <expN9> can be omitted, but not both. If one bound is omitted, that end of the range isn't checked.

The range isn't checked if you press Enter without changing the initial spinner value.

VALID <expL1> | <expN10> [ERROR <expC6>]

You can include an optional VALID expression <expL1> or <expN10> that is evaluated when you attempt to exit the spinner.



Unlike the RANGE check, the VALID check is *always* performed when you attempt to exit the spinner, unless you exit by pressing Escape. The RANGE check is only performed when a change is made to the initial spinner value.

Typically, <expL1> or <expN10> is a user-defined function. With a user-defined function you can select, enable or disable other objects, open a Browse window, open another data entry screen or move to a new record. CLEAR READ can be included in the user-defined function to terminate the READ.

<expL1>

If <expL1> evaluates to a logical true (.T.), the value in the spinner is considered correct and the spinner is exited.

If <expL1> evaluates false (.F.), the value entered in the spinner is considered incorrect and the spinner remains the active control. A message is displayed in the status bar directing you to reenter the data after pressing the Spacebar.

<expN10>

A VALID clause that includes a numeric expression is used to specify which object is activated after a spinner is chosen. Objects include @ ... GET input fields, check boxes, lists, popups, spinners, text editing regions and each individual button in a set of push, radio and invisible buttons.

- When <expN10> = 0, the spinner remains the active object and the MESSAGE <expC5> and ERROR <expC6> messages are suppressed. A special error message routine can be written as part of a user-defined function called by the VALID clause.
- When <expN10> is positive, <expN10> specifies the number of objects to advance. For example, when the spinner is selected and VALID returns 1, the next object is activated. If <expN10> is greater than the number of remaining objects, the READ is terminated (unless READ CYCLE is issued to activate the objects).
- When <expN10> is negative, <expN10> specifies the number of objects to move back. For example, when the spinner is selected and VALID returns -1, the previous object is activated. If <expN10> moves back past the first object, the READ is terminated (unless READ CYCLE is issued to activate the objects).

ERROR <expC6>

The ERROR clause lets you specify a custom error message that is displayed when the VALID clause evaluates to false (.F.). <expC6> is displayed in place of the default error message.

WHEN <expL2>

The optional WHEN clause allows or prohibits the selection of a spinner based on the logical value of <expL2> which must evaluate to a logical true (.T.) before the spinner can be selected. If <expL2> evaluates to a logical false (.F.), the spinner cannot be selected and is skipped over if placed between other objects.

COLOR SCHEME <expN11>

I COLOR <color pair list>

If you do not include a COLOR clause, spinner colors are determined by the color scheme for the desktop or the main FoxPro window; if a spinner is placed in a user-defined window, the window's color scheme determines the spinner's colors.

The colors of a spinner can be specified by including the number of an existing color scheme in the COLOR SCHEME clause or a set of color pairs in the COLOR clause.

A color scheme is a set of 10 predefined color pairs. The color pairs in a color scheme can be changed with SET COLOR OF SCHEME. In FoxPro for MS-DOS the color pairs in a color scheme can also be changed in the Color Picker.

A color pair is a set of two letters separated by a forward slash. The first color letter specifies the foreground color and the second letter specifies the background color.

For example, this color pair specifies a red foreground on a white background:

R/W

For a list of colors and their corresponding color letters, see the SET COLOR commands section in this chapter or the Color Table by Color Pair topic in the online help facility.

In FoxPro for Windows, a color pair can also be specified with a set of 6 RGB (Red Green Blue) color values separated by commas. The first 3 color values specify the foreground color and the second 3 color values specify the background color. The color values can range from 0 through 255.

In FoxPro for Windows, the R/W color pair in the example above can also be specified with this RGB color pair:

RGB(255,0,0,255,255,255)

@ ... EDIT — Text Editing Regions

Purpose	Creates a text editing region
Syntax	<pre> @ <row, column> EDIT <memvar> <field> SIZE <expN1>, <expN2> [, <expN3>] [FUNCTION <expC1>] [FONT <expC2> [, <expN4>]] [STYLE <expC3>] [DEFAULT <expr>] [ENABLE DISABLE] [MESSAGE <expC4>] [VALID <expL1> <expN5> [ERROR <expC5>]] [WHEN <expL2>] [NOMODIFY] [SCROLL] [TAB] [COLOR SCHEME <expN6> COLOR <color pair list>] </pre>
Remarks	FONT and STYLE clauses and support for RGB color pairs — FoxPro for Windows only
See Also	@ ... SAY, @ ... GET, Controls, _CUROBJ, OBJNUM(), READ, SHOW GET, User-Defined Functions

Description This command creates a rectangular text editing region for editing a memory variable or array element <memvar> or a field or a memo field <field>. The memory variable, array element or field must be of character type.

If you use the Screen Builder to create your data entry screens, you may not have to use this command at all. The Screen Builder automatically generates the commands that create text editing regions.

Note that this control uses EDIT in its syntax instead of GET like other FoxPro controls such as check boxes, radio and push buttons.

When a text editing region is activated with READ or READ CYCLE, the contents of <memvar> or <field> are displayed in the region for editing. When you exit the text editing region, your changes can be saved in the memory variable, array element or field specified with <memvar> or <field>.

Press Tab or Ctrl+Tab to save your changes and move to the next object. To save your changes and move to the previous object, press Shift+Tab. Press Escape to discard your changes and move to the next object. If you're editing a memo field and a VALID clause is included, your changes may or may not be saved — the value returned by the VALID clause determines whether your changes are saved. See the VALID clause discussion later in this section for more information.

All standard FoxPro editing features, such as cutting, copying and pasting, are available in the text editing region. The text in the text editing region scrolls vertically, and words are wrapped horizontally.

Clauses **<row, column>**

The upper left corner of the text editing region is placed at the location designated by <row, column>. The row and column coordinates are numeric expressions. Row and column values can range from 0 through the maximum number of rows and columns on the desktop (FoxPro for MS-DOS), the main FoxPro window (FoxPro for Windows), or a user-defined window. Rows are numbered from top to bottom, and columns are numbered from left to right.

In FoxPro for Windows the position and size of a text editing region in the main FoxPro window or in a user-defined window are determined by the font of the main FoxPro window or the user-defined window. Most fonts Microsoft Windows uses can be displayed in a wide variety of sizes, and some are proportionally spaced; thus, positions and sizes depend on the current font. A row corresponds to the height of the current font. A column corresponds to the average width of a letter in the current font.

To facilitate precise positioning of a text editing region on the main FoxPro window or in a user-defined window, you can use decimal fractions for row and column coordinates in FoxPro for Windows.

SIZE <expN1>, <expN2> [, <expN3>]

The SIZE clause must be included to specify the height and width of the text editing region. The height of the text editing region in rows is specified by <expN1>, and the width in columns is specified by <expN2>.

If <expN1> is 1, a special one-line text editing region is created. You can scroll horizontally in the one-line region. Pressing Enter moves you to the next object.

The optional numeric expression <expN3> specifies the number of characters in the memory variable, array element or field that can be edited, starting with the first character. If <expN3> is omitted, all of the memory variable, array element or field can be edited.

In FoxPro for Windows, the text editing region font determines the size of the editing region. The text editing region font is specified with the FONT clause. If the FONT clause is omitted, the text editing region uses the font of its parent window (the main FoxPro window or a user-defined window).

FUNCTION <expC1>

You can include the I or J option in the FUNCTION clause to specify how text is justified in the text editing region. Any additional characters included in the FUNCTION clause are ignored.

Option	Purpose
I	Centers text in the text editing region.
J	Right-justifies text. Text in the editing region is left-justified by default.

This example creates a text editing region in which text is right justified:

```
STORE 'This will be edited' TO text
@ 2,2 EDIT text FUNCTION 'J' SIZE 5, 50
READ
```

FONT <expC2> [, <expN4>] — FoxPro for Windows only

The character expression <expC2> is the name of the font, and the numeric expression <expN4> is the font size. For example, the following clause can be used to display the text in the text editing region in 16 point Roman font:

```
FONT 'ROMAN', 16
```

If you include the FONT clause but omit the font size <expN4>, a 10 point font is used.

If the FONT clause is omitted and the text editing region is placed in the main FoxPro window, the main FoxPro window font is used. If the FONT clause is omitted and the text editing region is placed in a user-defined window, the user-defined window font is used.

If the font you specify is not available, Windows substitutes a font with similar font characteristics.

STYLE <expC3> — FoxPro for Windows only

In FoxPro for Windows, include the STYLE clause to specify a font style for the text editing region. The styles that are available for a font are determined by Windows. If the font style you specify is not available, Windows substitutes a font style with similar characteristics.

The font style is specified with <expC3>. If the STYLE clause is omitted, the standard font style is used.

Character	Font Style
B	Bold
I	Italic
N	Normal
O	Outline
S	Shadow
—	Strikeout
U	Underline

You can include more than one character to specify a combination of font styles. For example, the following clause specifies Bold Italic:

```
STYLE 'BI'
```

DEFAULT <expr>

When you edit the text in a text editing region, the text is read from and saved to the memory variable, array element or field specified with <memvar> or <field>. If you specify a memory variable that doesn't exist, it is automatically created and initialized if you include the DEFAULT clause. However, an array element isn't created if you specify an array element in a DEFAULT clause. The DEFAULT clause is ignored if the memory variable already exists or you specify a table/.DBF or memo field.



If the DEFAULT clause isn't included and <memvar> doesn't exist, the error message "Variable not found" is displayed.

The DEFAULT expression <expr> determines the type of memory variable created and its initial value. It must be of character or memo type. Here are examples of DEFAULT clauses for text editing regions:

```
@ 2,2 EDIT text DEFAULT 'This will be edited' SIZE 5, 50
READ
```

```
@ 2,2 EDIT text DEFAULT SPACE(20) SIZE 5, 50
READ
```

ENABLE | DISABLE

Text editing regions are by default enabled when READ or READ CYCLE is issued. Include DISABLE to prevent a text editing region from being selected when READ or READ CYCLE is issued. A disabled text editing region cannot be selected and is displayed in disabled colors. Use SHOW GET ENABLED to enable a disabled text editing region.

MESSAGE <expC4>

The MESSAGE clause character expression <expC4> is displayed when a text editing region is selected. In FoxPro for MS-DOS the message is centered on the last line of the desktop by default. The message location can be changed with SET MESSAGE.

In FoxPro for Windows, the message is placed in the Windows-style status bar. If the Windows-style status bar has been turned off with SET STATUS BAR OFF, the message is placed on the last line of the main FoxPro window.

VALID <expL1> | <expN5>

You can include an optional VALID expression <expL1> or <expN5> that is evaluated when you exit the text editing region.

Typically, <expL1> or <expN5> is a user-defined function. With a user-defined function you can select, enable or disable other objects, open a Browse window, open another data entry screen or move to a new record. CLEAR READ can be included in the user-defined function to terminate the READ.

<expL1>

If <expL1> evaluates to true (.T.), your changes are saved if you press Tab, Ctrl+Tab or Shift+Tab to exit the text editing region. If you press Escape, your changes are discarded.

If you press Tab, Ctrl+Tab or Shift+Tab to save changes to a memo field and <expL1> evaluates to false (.F.), a message is displayed prompting you for proper input. If you then press Escape to exit the text editing region, your changes are saved.

If <expL1> evaluates to false and you are editing a memory variable, array element or field, pressing Escape exits the text editing region and discards your changes.

<expN5>

A VALID clause that includes a numeric expression is used to specify which object is activated after the text editing region is exited.

The expression <expN5> has one of these effects:

- When <expN5> = 0, the text editing region remains the active control and the MESSAGE <expC4> and ERROR <expC5> messages are suppressed. A special error message routine can be written as part of a user-defined function called by the VALID clause.
- When <expN5> is positive, <expN5> specifies the number of objects to advance. For example, when the text editing region is selected and VALID returns 1, the next object is activated. If <expN5> is greater than the number of remaining objects, the READ is terminated (unless READ CYCLE is issued to activate the objects).
- When <expN5> is negative, <expN5> specifies the number of objects to move back. For example, when text editing region is selected and VALID returns -1, the previous object is activated. If <expN5> moves back past the first object, the READ is terminated (unless READ CYCLE is issued to activate the objects).

ERROR <expC5>

When the VALID clause expression <expL1> evaluates to false (.F.), the default message “Invalid Input” is displayed. Including the ERROR clause lets you replace this default message with the character expression <expC5>.

WHEN <expL2>

The WHEN clause allows or prohibits selection of the editing region based on the logical value of <expL2>, which must evaluate to a logical true value (.T.) before the text editing region can be selected. If <expL2> evaluates to false (.F.), the region cannot be selected and is skipped if placed between other objects.

NOMODIFY

Including the NOMODIFY displays the text editing region but doesn’t allow editing. You can scroll through the region and copy text, but you can’t modify the text.

SCROLL

If SCROLL is included, a scroll bar is placed to the right of the text editing region. The scroll bar lets you move quickly through the text with the mouse and provides a visual indication of your position in the text editing region. The scroll bar is displayed only when the text is too long to fit into the text editing region.

TAB

By default, pressing the Tab key doesn’t insert a tab into the text in a text editing region. Pressing Tab or Ctrl+Tab saves your changes, exits the text editing region and advances you to the next object.

If you include TAB, pressing Tab inserts a tab into the text at the current cursor location. Press Ctrl+Tab to save your changes, exit the text editing region and advance to the next object.

COLOR SCHEME <expN6>

| COLOR <color pair list>

By default, the colors of text editing regions are derived from color scheme 2, User Menus. If you do not include a COLOR clause, text editing region colors are determined by the color scheme for the desktop or the main FoxPro window; if a text editing region is placed in a user-defined window, the window's color scheme determines the text editing region colors.

The colors of text editing regions can be specified by including the number of an existing color scheme in the COLOR SCHEME clause or a set of color pairs in the COLOR clause.

A color scheme is a set of 10 predefined color pairs. The color pairs in a color scheme can be changed with SET COLOR OF SCHEME. In FoxPro for MS-DOS the color pairs in a color scheme can also be changed in the Color Picker.

A color pair is a set of two letters separated by a forward slash. The first color letter specifies the foreground color and the second letter specifies the background color.

For example, this color pair specifies a red foreground on a white background:

R/W

For a list of colors and their corresponding color letters, see the SET COLOR commands section in this chapter or the Color Table by Color Pair topic in the online help facility.

In FoxPro for Windows, a color pair can also be specified with a set of 6 RGB (Red Green Blue) color values separated by commas. The first 3 color values specify the foreground color and the second 3 color values specify the background color. The color values can range from 0 through 255.

In FoxPro for Windows, the R/W color pair in the example above can also be specified with this RGB color pair:

RGB(255,0,0,255,255,255)

Color Pair Number	Text Editing Region Attributes
2	Enabled text editing region
3	Scroll bar*
5	Message
6	Selected text editing region
10	Disabled text editing region

* Controls drawn in background color.
Bar drawn in dim of foreground color.

The following example defines a text editing region with the following color characteristics:

- When text is selected, the text is shown with bright white text on a blue background (W+/B).
- When enabled, the region is shown with yellow text on a blue background (GR+/B).
- When disabled, the region is shown with white text on a blue background (W/B).

When you skip a color pair, you must include a comma where the color pair is omitted.

Here are the commands:

```
STORE 'To be edited' TO text
@ 2,2 EDIT text SIZE 5, 50 COLOR ,GR+/B,,,W+/B,,,W/B
READ
```

The following example shows how to override the color scheme of the desktop or main FoxPro window with another predefined color scheme:

```
ACTIVATE SCREEN
STORE 'To be edited' TO text
@ 2,2 EDIT text COLOR SCHEME 4 SIZE 5, 50
READ
```

@ ... BOX

Purpose	Draws a box using specified coordinates
Syntax	@ <row1, column1>, <row2, column2> BOX [<expC>]
See Also	@ ... TO

Description This command draws a box on the desktop in FoxPro for MS-DOS, in the main FoxPro window in FoxPro for Windows, or in a user-defined window.

Parameters <row1, column1>, <row2, column2>

<row1, column1> are the coordinates of the upper-left corner of the box, and <row2, column2> are the coordinates of the lower right corner of the box. If row1 and row2 are the same, a horizontal line is drawn. If column1 and column2 are the same, a vertical line is drawn.

<expC>

The characters used to draw the box can be specified by including <expC>, the character expression, which can include nine characters — one for each corner, one for each side and one to fill the box. The characters are displayed starting with the upper-left corner and continuing clockwise. If a ninth character is specified, it is used to fill the interior of the box. If just one character is specified, it is used to draw the entire border. If <expC> is omitted, the box is drawn using a single-line.

Examples This example draws a single-line box:

```
@ 10,20,14,60 BOX
```

The next example draws a box composed of shaded rectangles. REPLICATE() returns nine rectangles, which @ ... BOX uses to draw each corner and side of the box. It then fills the box with the shaded rectangles.

```
@ 10,20,14,60 BOX REPLICATE(CHR(177),9)
```

@ ... CLEAR

Purpose	Clears a portion of the screen or active window
Syntax	@ <row1, column1> [CLEAR CLEAR TO <row2, column2>]
See Also	CLEAR
Description	This command clears a portion of the screen in FoxPro for MS-DOS, the main FoxPro window in FoxPro for Windows, or a user-defined window.
Clauses	<row1, column1> These are the coordinates of the upper-left corner of the area to clear. <row2, column2> If you omit CLEAR or CLEAR TO, FoxPro clears row1 from column1 to the end of the row. If you include CLEAR, FoxPro clears a rectangular area whose upper-left corner begins at row1 and column1 and continues to the bottom-right corner of the desktop, main FoxPro window, or user-defined window. If you include CLEAR TO <row2, column2>, FoxPro clears a rectangular area whose upper-left corner begins at row1 and column1 and ends at row2 and column2.
Examples	This example clears the screen, main FoxPro window, or user-defined window from the second row to the bottom of the desktop or window. <pre>@ 2,0 CLEAR</pre> The next example clears a rectangular region. The upper-left corner of the area is cleared from row 10, column 0 to row 20, column 20. <pre>@ 10,0 CLEAR TO 20,20</pre>

@ ... FILL

Purpose	Changes the colors of existing text within an area of the screen
Syntax	@ <row1, column1> FILL TO <row2, column2> COLOR SCHEME <expN> COLOR <color pair list>
Remarks	RGB color pairs — FoxPro for Windows only
See Also	SET COLOR

Description This command changes the colors of text within a rectangular area of the desktop in FoxPro for MS-DOS, the main FoxPro window in FoxPro for Windows, or a user-defined window. You can set the foreground and background color attributes for *existing text only*. Any text output to the same area after you issue @ ... FILL is displayed in the default screen colors. See the SET COLOR command for more information on defining colors.

Clauses <row1, column1>
TO <row2, column2>

The upper-left corner of the area is specified with <row1, column1>, and the bottom-right corner is specified with <row2, column2>.

**COLOR SCHEME <expN>
COLOR <color pair list>**

The color of the rectangular area can be specified by including the number of an existing color scheme in the COLOR SCHEME clause or a set of color pairs in the COLOR clause.

A color scheme is a set of 10 predefined color pairs. The color pairs in a color scheme can be changed with SET COLOR OF SCHEME. In FoxPro for MS-DOS the color pairs in a color scheme can also be changed in the Color Picker.

A color pair is a set of two letters separated by a forward slash. The first color letter specifies the foreground color and the second letter specifies the background color.

For example, this color pair specifies a red foreground on a white background:

R/W

For a list of colors and their corresponding color letters, see the SET COLOR commands section in this chapter or the Color Table by Color Pair topic in the online help facility.

In FoxPro for Windows, a color pair can also be specified with a set of 6 RGB (Red Green Blue) color values separated by commas. The first 3 color values specify the foreground color and the second 3 color values specify the background color. The color values can range from 0 through 255.

In FoxPro for Windows, the R/W color pair in the example above can also be specified with this RGB color pair:

```
RGB(255,0,0,255,255,255)
```

Only the first color pair in the specified color scheme or color pair list determines the color of the area.

If you omit one of the color clauses, the rectangular portion is cleared. An area can also be cleared with @ ... CLEAR.

Example

The following example clears the desktop or the main FoxPro window and fills an area with color.

```
CLEAR  
@ 4, 1 FILL TO 10, 8 COLOR GR+/B
```

@ ... MENU

Purpose	Creates a menu popup
Syntax	@ <row, column> MENU <array>, <expN1> [, <expN2>] [TITLE <expC>] [SHADOW]
Remarks	Included for backward compatibility — use the Menu Builder to create menus and menu popups
See Also	CREATE MENU, DEFINE POPUP, MENU, MENU TO, READ MENU

Description This command creates a menu popup. A menu popup is a bordered box that contains a list of options you can choose from. Menu popups are activated when READ MENU is issued. Only one menu popup can be active at a time.

DEFINE POPUP and ACTIVATE POPUP are another pair of commands that create and activate menu popups. DEFINE POPUP and ACTIVATE POPUP operate independently of @ ... MENU and READ MENU.

Clauses **<row, column>**
<row, column> are numeric expressions that specify where the top-left corner of the menu popup is placed on the desktop in FoxPro for MS-DOS, the main FoxPro window in FoxPro for Windows, or a user-defined window.

<array>
<array> is a one-dimensional array containing the options that appear on the menu popup. Menu options must be character type.

<expN1>
The numeric expression **<expN1>** is the total number of menu options available on the menu popup. If the array has 10 elements and the value of **<expN1>** is 5, the first five elements of the array are available on the menu popup.

<expN2>

The numeric expression <expN2> is the number of menu options displayed on the menu popup. If <expN1> is greater than <expN2>, <expN2> determines how many options are displayed on the popup. If there are too many options to fit on the menu popup, you can scroll through them using the keyboard or the mouse.

TITLE <expC>

The character expression <expC> is the title displayed in the top border of the menu popup.

SHADOW

Behind a menu popup defined with SHADOW is a darkened area that resembles a shadow. Text or objects covered by the shadow are still visible. By default, popup menus don't have shadows.

Example

```
SET TALK OFF
CLEAR
DIMENSION choices(3)
STORE 'Chicken' TO choices(1)
STORE 'Beef' TO choices(2)
STORE 'Fish' TO choices(3)
STORE 0 TO mchoice
@ 5,0 MENU choices,3 TITLE 'Dinner Choices'
READ MENU TO mchoice

DO CASE
  CASE mchoice = 1
    @ 10,0 SAY 'You will be having chicken for dinner.'
  CASE mchoice = 2
    @ 10,0 SAY 'You will be having beef for dinner.'
  CASE mchoice = 3
    @ 10,0 SAY 'You will be having fish for dinner.'
ENDCASE
```

@ ... PROMPT

Purpose	Creates a menu bar
Syntax	@ <row, column> PROMPT <expC1> [MESSAGE <expC2>]
Remarks	Included for backward compatibility — use the Menu Builder to create menus and menu popups
See Also	CREATE MENU, MENU TO, SET CONFIRM, SET MESSAGE

Description Use @ ... PROMPT to create a menu bar. Using a series of @ ... PROMPTS you can create the menu bar pads and specify their positions on the desktop (FoxPro for MS-DOS), the main FoxPro window (FoxPro for Windows), or in a user-defined window.

After the menu bar pads are created, issue MENU TO to activate the menu bar.

When you choose a menu bar pad, a number is returned to a memory variable or array element specified in MENU TO. The number returned is determined by the order in which the pads are defined. For example, if five pads are defined with five @ ... PROMPTS and you choose the third pad, 3 is stored to the memory variable or array element.

To choose a menu bar pad, press Enter or the Spacebar when the desired pad is selected, or click the pad.

A menu pad is disabled by placing a backslash (\) before the pad prompt. A disabled menu pad appears on the menu bar but cannot be selected. You can also create a hot key for a menu pad by placing a backslash and a less-than sign (<) before the character in the prompt that you would like to be the hot key.

If a prompt contains more than one occurrence of a character, the first occurrence of that character is the hot key. In the following example, the second D in ADD is specified as the hot key, but the first D is the hot key.

```
@ 1,20 PROMPT 'AD\<D' MESSAGE 'Add a record'
```

If CONFIRM is set on with SET CONFIRM, you can select a menu bar pad by pressing the key corresponding to the first character of the prompt. Once a menu bar pad is selected, press Enter or the Spacebar to choose the pad. If hot keys have been assigned to the pads, you can choose a pad by pressing its hot key. Hot keys aren't affected by SET CONFIRM.

If CONFIRM is set off, pressing the key corresponding to the first character of the prompt of a menu bar pad chooses the pad. If menu bar pads have been defined with hot keys other than the first character of the menu bar pads, the hot keys take precedence.

Clauses

<row, column>

The <row, column> coordinates specify where the menu bar pads are placed. If <row> is the same for each @ ... PROMPT, a horizontal menu bar is created. If <column> is the same for each @ ... PROMPT command, a vertical menu bar is created.

<expC1>

The character expression <expC1> is the prompt that appears in the menu bar pad.

MESSAGE <expC2>

The character expression <expC2> is displayed when the menu bar is activated with MENU TO.

Example

```
SET TALK OFF
CLEAR
SET MESSAGE TO 24 CENTER
@ 1,20 PROMPT '\<ADD' MESSAGE 'Add a record to the table'
@ 1,30 PROMPT '\<DELETE' MESSAGE;
      'Delete a record from the table'
@ 1,40 PROMPT '\<UPDATE' MESSAGE 'Update records in the table'
@ 1,50 PROMPT '\<QUIT' MESSAGE 'Exit this program'
STORE 1 TO mchoice
MENU TO mchoice

DO CASE
  CASE mchoice= 1
    @ 2,0 SAY 'Menu pad # ' + STR(mchoice,1) + ', ADD'
  CASE mchoice= 2
    @ 2,0 SAY 'Menu pad # ' + STR(mchoice,1) + ', DELETE'
  CASE mchoice = 3
    @ 2,0 SAY 'Menu pad # ' + STR(mchoice,1) + ', UPDATE'
  CASE mchoice = 4
    @ 2,0 SAY 'Menu pad # ' + STR(mchoice,1) + ', QUIT'
ENDCASE
```

@ ... TO

Purpose	Draws a box, circle, or ellipse using specified coordinates
Syntax	<code>@ <row1, column1> TO <row2, column2></code> <code>[DOUBLE PANEL <border string>]</code> <code>[PATTERN <expN1>]</code> <code>[PEN <expN2> [, <expN3>]]</code> <code>[STYLE <expC>]</code> <code>[COLOR SCHEME <expN4></code> <code> COLOR <color pair list></code>
Remarks	PATTERN, PEN, STYLE clauses and RGB color pairs — FoxPro for Windows only
See Also	@ ... BOX, SET COLOR

Description Use this command to draw a box. If you omit the optional clauses, the box is drawn with a single line for the border and the current colors.

In FoxPro for Windows, you can draw rectangles, rounded rectangles, circles and ellipses by including the PATTERN, PEN and STYLE clauses. These clauses are optional and can be included in any combination. If a PATTERN, PEN or STYLE clause is included with a DOUBLE or PANEL clause, the DOUBLE or PANEL clause is ignored.

The PATTERN clause specifies a fill pattern for drawn objects. The PEN clause specifies the width and type of an object's outline. STYLE lets you create a transparent or opaque object and specify the curvature for rectangle edges, circles and ellipses.



In FoxPro for Windows, if FoxFont is the current font and the PATTERN, PEN and STYLE clauses aren't included, rectangles are drawn with the FoxFont box drawing characters (┌,└,┐,┘ ...). This provides compatibility with earlier versions of FoxPro.

If FoxFont isn't the current font, or if FoxFont is the current font and a PATTERN, PEN or STYLE clause is included, rectangles are drawn using Windows-style graphics. A rectangle drawn with the FoxFont box-drawing characters and the same rectangle drawn with Windows graphics *are drawn at slightly different positions*.

Clauses

<row1, column1>

<row2, column2>

<row1, column1> are the coordinates of the upper-left corner of the box. <row2, column2> are the coordinates of the lower-right corner of the box. If <row1> and <row2> are the same, a horizontal line is drawn. If <column1> and <column2> are the same, a vertical line is drawn.

DOUBLE | PANEL | <border string>

If DOUBLE is included, the box is drawn with a double-line border. If PANEL is included, the box is drawn with a solid border.

<border string> is a set of characters that specify parts of the box in this order: top, bottom, left side, right side, upper-left corner, upper-right corner, lower-left corner, lower-right corner.

Specify the characters in the border string using one of the following forms and separate them with commas:

- Literal characters enclosed in string delimiters (single or double quotation marks or brackets).

```
@ 1,10 TO 22,45 '=' , '=' , '|' , '|' , '+' , '+' , '+' , '+'
```

- Character expressions or variables delimited with commas.

```
STORE '=' TO END
```

```
STORE '|' TO SIDE
```

```
STORE '+' TO CORNER
```

```
@ 1,10 TO 22,45 END,END,SIDE,SIDE,CORNER,CORNER,CORNER,CORNER
```

- The ASCII value of the characters specified in CHR() functions.

```
STORE CHR(61) TO END
STORE CHR(124) TO SIDE
STORE CHR(43) TO CORNER
@ 1,10 TO 22,45 END,END,SIDE,SIDE,CORNER,CORNER,CORNER,CORNER

***Or***

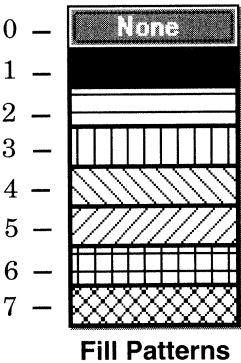
@ 1,10 TO 22,45 CHR(61),CHR(61),CHR(124),CHR(124),;
CHR(43),CHR(43),CHR(43),CHR(43)
```

You can draw the box with a single character by including the top character and omitting the remaining characters. This example draws a box using asterisks:

```
STORE '*' TO BOX_CHAR
@ 1,10 TO 22,45 BOX_CHAR
```

PATTERN <expN1> — FoxPro for Windows only

PATTERN accepts a numeric argument from 0 through 7 that specifies the pattern that fills the object. This numeric argument corresponds to the fill patterns available for an object with the Screen Builder.



PEN <expN2> [, <expN3>] — FoxPro for Windows only

Include the PEN clause to specify an object's outline. The width of the pen used to draw the outline is specified with <expN2>, and the pen type is specified with <expN3>.

The following table lists values for <expN2> and <expN3> and the corresponding pen widths and types.

<expN2>	Pen Width	<expN3>	Pen Type
0	Hairline* (default)	0	None
1	1 point	1	Dotted
2	2 point	2	Dashed
3	3 point	3	Dash-dot
4	4 point	4	Dash-dot-dot
5	5 point		
6	6 point		

* Lines drawn with the hairline and 1-point pens may display as 1 point wide with some display drivers and hardware.

STYLE <expC> — FoxPro for Windows only

The type of object created (rectangle, rounded rectangle, circle or ellipse) is specified by <expC>. Objects created with @ ... TO are opaque by default; include T in <expC> to create a transparent object.

Include a number from 0 to 99 in <expC> to specify the curvature of the object's corner: 0 specifies no curvature, and creates square corners; 99 specifies the maximum curvature, and creates circles and ellipses.

COLOR SCHEME <expN4>

I COLOR <color pair list>

The color of the object you create with @ ... FILL can be specified by including the number of an existing color scheme in the COLOR SCHEME clause or a set of color pairs in the COLOR clause.

A color scheme is a set of 10 predefined color pairs. The color pairs in a color scheme can be changed with SET COLOR OF SCHEME. In FoxPro for MS-DOS the color pairs in a color scheme can also be changed in the Color Picker.

A color pair is a set of two letters separated by a forward slash. The first color letter specifies the foreground color and the second letter specifies the background color.

For example, this color pair specifies a red foreground on a white background:

R/W

For a list of colors and their corresponding color letters, see the SET COLOR commands section in this chapter or the Color Table by Color Pair topic in the online help facility.

In FoxPro for Windows, a color pair can also be specified with a set of 6 RGB (Red Green Blue) color values separated by commas. The first 3 color values specify the foreground color and the second 3 color values specify the background color. The color values can range from 0 through 255.

In FoxPro for Windows, the R/W color pair in the example above can also be specified with this RGB color pair:

```
RGB(255,0,0,255,255,255)
```

Only the first color pair in the specified color scheme or color pair list determines the color of the object.

In FoxPro for Windows, the first color in the color pair is the pen color, the second color is the fill color. For example, the following creates an ellipse with a solid red interior and a blue outline:

```
@ 2,2 TO 12,22 STYLE '99' PATTERN 1 PEN 2 COLOR 'B/R'
```

If you omit the color clause, the rectangular portion is cleared. An area can also be cleared with @ ... CLEAR.

ABS()

Purpose	Returns the absolute value of the specified numeric expression
Syntax	ABS(<expN>)
Returns	Numeric
See Also	INT(), ROUND(), SIGN()

Description ABS() returns the absolute value of a numeric expression.

Parameter <expN>
<expN> is the numeric expression whose absolute value is returned by ABS().

Examples

```
? ABS(-45)
45
? ABS(10-30)
20
? ABS(30-10)
20
STORE 40 TO num1
STORE 2 TO num2
? ABS(num2-num1)
38
```

ACCEPT

Purpose	Accepts character string data from the screen
Syntax	ACCEPT [<expC>] TO <memvar>
Remarks	Included for backward compatibility — use @ ... GET instead
See Also	@ ... GET, READ, WAIT

Description This command allows you to enter character data directly into a memory variable or array element without delimiting the characters with quotation marks.

If you need to perform data validation or trap errors, @ ... GET is a better choice.

ACCEPT differs from INPUT in two ways. With ACCEPT:

- The data you enter is always treated as character type.
- You don't have to enclose the entered data in quotation marks.

Clauses **<expC>**

The character expression <expC> is the prompt text that is displayed next to the area in which data is entered.

<memvar>

The memory variable or array element in which the character data is stored is specified with <memvar>. If the memory variable or array element isn't defined, it is automatically created by ACCEPT.

If you press Enter without entering data, the memory variable or array element contains the null string. If you press Escape when SET ESCAPE is OFF, the memory variable contains the null string. If you press Escape when SET ESCAPE is ON, program execution is suspended.

Example This example prompts you for a customer name and displays the memory variable containing the name you entered.

```
ACCEPT 'ENTER THE CUSTOMER NAME: ' TO mcustname
ENTER THE CUSTOMER NAME: Fred Frederick
? mcustname
Fred Frederick
```

ACOPY()

Purpose	Copies elements from one array to another array
Syntax	ACOPY(<array1>, <array2> [, <expN1> [, <expN2> [, <expN3>]]])
Parameters	<array1> The source array <array2> The destination array <expN1> The first element in the source array to be copied <expN2> The number of elements to copy <expN3> The first element in the destination array to be copied
Returns	Numeric
See Also	ADEL(), AELEMENT(), AINS(), ASCAN(), ASORT(), DIMENSION

Description This function copies a series of elements from one array to another and returns the number of elements copied to the destination array.

Parameters **<array1>, <array2>**
Elements from the source array <array1> are copied one-to-one to elements of the destination array <array2>. The elements in the destination array are replaced by those copied from the source array.

The arrays can be one- or two-dimensional. If the destination array doesn't exist, FoxPro automatically creates it. In such a case, the size of the destination array will be the same as that of the source array.



You can refer to an element in a two-dimensional memory variable array in two ways: The first uses two subscripts to specify the row and column position of the element in the array; the other uses a single-element number. This function and others that manipulate two-dimensional arrays require single-element numbers (here, <expN1> and <expN3>).

Use AELEMENT() to return the proper element number for a two-dimensional array from its row and column subscripts.

<expN1>
The optional numeric expression <expN1> specifies the first element number in the source array to be copied and is inclusive (element number <expN1> is included in the copying). If <expN1> isn't included, copying begins with the first element in the source array.

<expN2>

You can specify the number of elements copied from the source array by including the optional numeric expression <expN2>. If <expN2> is -1, all elements of the source array beginning with element <expN1> are copied.

<expN3>

You can specify the first element in the *destination* array to be replaced with <expN3>.

Example

The following example creates an array from selected records in the CUSTOMER table and then uses ACOPY to copy the array.

```
CLOSE DATABASES
SELECT DISTINCT company ;
      FROM customer ;
      ORDER BY company ;
      WHERE state = 'WA';
      INTO ARRAY companies
= ACOPY(companies, temp_comp)  && Make a copy of the companies array

DISPLAY MEMORY LIKE *
```

ACOS()

Purpose	Returns the arc cosine of a specified numeric expression
Syntax	ACOS(<expN>)
Returns	Numeric
See Also	ASIN(), ATAN(), ATN2(), COS(), DTOR(), RTOD(), SET DECIMALS, SIN(), TAN()

Description This trigonometric function returns the arc cosine of a numeric expression in radians.

Parameter **<expN>**
The arc cosine of <expN> is returned by ACOS(). The value of <expN> can range from +1 through -1. The value returned by ACOS() ranges from 0 through π (≈ 3.141592). The number of decimal places ACOS() returns is determined by SET DECIMALS.
Use RTOD() to convert radians to degrees.

Examples

```
? RTOD(ACOS(0))  
90.00  
STORE -1 to arcangle  
? RTOD(ACOS(arcangle))  
180.00  
? RTOD(ACOS(sqrt(2)/2))  
45.00
```

ACTIVATE MENU

Purpose	Displays and activates a menu bar
Syntax	ACTIVATE MENU <menu name> [NOWAIT] [PAD <pad name>]
See Also	CLEAR MENU, CREATE MENU, DEACTIVATE MENU, DEFINE MENU, DEFINE PAD, HIDE MENU, SET SYSMENU, SHOW MENU

Description This command works in conjunction with DEFINE MENU and DEFINE PAD. ACTIVATE MENU displays and activates the menu bar specified with <menu name>.

Navigating in a Menu Bar

When the menu bar is activated, you can move from menu pad to menu pad by pressing the Right and Left Arrow keys. If SET CONFIRM is ON, you can also move across the menu bar by pressing the key corresponding to the first character of a menu bar pad.

Choosing an Option

You can choose a pad on a menu bar by selecting it and pressing Enter or the Spacebar, or by clicking the pad. If hot keys are assigned to the menu bar pads, you can choose a pad by pressing the corresponding hot key. If no hot keys are defined and SET CONFIRM is OFF, you can choose a menu bar pad by pressing the key corresponding to the first character of the menu bar pad.

Press Escape to deactivate the menu bar without making a choice. The menu bar can also be deactivated by issuing DEACTIVATE MENU or by activating another menu bar.



When you include the FoxPro system menu bar (_MSYSMENU) in an application, there is no need to activate the menu. Instead, issue SET SYSMENU AUTOMATIC. For more information on using the FoxPro system menu bar, see the Menu Builder chapter in the FoxPro *User's Guide*.

Clauses <menu name>

Specify the name of the menu to activate with <menu name>.

NOWAIT

When a menu bar is displayed and activated, program execution pauses until you choose a menu pad or press Escape. Including **NOWAIT** continues program execution after the menu bar is displayed and activated. The program doesn't wait for a choice to be made but continues executing. You can choose a pad from the menu bar whenever the program is waiting for input from the keyboard.

If you want a menu bar to be active during a **READ** command, include **NOWAIT** when you activate the menu. Activate the menu before you issue **READ**.



A menu activated with the **NOWAIT** option doesn't return program execution to the line following the **ACTIVATE MENU** command when **DEACTIVATE MENU** is issued.

PAD <pad name>

When a menu bar is activated, the first menu pad is selected by default. If you include **PAD <pad name>** in **ACTIVATE MENU**, the menu pad specified with **<pad name>** is automatically selected when the menu bar is activated.

Example

This example uses **ACTIVATE MENU** to display and activate a user-defined menu system. The current system menu bar is first saved to memory with **SET SYSMENU SAVE**, and then all system menu pads are removed with **SET SYSMENU TO**.

Two menu pads are created with **DEFINE PAD**, and **DEFINE POPUP** is used to create a menu popup for each menu pad. **DEFINE BAR** is used to create options on each of the menu popups. When a menu pad is chosen, **ON PAD** uses **ACTIVATE POPUP** to activate the corresponding menu popup. **ACTIVATE MENU** displays and activates the menu bar.

When an option is chosen from a menu popup, the **CHOICE** procedure is executed. **CHOICE** displays the name of the chosen option and the name of the menu popup containing the option.

```
*** Name this program ACTIMENU.PRG ***
CLEAR
SET SYSMENU SAVE
SET SYSMENU TO
ON KEY LABEL ESC KEYBOARD CHR(13)
DEFINE MENU example BAR AT LINE 1
```

```
DEFINE PAD convpad OF example PROMPT '\<Conversions' COLOR SCHEME 3 ;
KEY ALT+C, ''
```

ACTIVATE MENU

```
DEFINE PAD cardpad OF example PROMPT 'Card \<Info' COLOR SCHEME 3 ;
    KEY ALT+I, ''
ON PAD convpad OF example ACTIVATE POPUP conversion
ON PAD cardpad OF example ACTIVATE POPUP cardinfo
```

```
DEFINE POPUP conversion MARGIN RELATIVE SHADOW COLOR SCHEME 4
DEFINE BAR 1 OF conversion PROMPT 'Ar\<ea' ;
    KEY CTRL+E, '^E'
DEFINE BAR 2 OF conversion PROMPT '\<Length' ;
    KEY CTRL+L, '^L'
DEFINE BAR 3 OF conversion PROMPT 'Ma\<ss' ;
    KEY CTRL+S, '^S'
DEFINE BAR 4 OF conversion PROMPT 'Spee\<d' ;
    KEY CTRL+D, '^D'
DEFINE BAR 5 OF conversion PROMPT '\<Temperature' ;
    KEY CTRL+T, '^T'
DEFINE BAR 6 OF conversion PROMPT 'T\<ime' ;
    KEY CTRL+I, '^I'
DEFINE BAR 7 OF conversion PROMPT 'Volu\<me' ;
    KEY CTRL+M, '^M'
```

```
ON SELECTION POPUP conversion;
    DO choice IN actimenu WITH PROMPT( ), POPUP( )
```

```
DEFINE POPUP cardinfo MARGIN RELATIVE SHADOW COLOR SCHEME 4
DEFINE BAR 1 OF cardinfo PROMPT '\<View Charges' ;
    KEY ALT+V, ''
DEFINE BAR 2 OF cardinfo PROMPT 'View \<Payments' ;
    KEY ALT+P, ''
DEFINE BAR 3 OF cardinfo PROMPT 'Vie\<w Users' ;
    KEY ALT+W, ''
DEFINE BAR 4 OF cardinfo PROMPT '\<- '
DEFINE BAR 5 OF cardinfo PROMPT '\<Charges '
ON SELECTION POPUP cardinfo;
    DO choice IN actimenu WITH PROMPT( ), POPUP( )
```

```
ACTIVATE MENU example
DEACTIVATE MENU example
RELEASE MENU example EXTENDED
SET SYSMENU TO DEFAULT
ON KEY LABEL ESC
```

```
PROCEDURE choice
PARAMETERS mprompt, mpopup
WAIT WINDOW 'You chose ' + mprompt + ;
    ' from popup ' + mpopup NOWAIT
```


ACTIVATE POPUP

Purpose	Displays and activates a popup created with DEFINE POPUP, or a FoxPro menu system popup
Syntax	ACTIVATE POPUP <popup name> [AT <row, column>] [BAR <expN>] [NOWAIT] [REST]
See Also	CLEAR POPUPS, CREATE MENU, DEACTIVATE POPUP, DEFINE BAR, DEFINE POPUP, HIDE POPUP, MOVE POPUP, ON PAD, ON SELECTION PAD, ON SELECTION POPUP, POP POPUP, PUSH POPUP, SHOW POPUP

Description ACTIVATE POPUP works in conjunction with the commands DEFINE POPUP and DEFINE BAR. DEFINE POPUP creates a popup, and DEFINE BAR creates the individual options on the popup. ACTIVATE POPUP displays and activates the popup.

Navigating in a Popup

When the popup is activated, you can move between the popup options by pressing the Up and Down Arrow keys. If SET CONFIRM is ON, you can select a popup option by pressing the key corresponding to the first character in the name of the option.

Choosing an Option

You can choose an option on a popup by selecting it and pressing Enter or the Spacebar, or by clicking the option. If hot keys are assigned to the menu options, you can choose a popup option by pressing the corresponding hot key. If SET CONFIRM is OFF, you can choose a popup option by pressing the key corresponding to the first character in the name of the popup option.

Press Escape to deactivate the popup without making a choice. The popup can also be deactivated by issuing DEACTIVATE POPUP or by activating another popup.

Clauses <popup name>
Specify the name of the popup to activate with <popup name>.

AT <row, column>

You can specify where the popup is activated by including AT <row, column>. The upper-left corner of the popup is positioned at the coordinate specified by <row, column>. The position specified by this clause takes precedence over a position specified by the FROM clause in DEFINE POPUP.

BAR <expN>

If BAR <expN> is included, popup option number <expN> is automatically selected when the popup is displayed. For example, if <expN> is 2, the second option is selected. The first option is selected if BAR <expN> is omitted or if <expN> is greater than the number of options in the popup.

NOWAIT

When ACTIVATE POPUP is issued from within a program, the popup is displayed and activated. Program execution pauses until an option is chosen or Escape is pressed. Including NOWAIT causes program execution to continue after the popup is displayed and activated. The program doesn't wait for a choice to be made. You can choose an option from the popup whenever the program is waiting for input from the keyboard.

REST

A popup created with the PROMPT FIELD clause of DEFINE POPUP places records from a field into the popup. When the popup is activated, the first option in the popup is initially selected, even if the record pointer in the table is positioned on a record other than the first record.

To change this default behavior and to initially select the popup option corresponding to the current record pointer position in the table, include the REST key word.

Example

This example uses ACTIVATE POPUP with ON PAD to activate a popup when a menu pad is chosen. The current system menu bar is first saved to memory with SET SYSMENU SAVE, and then all system menu pads are removed with SET SYSMENU TO.

Two new system menu pads are created with DEFINE PAD, and DEFINE POPUP is used to create a popup for each menu pad. DEFINE BAR is used to create options on each of the popups. When a menu pad is chosen, ON PAD uses ACTIVATE POPUP to activate the corresponding popup.

When an option is chosen from a popup, the CHOICE procedure is executed. CHOICE displays the name of the chosen option and the name of the popup containing the option. If the **Exit** option in the **Card Info** popup is chosen, the original FoxPro system menu is restored.

```
*** Name this program ACTIPOP.PRG ***
```

```
CLEAR
```

```
SET SYSMENU SAVE
```

```
SET SYSMENU TO
```

```
DEFINE PAD convpad OF _MSYSMENU PROMPT '\<Conversions' COLOR SCHEME 3 ;  
    KEY ALT+C, ''
```

```
DEFINE PAD cardpad OF _MSYSMENU PROMPT 'Card \<Info' COLOR SCHEME 3 ;  
    KEY ALT+I, ''
```

```
ON PAD convpad OF _MSYSMENU ACTIVATE POPUP conversion
```

```
ON PAD cardpad OF _MSYSMENU ACTIVATE POPUP cardinfo
```

```
DEFINE POPUP conversion MARGIN RELATIVE SHADOW COLOR SCHEME 4
```

```
DEFINE BAR 1 OF conversion PROMPT 'Ar\<ea' ;
```

```
    KEY CTRL+E, '^E'
```

```
DEFINE BAR 2 OF conversion PROMPT '\<Length' ;
```

```
    KEY CTRL+L, '^L'
```

```
DEFINE BAR 3 OF conversion PROMPT 'Ma\<ss' ;
```

```
    KEY CTRL+S, '^S'
```

```
DEFINE BAR 4 OF conversion PROMPT 'Spee\<d' ;
```

```
    KEY CTRL+D, '^D'
```

```
DEFINE BAR 5 OF conversion PROMPT '\<Temperature' ;
```

```
    KEY CTRL+T, '^T'
```

```
DEFINE BAR 6 OF conversion PROMPT 'T\<ime' ;
```

```
    KEY CTRL+I, '^I'
```

```
DEFINE BAR 7 OF conversion PROMPT 'Volu\<me' ;
```

```
    KEY CTRL+M, '^M'
```

```
ON SELECTION POPUP conversion;
```

```
    DO choice IN actipop WITH PROMPT( ), POPUP( )
```

ACTIVATE POPUP

```
DEFINE POPUP cardinfo MARGIN RELATIVE SHADOW COLOR SCHEME 4
DEFINE BAR 1 OF cardinfo PROMPT '\<View Charges' ;
    KEY ALT+V, ''
DEFINE BAR 2 OF cardinfo PROMPT 'View \<Payments' ;
    KEY ALT+P, ''
DEFINE BAR 3 OF cardinfo PROMPT 'Vie\<w Users' ;
    KEY ALT+W, ''
DEFINE BAR 4 OF cardinfo PROMPT '\-'
DEFINE BAR 5 OF cardinfo PROMPT '\<Charges '
DEFINE BAR 6 OF cardinfo PROMPT '\-'
DEFINE BAR 7 OF cardinfo PROMPT 'E\<xit '
ON SELECTION POPUP cardinfo;
    DO choice IN actipop WITH PROMPT( ), POPUP( )

PROCEDURE choice
PARAMETERS mprompt, mpopup

WAIT WINDOW 'You chose ' + mprompt + ;
    ' from popup ' + mpopup NOWAIT

IF mprompt = 'Exit'
    SET SYSMENU TO DEFAULT
ENDIF
```

ACTIVATE SCREEN

Purpose	Sends all subsequent output to the screen instead of the current window
Syntax	ACTIVATE SCREEN
See Also	ACTIVATE WINDOW, DEACTIVATE WINDOW, DEFINE WINDOW, HIDE WINDOW, SHOW WINDOW

Description ACTIVATE SCREEN directs output to the desktop in FoxPro for MS-DOS or to the main FoxPro window in FoxPro for Windows instead of to the active user-defined window.

To direct output to a user-defined window, use ACTIVATE WINDOW to activate the window.



To view output on the desktop or a window that is obscured by a window, press Alt+Ctrl+Shift. Any windows on top of the desktop, the main FoxPro window or an active user-defined window are hidden until the keys are released.

ACTIVATE WINDOW

Purpose	Displays and activates one or more user-defined windows or FoxPro system windows
Syntax	ACTIVATE WINDOW [<window name1>[, <window name2> ...]] IACTIVATE WINDOW ALL [IN [WINDOW] <window name3> SCREEN] [BOTTOM TOP SAME] [NOSHOW]
See Also	CLEAR WINDOW, DEACTIVATE WINDOW, DEFINE WINDOW, HIDE WINDOW, RELEASE WINDOW, SHOW WINDOW

Description This command displays and activates one or more user-defined windows or FoxPro system windows. User-defined windows are created with DEFINE WINDOW.

Activating a window makes it the frontmost window and directs all output to that window. Output can be directed to only one window at a time. A window remains the active output window until it is deactivated or released, or until another window or the desktop is activated.

The names of user-defined windows appear in the bottom section of the **Window** menu popup. The name of the active user-defined window is marked with a diamond in FoxPro for MS-DOS and a check mark in FoxPro for Windows.

More than one window can be placed on the desktop or the main FoxPro window at a time, but output is directed only to the last window activated. When more than one window is open, deactivating the active output window removes it from the screen and sends subsequent output to another window. If there is no active output window, output is directed to the desktop or to the main FoxPro window.



To ensure that output is directed to a specific window when you deactivate the active output window, you must explicitly activate the window you want to send output to with ACTIVATE WINDOW.

All activated windows are displayed until DEACTIVATE WINDOW or HIDE WINDOW is issued to remove them from view. Issuing either command removes windows from view but not from memory. Windows can be redisplayed by issuing ACTIVATE WINDOW or SHOW WINDOW.

To remove windows from view and from memory, use CLEAR WINDOWS, RELEASE WINDOWS or CLEAR ALL. Windows that are removed from memory must be redefined to place them back on the desktop or the main FoxPro window.

Clauses

<window name1> [, <window name2> ...]

This clause lists the names of windows to activate. Separate the window names with a comma.

ALL

If you specify ALL, all defined windows are activated. The last window activated is the active output window.

IN [WINDOW] <window name3>

When you include this clause, the window is placed and activated in the parent window specified with <window name3>. The activated window is referred to as a *child window*. A parent window can have multiple child windows. A child window activated inside a parent window cannot be moved outside the parent window. If the parent window is moved, the child window moves with it.



The parent window must be visible for any of its child windows to be visible.

IN [WINDOW] SCREEN

By default, windows are activated on the desktop or the main FoxPro window rather than in a parent window. You can explicitly place and activate a window on the desktop or the main FoxPro window by including IN WINDOW SCREEN or IN SCREEN. If an IN WINDOW clause was included in DEFINE WINDOW when a window was created, the IN SCREEN clause takes precedence and the window is placed on the desktop or the main FoxPro window.

ACTIVATE WINDOW

You can use the ACTIVATE WINDOW to place FoxPro desk accessories and system windows on the desktop, the main FoxPro window or in a parent window. The following desk accessories and system windows can be opened with ACTIVATE WINDOW:

Desk Accessories	System Windows
Filer	Command
Calculator	Debug
Calendar/Diary	Trace
Special Characters*	View
ASCII Chart*	
Puzzle	

*Not available in FoxPro for Windows

To activate a desk accessory whose name has two parts (Calendar/Diary, Special Characters, ASCII Chart), include the first part of the name in upper case. For example, use ACTIVATE WINDOW CALENDAR to activate the Calendar/Diary.

Use RELEASE WINDOW to remove a desk accessory or a system window from the desktop or a window.

BOTTOM | TOP | SAME

By default, a window becomes the frontmost window when it is activated. Include BOTTOM, TOP or SAME to specify where windows are activated with respect to other previously activated windows. Including BOTTOM places a window *behind* all other windows; including TOP places it in *front* of all other windows. Issuing ACTIVATE WINDOW with SAME activates a window without affecting its front to back placement.

NOSHOW

Include NOSHOW to activate and direct output to a window without displaying the window.

In FoxPro version 2.0 and later, including NOSHOW automatically displays the window when it becomes the current READ window. In earlier FoxPro versions, the window is *not* displayed.



Activate a window using NOSHOW to draw an input window. Once the window is drawn, issue SHOW WINDOW to “pop” the window open. This technique displays windows quickly.

Example

This example defines a window named Output and activates it, placing it on the desktop or main FoxPro window. The WAIT command pauses execution, the window is hidden and then redisplayed.

```
CLEAR
DEFINE WINDOW output FROM 2,1 TO 13,75 TITLE 'Output' ;
    CLOSE FLOAT GROW SHADOW ZOOM

ACTIVATE WINDOW output
WAIT WINDOW 'press any key to hide window output'
HIDE WINDOW output
WAIT WINDOW 'press any key to show window output'
SHOW WINDOW output
WAIT WINDOW 'press any key to release window output'
RELEASE WINDOW output
```

ADEL()

Purpose	Deletes an element, row or column from an array
Syntax	ADEL(<array>, <expN> [, 2])
Parameters	<array> <expN> 2 The array from which data is to be deleted The number of the element, row or column to delete Indicates that <expN> is a column
Returns	Numeric
See Also	ACOPY(), ADIR(), AELEMENT(), AFIELDS(), AINS(), ALLEN(), ASCAN(), ASORT(), ASUBSCRIPT(), DIMENSION

Description This function deletes an element from a one-dimensional array or a row or column from a two-dimensional array. Deleting an element, row or column from an array doesn't change the size of the array instead; the trailing elements, rows or columns are moved to the start of the array, and the last element, row or column in the array is set to a logical false (.F.).

If the element, row or column is successfully deleted, 1 is returned.



You can refer to an element in a two-dimensional array in two ways. The first uses two subscripts to specify the row and column position of the element in the array; the other uses a single-element number. This function and others that manipulate two-dimensional arrays require single-element numbers (here, the numeric expression <expN>). You can use AELEMENT() to return the proper element number for a two-dimensional array from its row and column subscripts.

For more information on how to reference elements in an array, see DIMENSION in this chapter.

Parameters <array>, <expN>

To delete an element from a one-dimensional array, specify the array's name with <array> and the element's number with <expN>. To delete a row from a two-dimensional array, specify the array's name with <array> and the row's number with <expN>.

2

To delete a column from a two-dimensional array, include this argument.

Example The following example creates and fills an array, then searches for a particular company name, which, if the company is found, it removes from the array.

```

CLOSE DATABASES
SELECT company FROM customer ;
    WHERE state = 'TN' ;
    INTO ARRAY companies
m.count = _TALLY
m.name = 'Data Inc.'
m.pos = ASCAN(companies, m.name)  && Search for company

IF m.pos != 0

    *** Company found, remove it from the array ***

    = ADEL(companies, m.pos)
    m.count = m.count - 1
ENDIF
DISPLAY MEMORY LIKE companies*

```

ADIR()

Purpose	Places information about specified files into an array
Syntax	ADIR(<array> [, <expC1> [, <expC2>]])
Parameters	<array> The array where information about specified files is placed <expC1> A file skeleton <expC2> Expands the search to include additional files. <expC2> can be one or more of the following: D for subdirectories H for hidden files S for system files V for volume name
Returns	Numeric
See Also	ADEL(), AELEMENT(), AFIELDS(), AINS(), ALLEN(), ASCAN(), ASORT(), ASUBSCRIPT(), DIMENSION, DIRECTORY, FILER

Description ADIR() places information about all files in the current directory or all files that match a file skeleton into an array, and then returns the number of files. For each file, ADIR() places into the array the file name, size, date, time and MS-DOS attributes.

Parameters <array>
Information about the files is placed into the memory variable array specified in <array>. If the array you include doesn't exist, FoxPro automatically creates the array. If the array exists and isn't large enough to contain all the information, the size of the array is automatically increased to accommodate the information. If the array is larger than necessary, the array size is truncated. If the array exists and ADIR() returns 0 because no matching files or directories were found, the array remains unchanged.

The following table describes what each column in the array contains and the data type of each column:

Array Column Information and Type		
Column	Information	Data Type
1	File names	Character
2	File sizes	Numeric
3	File dates	Date
4	File times	Character
5	File attributes	Character

The last array column contains the MS-DOS attributes of the matching files. Each attribute is expressed by a letter, and a file can have more than one attribute. The following table indicates what attribute each letter represents.

MS-DOS Attribute Letters	
Letter	Attribute
A	Archive (Read and Write)
H	Hidden
R	Read only
S	System
D	Directory

<expC1>

You can include a file skeleton <expC1> that lets you store information about files with names or extensions that match a criterion. For example, the criterion can be all tables, all text files, all files with names that have A as their second letter and so on. These general searches are done by including the wildcard characters * and ? in <expC1>. A question mark represents a single character and an asterisk represents any number of characters. You can use any number of wild card characters in any position within the file skeleton.

A drive and/or directory specification can be included in <expC1> to search for matching file names on a specific drive and directory. If a drive and directory specification is omitted, information about files in the current directory is placed into the array.

<expC2>

Include <expC2> to expand the search to subdirectory names, hidden or system files, or a volume name.

The character expression <expC2> can contain any combination of D, H, and S. Including D returns subdirectory names in addition to file names that match the file skeleton specified in <expC1>. Including H returns information about hidden files that match the file skeleton specified in <expC1>. Including S returns information about system files that match the file skeleton specified in <expC1>.

You can include V in <expC2> to return the volume name of the current drive. Only the volume name is returned to the array if V is included with D, H, and S. The volume name is stored in the first array element and the remainder of the array is truncated.

Include the null string in <expC1> to return just subdirectory names, hidden files or system files.

Examples

This procedure returns the volume label when you supply the drive letter:

```
PROCEDURE volume
PARAMETER drive
string = drive + ':'
DIMENSION temparray(1)
=ADIR(temparray, (string), 'V')
RETURN temparray(1)
```

The following example shows how to call the procedure above:

```
? VOLUME('C')
CDISK
```

The following table shows some sample ADIR() arguments and their effects on file searches:

Arguments	Files Searched
ARRAY1, '*.DBF'	All tables in the current directory
ARRAY1, 'A*.DBF'	All tables in the current directory that begin with the letter A
ARRAY1, '?A*.DBF'	All tables in the current directory with A as the second letter
ARRAY1, '*A?.DBF'	All tables in the current directory with A as the second to the last letter
ARRAY1, '?.*'	All single-letter files in the current directory with or without an extension
ARRAY1, ', 'H'	All hidden files in the current directory
ARRAY1, '*.DBF', 'H'	All tables in the current directory, including hidden ones
ARRAY1, ', 'D'	All subdirectories of the current directory
ARRAY1, 'C:\FOXPRO25*', 'D'	All subdirectories of the FOXPRO25 directory
ARRAY1, 'C:\FOXPROW*', 'D'	All subdirectories of the FOXPROW directory

AELEMENT()

Purpose	Returns the number of an array element from the element's subscripts
Syntax	AELEMENT(<array>, <expN1> [, <expN2>])
Parameters	<array> <expN1> <expN2> A one- or two-dimensional array The row subscript The column subscript
Returns	Numeric
See Also	ADEL(), ADIR(), AFIELDS(), AINS(), ALLEN(), ASCAN(), ASORT(), ASUBSCRIPT(), DIMENSION, DISPLAY MEMORY

Description You can reference an element in a two-dimensional array in one of two ways. The first method uses two subscripts to specify the row and column position of the element in the array and the second method uses a single element number. AELEMENT() returns the element number when supplied an element's row and column subscripts.

The FoxPro functions ADEL(), ADIR(), AFIELDS(), AINS(), ALLEN(), ASCAN(), ASORT() and ASUBSCRIPT() can manipulate two-dimensional arrays, and require that elements be referred to by their element number. AELEMENT() facilitates conversion from subscripts to an element number for use by these functions. The corresponding row and column subscripts can be returned from an element number with ASUBSCRIPT().

Example In the following example, an array with two rows and three columns is created. DISPLAY MEMORY shows the contents of the elements of the array listed in element number order.

```
DIMENSION x(2,3)
DISPLAY MEMORY LIKE x

X          Pub  A
( 1, 1)    L .F. (element number 1)
( 1, 2)    L .F. (element number 2)
( 1, 3)    L .F. (element number 3)
( 2, 1)    L .F. (element number 4)
( 2, 2)    L .F. (element number 5)
( 2, 3)    L .F. (element number 6)
```


An element can be referred by its subscripts or its element number. The commands STORE 'INVOICE' TO X(2, 1) and STORE 'INVOICE' TO X(4) both store the character string INVOICE to the same array element.

In one-dimensional arrays an element number is identical to its single row subscript. It isn't necessary to use AELEMENT() with one-dimensional arrays.

Parameters <array>

Include the name of the array whose element number you want to return.

<expN1>

Specify the row subscript with <expN1>. If the array is one-dimensional, AELEMENT() identically returns <expN1>.

<expN2>

Specify the column subscript with <expN2>. If the array is two-dimensional, include both <expN1> and <expN2>.

If you include just <expN1>, the element number is returned until <expN1> exceeds the number of rows in the array, then the error message "Subscript out of bounds" is displayed.

Example

The following example creates and fills an array then sorts and displays a list of the companies in the CUSTOMER table.

```
CLOSE DATABASES
USE customer
COUNT TO m.count
DIMENSION companies[m.count,2]
COPY TO ARRAY companies FIELD Cno,Company

*** The first three elements are not sorted ***

= ASORT(companies, AELEMENT(companies,4,2))
@ 5,30 SAY 'List of Companies:'
@ 7,30 GET m.choice ;
    FROM companies ;
    RANGE 2 ;
    SIZE 10,30 ;
    DEFAULT 1
READ
CLEAR
```

AFIELDS()

Purpose	Places table structure information into an array
Syntax	AFIELDS(<array>)
Returns	Numeric
See Also	ADEL(), AELEMENT(), ADIR(), AINS(), ALEN(), ASCAN(), ASORT(), ASUBSCRIPT(), COPY STRUCTURE TO, DIMENSION

Description AFIELDS() places information about the structure of the current table into an array and returns the number of fields in the table. AFIELDS() stores each field name, type, length, and the number of decimal places in numeric fields. You can use COPY STRUCTURE EXTENDED to place the same information into a table instead of an array.

Parameters <array>

Information about the table structure is placed into a four-column memory variable <array>. If the array you include in AFIELDS() doesn't exist, FoxPro automatically creates it. If the array exists and isn't large enough to contain all the information returned by AFIELDS(), the size of the array is automatically increased to accommodate the information.

The following table describes what each column in the array contains and the data type of the information stored in each column.

Array Column Information and Type		
Column Number	Field Information	Data Type
1	Field names	Character
2	Field types: C = Character D = Date L = Logical M = Memo N = Numeric F = Float G = General	Character
3	Field lengths	Numeric
4	Decimal places in numeric fields	Numeric

AFONT()

Purpose	Places information about available Windows fonts into an array
Syntax	AFONT(<array> [, <expC> [, <expN>]])
Parameters	<array> The array into which data is inserted <expC> Font name <expN> Font size
Returns	Logical
Remarks	FoxPro for Windows only
See Also	GETFONT(), FONTMETRIC(), TXTWIDTH(), SYSMETRIC(), WFONT()

Description AFONT() places the names of available Windows fonts into an array. AFONT() can also be used to determine available font sizes or if a font is scalable. Use GETFONT() to display a dialog containing available Windows fonts and font sizes and styles.

Parameters **<array>**

The names of available Windows fonts are placed into the memory variable array specified in <array>. If the array isn't large enough to contain all the fonts, the size of the array is automatically increased. If you specify an existing two-dimensional array, the array is redimensioned to a one-dimensional array.

If the font information is successfully placed into the array, AFONT() returns true (.T.); otherwise AFONT() returns false (.F.).

<expC>

Include <expC> to place the available font sizes for a specific font into the array.

If the font you specify in <expC> supports only discrete font sizes (8 point, 10 point, ...), the font sizes are stored to the array and AFONT() returns true (.T.). If the font you specify in <expC> is scalable (supports fractional font size values), the array has a single element containing -1 and AFONT() returns true (.T.).

If the font you specify in <expC> isn't available, the font information isn't placed into the array and AFONT() returns false (.F.).

<expN>

Include <expN> to determine if a specific font size is available for a font.

If the font size <expN> is available for the font specified in <expC>, the array has a single element containing a logical true (.T.) and AFONT() returns true (.T.). If the font size is not available for the specified font, the array has a single element containing false (.F.) and AFONT() returns false (.F.).

AINS()

Purpose	Inserts an element, row or column into an array
Syntax	AINS(<array>, <expN> [, 2])
Parameters	<array> The array into which data is inserted <expN> For one-dimensional arrays, <expN> is an element number For two-dimensional arrays, <expN> is a row number or a column number 2 Indicates that <expN> is a column number
Returns	Numeric
See Also	ACOPY(), ADEL(), ADIR(), AELEMENT(), AFIELDS(), ALLEN(), ASCAN(), ASORT(), ASUBSCRIPT(), DIMENSION

Description AINS() inserts an element into a one-dimensional array, or a row or column into a two-dimensional array. Inserting an element, row or column into an array doesn't change the size of the array. The trailing elements, rows or columns are shifted towards the end of the array and the last element, row or column in the array is dropped from the array. The newly inserted element, row or column is initialized to a logical false (.F.).

AINS() returns 1 if the element, row or column is successfully inserted.

Parameters <array>, <expN>

To insert an element into a one-dimensional array, include the array name <array>. The number of the element <expN> where the insertion occurs. The new element is inserted just before element <expN>. To insert a row into a two-dimensional array, include the array name <array> and the number of the row <expN> where the insertion occurs. The new row is inserted just before row <expN>.



You can refer to an element in a two-dimensional array in one of two ways. The first method uses two subscripts to specify the row and column position of the element in the array; the second method uses a single element number. AINS() requires an element number (<expN>). You can use AELEMENT() to return the element number from a two-dimensional array's row and column subscripts.

For more information on referencing an array element by its subscripts, refer to DIMENSION in this chapter.

2

To insert a column into a two-dimensional array, include the array name, the column number <expN> where the insertion occurs and the argument 2. The new column is inserted just before the column specified with <expN>.

Example

The following example creates and fills an array, scans it for a company name, and if the company name isn't found, inserts it.

```

CLOSE DATABASES
SELECT company FROM customer ;
    WHERE state = 'GA' ;
    INTO ARRAY companies
m.count = _TALLY
m.name = 'AAA Incorporated'

IF ASCAN(companies, m.name) = 0  && Search for company

    *** Company not found - add it ***

    DIMENSION companies[m.count+1,1]
    = AINS(companies, m.count+1)
    companies[m.count+1] = m.name
ENDIF

DISPLAY MEMORY LIKE companies

```

ALEN()

Purpose	Returns the number of elements, rows or columns in an array
Syntax	ALEN(<array> [, <expN>])
Parameters	<array> An existing array <expN> If 0, ALEN() returns the number of elements If 1, ALEN() returns the number of rows If 2, ALEN() returns the number of columns
Returns	Numeric
See Also	ADEL(), ADIR(), AELEMENT(), AFIELDS(), AINS(), ASCAN(), ASORT(), ASUBSCRIPT(), DIMENSION, STORE

Description ALEN() returns the number of elements, rows or columns in an array.

Parameters <array>

If you include only the array name <array>, ALEN() returns the number of elements in the array.

<expN>

<expN> can be 0, 1 or 2. Omitting <expN> is identical to specifying 0 as <expN>: ALEN() returns the number of elements in the array.

If you specify 1 as <expN>, ALEN() returns the number of rows in the array. If the array is a one-dimensional array, ALEN() returns a single subscript. If the array is a two-dimensional array, ALEN() returns the first of the two subscripts.

If 2 is specified, ALEN() returns the number of columns in the array. If the array is a one-dimensional array, ALEN() returns 0 (no columns). If the array is a two-dimensional array, ALEN() returns the second of the two subscripts.

Examples

```
DIMENSION x(5)
DIMENSION y(3,7)
```

```
? ALEN(x)
```

5

```
? ALEN(x,0)
```

5

```
? ALEN(x,1)
```

5

```
? ALEN(x,2)
```

0

```
? ALEN(y)
```

21

```
? ALEN(y,1)
```

3

```
? ALEN(y,2)
```

7

```
SELECT DISTINCT state FROM customers INTO ARRAY states
```

```
? ALEN(states)
```

49

The following example copies OFFICES.DBF to the memory variable array OFF, then displays the contents of two of the fields from the array.

```
CLOSE DATABASES
```

```
SELECT * FROM offices ;
```

```
    INTO ARRAY off
```

```
FOR i = 1 to ALEN(off,1)
```

```
    ? 'Office: ',off[i,6], 'Sales: ',off[i,2]
```

```
ENDFOR
```


ALIAS()

Purpose	Returns the alias of the current or specified work area
Syntax	ALIAS([<expN> <expC>])
Returns	Character
See Also	DBF(), SELECT()

Description ALIAS() returns the alias for a table open in a specified work area.

Parameter <expN> | <expC>

ALIAS() returns the table alias for the work area specified by the numeric expression <expN> or table alias <expC>.

If <expN> or <expC> are omitted, ALIAS() returns the alias of the table open in the current work area. If a table isn't open in the current or specified work area, the null string is returned.

Example The following example uses tables and displays the alias of each.

```
CLOSE DATABASES
SELECT A
USE salesman
SELECT B
USE customer ALIAS cust
```

```
? ALIAS( )
CUST
```

```
? ALIAS('salesman')
SALESMAN
```

ALLTRIM()

Purpose	Returns the specified character expression with leading and trailing blanks removed
Syntax	ALLTRIM(<expC>)
Returns	Character
See Also	LTRIM(), RTRIM(), TRIM()

Description ALLTRIM() removes leading and trailing blanks from the specified character expression and returns the trimmed expression as a character string. This function can be used to ensure that blanks are removed from data entered by a user.

Parameter <expC>
Leading and trailing blanks are removed from the character expression <expC>.

Example In this example, if extra spaces are included before the name of the contact person (here the name Gene Smith is entered), ALLTRIM() removes the spaces. Removing the leading spaces ensures that LOCATE can find the name.

```
CLOSE DATABASES
USE customer
@ 5,0 SAY 'Name of Contact person: ' GET person ;
    DEFAULT 'Gene Smith'
READ
LOCATE FOR contact = ALLTRIM(person)

? RECNO( )
494

DISPLAY contact, company
Record#  CONTACT                COMPANY
494      Gene Smith              Wilson Solutions
```

ANSITOOEM()

Purpose	Converts each character of a character expression to the corresponding character in the MS-DOS (OEM) character set.
Syntax	ANSITOOEM(<expC>)
Parameters	<expC> The character expression to convert
Returns	Character
Remarks	FoxPro for Windows only
See Also	CHRTRAN()

Description Use ANSITOOEM() to move data from FoxPro for Windows to FoxPro for MS-DOS.

ANSITOOEM() converts each character in <expC> to the corresponding character in the MS-DOS (OEM) character set. The character expression <expC> should contain characters from the ANSI character set.

If a character in <expC> doesn't have an MS-DOS equivalent, the character is converted to a similar MS-DOS character.

APPEND

Purpose	Enables you to add records to the end of the current table
Syntax	APPEND [BLANK]
See Also	APPEND FROM ARRAY, BROWSE, CHANGE, EDIT, INSERT - SQL, REPLACE, SET CARRY

Description APPEND enables you to add one or more new records to the end of the current table/.DBF. The Open dialog appears when you issue APPEND or APPEND BLANK and a table isn't open in the current work area. Choose a table to append records to.

APPEND opens an editing window so you can enter data into new records. Any indexes that are open when APPEND is executed are updated when the record is added.

Clause **BLANK**

APPEND BLANK adds one blank record to the end of the current table. An editing window isn't opened when you issue APPEND BLANK. The faster INSERT - SQL command is preferable.

You can edit the new record with BROWSE, CHANGE or EDIT.

When a blank record is appended, each field in the record is initially empty unless CARRY is SET ON.

APPEND FROM

Purpose	Adds records to the end of the current table from another file
Syntax	APPEND FROM <file> ? [FIELDS <field list>] [FOR <expL>] [[TYPE] [DELIMITED [WITH TAB WITH <delimiter> WITH BLANK] DIF FW2 MOD PDOX RPD SDF SYLK WK1 WK3 WKS WR1 WRK XLS]]
See Also	COPY FILE, COPY TO, IMPORT, EXPORT

Description The file you are appending from is assumed to be a FoxPro table with a .DBF extension. If the file you want to append from is a FoxPro table and doesn't have a .DBF extension, you must specify its extension. If the file is a not a FoxPro table, you must specify the type of file you append from.

Before you can append from a table created in DB4 that contains a memo field, you must first open the table in FoxPro with USE. You are prompted with "Convert MEMO file to FoxPro Format?" Choose **Yes**.

If you append from a FoxPro table, the table you append from can be open in another work area. You can also append from a table that isn't open but is available on disk and a shared table opened when SET EXCLUSIVE is OFF. When the table you append from contains records marked for deletion, the records are not marked for deletion after they are appended.

If you include the ? clause instead of including a table name, the Open dialog appears so you can choose a table to append from.

Clauses **<file>**

Specify the name of the file to append from with <file>. If you don't include a file name extension, the default extension .DBF is assumed.

FIELDS <field list>

APPEND FROM supports an optional <field list>. Data is only appended to the fields specified in the field list.

FOR <expL>

The entire source file is appended to the table unless you include the FOR clause. If the FOR clause is included, a new record is appended for each record in the file source for which <expL> evaluates to a logical true (.T.). Records are appended until the end of the file is reached.

TYPE

If the file you are appending from isn't a FoxPro table, you must specify the file TYPE. Although you must specify the file type, you need not include the key word TYPE. You can append from a wide variety of different file types including DELIMITED ASCII text files in which you can specify a field delimiter.

If the file you are appending from doesn't have the usual default file extension for that type of file, the source file name must include the file's extension. For example, Microsoft Excel spreadsheets normally have an .XLS file name extension. If the spreadsheet you are appending from has an extension other than the expected .XLS, be sure to specify the extension.



When appending from a spreadsheet, the data in the spreadsheet must be stored in a row major order rather than a column major order. This allows the appended spreadsheet data to match the table structure.

DELIMITED [WITH TAB | WITH <delimiter> | WITH BLANK]

A DELIMITED file is an ASCII text file in which each record ends with a carriage return and line feed. Field contents are by default assumed to be separated from each other by commas, and character field values to be additionally delimited by double quotation marks. For example:

```
''Smith'', 9999999, ''TELEPHONE''
```

The DELIMITED WITH TAB option can be used to specify files which contain fields separated from each other by tabs rather than commas. The DELIMITED WITH <delimiter> option can be used to indicate that character fields are delimited by a character other than the quotation mark. The DELIMITED WITH BLANK option can be used to specify files which contain fields separated by spaces instead of commas. The file extension is assumed to be .TXT for all delimited files.

You can import dates from delimited files if the dates are in proper date format. The date format defaults to 'mm/dd/yy'. Including the century portion of a date is optional. FoxPro will import a date that includes the century. If the century isn't included in a date (for example '12/25/92'), the Twentieth century is assumed. Date delimiters can be any non-numeric character except the delimiter that separates the fields in the delimited file.

Dates in other formats can be imported if their format matches a date format available in SET DATE. To import dates that are not in the default format, issue SET DATE with the proper date format before using APPEND FROM. To test if a date format can be successfully imported, use it with CTOD(). If the date is acceptable to CTOD(), the date will import properly.

DIF

In a DIF (VisiCalc Data Interchange Format) file, vectors (columns) become fields and tuples (rows) become records. DIF file names are assumed to have a .DIF extension.

FW2

FW2 files are created by Framework II™. FW2 file names are assumed to have a .FW2 extension.

MOD

MOD files are created by Microsoft Multiplan® version 4.01. MOD files are assumed to have a .MOD extension.

PDOX

Database files in Paradox® versions 3.5 and 4.0 can be appended to a FoxPro table by including the PDOX option. Paradox file names are assumed to have a .DB extension.

RPD

RPD files are created by RapidFile® version 1.2. RPD file names are assumed to have an RPD extension.

SDF

An SDF (System Data Format) file is an ASCII text file in which records have a fixed length and end with a carriage return and a line feed. Fields are not delimited. The file name extension is assumed to be .TXT for SDF files.

SYLK

An SYLK file is in a Symbolic Link interchange format (used in Microsoft Multiplan) in which columns become fields and rows become records. SYLK file names have no extension by default.

WK1

Data from a Lotus[®] 1-2-3[®] spreadsheet can be appended to a FoxPro table with this option. Each column from the spreadsheet becomes a field in the table; each spreadsheet row becomes a record in the table. A .WK1 file name extension is assigned to a spreadsheet created in Lotus 1-2-3 revision 2.x.

WK3

Data from a Lotus 1-2-3 spreadsheet. Each column from the spreadsheet becomes a field in the table; each spreadsheet row becomes a record in the table. A .WK3 file name extension is assigned to a spreadsheet created in Lotus 1-2-3 revision 3.x.

WKS

Data from a Lotus 1-2-3 spreadsheet. Each column from the spreadsheet becomes a field in the table and each spreadsheet row becomes a record in the table. A .WKS file name extension is assigned to a spreadsheet created in Lotus 1-2-3 revision 1-A.

WR1

Data from a Lotus Symphony[®] spreadsheet. Each column from the spreadsheet becomes a field in the table and each spreadsheet row becomes a record in the table. A .WR1 file name extension is assigned to a spreadsheet created in Symphony versions 1.1 or 1.2.

WRK

Data from a Lotus Symphony spreadsheet. Each column from the spreadsheet becomes a field in the table and each spreadsheet row becomes a record in the table. A .WRK file name extension is assigned to a spreadsheet created in Symphony version 1.0.

XLS

The data in Microsoft Excel versions 2.0, 3.0 and 4.0 spreadsheets can be appended to a FoxPro table by including the XLS option. Each column from the spreadsheet becomes a field in the table and each spreadsheet row becomes a record in the table. Spreadsheet files created in Microsoft Excel are given an .XLS file name extension.

Example In this example, the CUSTOMER table is opened, its structure is copied to a table called BACKUP and BACKUP is opened. FoxPro then appends all records from the CUSTOMER table from the state of Ohio (OH). These records are then copied to a new delimited file called TEMP.TXT.

```
CLOSE DATABASES
USE customer
COPY STRUCTURE TO backup
USE backup
APPEND FROM customer FOR state = 'OH'
COPY TO temp TYPE DELIMITED
MODIFY FILE temp.txt
USE
DELETE FILE backup.dbf
DELETE FILE temp.txt
```

APPEND FROM ARRAY

Purpose	Adds records to the current table and fills the records with data from an array
Syntax	APPEND FROM ARRAY <array> [FOR <expL>] [FIELDS <field list>]
See Also	APPEND BLANK, COPY TO ARRAY, DIMENSION, GATHER, SCATTER

Description APPEND FROM ARRAY adds one record to the currently selected table for each row in an array and fills each record with data from the corresponding array row. Memo and general fields are ignored in APPEND FROM ARRAY. When a table is open for shared use, APPEND FROM ARRAY locks the table header while records are being added.

Clauses <array>
Include the name of the array that contains the data to be copied to the new records. New records are added to the table until all rows in the array are appended.

One-Dimensional Arrays

If the array is one-dimensional, APPEND FROM ARRAY adds one record to the table. The contents of the first array element fill the first field of the newly added record, the contents of the second element of the array fill the second field of the record and so on.

If the one-dimensional array has more elements than the table has fields, the additional elements are ignored. If the table has more fields than the array has elements, the additional fields are initialized to the default empty value. Here are the default empty values for each field type:

Field Type	Default Value
Character	Spaces
Numeric	0
Date	Null (/ /)
Logical	False (.F.)

Two-dimensional Arrays

When appending from a two-dimensional array, a new record is appended to the table for each row in the array. For example, if the array has four rows, four new records are appended to the table.

The contents of the first array column fills the first field of the newly added records, the second array column fills the second field of the new records and so on. For example, if the array has four rows and three columns, elements from the first array column fill the first field in each of the four new records appended to the table.

If the two-dimensional array has more columns than the table has fields, the additional columns are ignored. If the table has more fields than the array has columns, the additional fields are initialized to empty values.

FOR <expL>

If you include the FOR clause, you can conditionally append records from the array. <expL> must contain the name of a target field in its conditional expression.

Before a row of the array is appended to a record in the table, the array element corresponding to the target field specified in <expL> is checked to see if that array element meets the condition in <expL>. If the array element satisfies the condition, a record is appended.

If the array element doesn't satisfy the condition, the array row isn't appended and the next row in the array is checked to see if it meets the condition.

FIELDS <field list>

If FIELDS <field list> is included, only the fields specified in the list are updated from the array. The first field in the list is updated with the contents of the first element in the array, the second field is updated from the second element and so on.

Data Type Mismatches

APPEND FROM ARRAY can fill a field even if the data type of the corresponding array element doesn't match the data type of the field, provided the array element data is compatible with the data type of the corresponding field. If the data isn't compatible, the field is initialized to an empty value. See the default value table above for the empty values for each field type.

The following table illustrates whether the replacement occurs when there is a data type mismatch. **Yes** indicates that if there is a data type mismatch the element's data is converted to the data type of the corresponding field and a replacement is made. **No** indicates the field is initialized to its default value.

Element Type	Field Type			
	Character	Numeric	Date	Logical
Character	Yes	No	No	No
Numeric	Yes	Yes	No	Yes
Date	Yes	Yes	Yes	No
Logical	Yes	No	No	Yes

For example, if an array element contains a logical true (.T.) or false (.F.) value and the element's corresponding field in the newly appended record is of character type, FoxPro replaces the field with the character T or F respectively.

Example

This example illustrates how you can add a record to a table. The CUSTOMER table is opened and SCATTER creates an array named CUSTARRAY with a blank element for each field in the table. A FOR ... ENDFOR loop displays each field. The READ activates the @ ... GETs.

If Escape isn't pressed to stop the editing, APPEND FROM ARRAY appends a new record to the table. The data in the CUSTARRAY memory variable array then replaces each field.

```
SET TALK OFF
CLEAR
CLOSE DATABASES
USE customer
SCATTER TO custarray BLANK      && Create an blank array

FOR count = 1 TO ALLEN(custarray)  && Number of rows in array
    @ count, 2 SAY FIELD(count) ;
    GET custarray(count)          && Display fields and get info
ENDFOR

READ
IF READKEY( ) != 12              && If Escape isn't pressed
    APPEND FROM ARRAY custarray && Append a record, fill from array
    GOTO BOTTOM
    BROWSE
    DELETE
    PACK
ENDIF
USE
CLEAR
```

APPEND GENERAL

Purpose	Imports an OLE object into a general field
Syntax	APPEND GENERAL <general field> FROM <file> [LINK] [CLASS <ole class>]
Remarks	FoxPro for Windows only
See Also	@ ... SAY, MODIFY GENERAL

Description Use APPEND GENERAL to import an OLE object from a file and place the object in a general field. If an OLE object already exists in the general field, it is replaced with the OLE object from the file.

For additional information about OLE objects in FoxPro for Windows, see the Object Linking and Embedding (OLE) chapter in the FoxPro for Windows *Developer's Guide*.

Clauses **<general field>**
Include <general field> to specify the name of the general field the OLE object is placed in. You can specify a general field in a table open in a noncurrent work area by including the table alias with the field name.

<file>
Include <file> to specify the file containing the OLE object. You must include the entire file name, including its extension. If the file is located in a directory other than the current default directory, include the path with the file name.

LINK
If LINK is included, a link is created between the OLE object and the file that contains the object. The OLE object appears in the general field but the object's definition remains in the file. If LINK is omitted, the OLE object is embedded in the general field.

CLASS <ole class>

Include the CLASS clause to specify an OLE class for an OLE object other than the default class.



To determine the class for an OLE object, run REGEDIT and double-click on the OLE object. The class name is listed under Identifier.

You can specify a class name when the file extension for the file containing the OLE object is different than the default extension and you want to force the behavior for the class. If the default extension can be used by multiple OLE servers, include the class to specify a particular server.

Example

The following example imports an Excel chart file to a general field named CHART.

```
APPEND GENERAL chart FROM ;  
"C:\EXCEL\CHART1.XLC" CLASS EXCELCHART
```

APPEND MEMO

Purpose	Copies the contents of a file to a memo field
Syntax	APPEND MEMO <memo field> FROM <file> [OVERWRITE]
See Also	COPY MEMO

Description APPEND MEMO lets you import data from a file to a memo field. The entire contents of the file are appended to the contents of the specified memo field in the current record.

Clauses **<memo field>**
The memo field to be updated.

FROM <file>
The file from which the memo field is to be appended. You must include the entire file name, including its extension.

OVERWRITE
If the optional key word OVERWRITE is included, the current contents of the memo field are replaced with the contents of the file.

Example In this example, the contents of the memo field NOTES are copied to a file called TEST.TXT. The contents of TEST.TXT is then appended to the contents of the memo field. Finally, the contents of TEST.TXT replace the current contents of the memo field.

```
CLOSE DATABASES
USE salesman
WAIT WINDOW 'salesman.notes -- press Esc' NOWAIT
MODIFY MEMO notes
COPY MEMO notes TO test.txt
WAIT WINDOW 'test.txt -- press Esc' NOWAIT
MODIFY FILE test.txt
WAIT WINDOW 'salesman.notes now appended to -- press Esc' NOWAIT
APPEND MEMO notes FROM test.txt
MODIFY MEMO notes
WAIT WINDOW 'salesman.notes append overwrite -- press Esc' NOWAIT
APPEND MEMO notes FROM test.txt OVERWRITE
MODIFY MEMO notes
DELETE FILE test.txt
```


ASC()

Purpose	Returns the ASCII code for the leftmost character in a character expression
Syntax	ASC(<expC>)
Returns	Numeric
See Also	CHR()

Description Every character has a unique ASCII value in the range from 0 to 255.

In FoxPro for MS-DOS you can open the ASCII Chart desk accessory for a list of characters and their ASCII values.

Parameters <expC>

The ASCII value is returned for the first character in the character expression <expC>. Any characters after the first character in <expC> are ignored by ASC().

Examples

```
USE salesman
? name
Janet A. McCarthy
? ASC(name)
74
? ASC('A')
65
? ASC('a')
97
```

ASCAN()

Purpose	Searches an array for an expression
Syntax	ASCAN(<array>, <expr> [, <expN1> [, <expN2>]])
Parameters	<array> The array to search <expr> The expression to search for <expN1> The element where the search begins <expN2> The number of elements to search
Returns	Numeric
See Also	ACOPY(), ADEL(), ADIR(), AELEMENT(), AFIELDS(), AINS(), ASORT(), ASUBSCRIPT(), DIMENSION, SET EXACT

Description ASCAN() searches an array for an element containing the same data and data type as <expr>. If a match is found, the number of the element containing the expression is returned. If a match cannot be found, 0 is returned.

The criteria for a successful match of character data is determined by the setting of SET EXACT (ON or OFF). If SET EXACT is ON, an element must match the search expression character for character and have the same length. If SET EXACT is OFF and an element and search expression match up to the point where the expression is exhausted, the match is successful. For more information on match criteria for character strings, see the string comparison table in the SET EXACT topic in this chapter.

Parameters <array>, <expr>

Include the name of the array to search and the general expression <expr> to search for.

<expN1>

By default, the entire array is searched. The search can be started at an element other than the first element by including <expN1>. The search begins at element number <expN1>, with element number <expN1> included in the search.

<expN2>

A specific number of elements can be searched by including <expN2>. The array is searched beginning with element <expN1>, and <expN2> elements are searched for a match.

If the optional numeric expressions <expN1> and <expN2> are omitted the search begins with the first array element and continues to the last array element.



You can refer to an element in a two-dimensional memory variable array in one of two ways. The first method uses two subscripts to specify the row and column position of the element in the array; the other method uses an element number. This function and others that manipulate two dimensional arrays require element numbers (<expN1> and <expN2>).

Use AELEMENT() to return the element number from row and column subscripts in a two-dimensional array.

Example

The following example dimensions and fills an array from the COMPANY field of the CUSTOMER table, then searches the array for the element containing “1st Computers” and deletes that element.

```
CLOSE DATABASES
USE customer
COUNT TO m.count

*** DIMENSION array large enough to hold data ***

DIMENSION companies[m.count,1]

COPY TO ARRAY companies FIELD company  && Fill array

m.name = '1st Computers'
m.pos  = ASCAN(companies, m.name)  && Search for company

IF m.pos != 0
    msg1 = 'Company in cell 1: '+companies(m.pos)
    WAIT WINDOW (msg1) TIMEOUT 3
    = ADEL(companies, m.pos)
    m.count = m.count - 1
    msg1 = 'Company removed, now cell 1: '+companies(m.pos)
    WAIT WINDOW (msg1) NOWAIT
ELSE
    WAIT WINDOW '1st Computers not found.' NOWAIT
ENDIF
```

ASIN()

Purpose	Returns the arc sine of a specified numeric expression
Syntax	ASIN(<expN>)
Returns	Numeric
See Also	RTOD(), SET DECIMALS, SIN()

Description This trigonometric function returns in radians the arc sine of the numeric expression <expN>.

Parameters <expN>
The value of <expN> can range from +1 through -1, and the value returned by ASIN() can range from - $\pi/2$ through + $\pi/2$ ($\approx$ -1.57079 to 1.57079). The number of decimal places in the display of the result can be specified with SET DECIMALS.

Use RTOD() to convert radians to degrees.

Examples ? RTOD(ASIN(0))
 0.00
 STORE 1 to arcangle
 ? RTOD(ASIN(arcangle))
 90.00
 ? RTOD(ASIN(SQRT(2)/2))
 45.00

ASORT()

Purpose	Sorts elements in an array in ascending or descending order
Syntax	ASORT(<array> [, <expN1> [, <expN2> [, <expN3>]]])
Parameters	<array> The memory variable array to sort <expN1> Used singly, <expN1> is the element where sorting starts Used with <expN2>, <expN1> is the row where sorting starts <expN2> The number of elements or rows to sort <expN3> The sort order. 0 for ascending, 1 for descending
Returns	Numeric
See Also	ACOPY(), ADEL(), ADIR(), AELEMENT(), AFIELDS(), AINS(), ALLEN(), ASCAN()

Description All elements included in the sort must be of the same data type (character, numeric, date or logical). One-dimensional arrays are sorted by their elements; two-dimensional arrays are sorted by their rows. When a two-dimensional array is sorted, the order of the rows in the array is changed so the elements in a *column* of the array are in ascending or descending order.

If the sort is successful, 1 is returned; otherwise -1 is returned.

Parameters **<array>**

Include the name of the array to sort.

<expN1>

By default, arrays are sorted starting with the first array element. You may designate a different starting element for the sort by including <expN1>. If the array is one-dimensional, the array is sorted starting with the <expN1>th element; the starting element <expN1> is included in the sort. If the array is two-dimensional, the starting element <expN1> determines both the row where the sort begins and the column that determines the sort order of the rows.



You can refer to an element in a two-dimensional array in one of two ways. The first method uses two subscripts to specify the row and column position of the element in the array; the other method uses an element number. This function and others that manipulate two-dimensional arrays require element numbers (in ASORT() the numeric expressions <expN1> and <expN2>).

You can use AELEMENT() to return the element number from row and column subscripts in a two-dimensional array.

The following example illustrates that the starting element <expN1> determines how the rows in a two-dimensional array are sorted. A small array named MARRAY is created and sorted twice. The first sort begins with the first element of MARRAY, the rows are sorted based on the values contained in the first *column* of the array. The second sort begins with the fourth element of MARRAY, and the rows are sorted based on the values contained in the second *column*.

The first sort begins with the first row. The second sort begins with the second row. You can use DISPLAY MEMORY to display the contents of the array; in these examples tables are used to graphically display the results of the sorts.

These commands create the array named MARRAY:

```
DIMENSION marray(3,2)
marray(1) = 'G'
marray(2) = 'A'
marray(3) = 'C'
marray(4) = 'Z'
marray(5) = 'B'
marray(6) = 'N'
```

MARRAY looks like this:

	Column 1	Column 2
Row 1	G	A
Row 2	C	Z
Row 3	B	N

The array is then sorted by ASORT() starting with the first element (1,1) in the array. The elements in the first column are placed in ascending order by rearranging the rows of the array.

```
=ASORT(marray,1)
```

Note the new order of the rows:

	Column 1	Column 2
Row 1	B	N
Row 2	C	Z
Row 3	G	A

The array is then sorted starting with the fourth element (2,2) in the array. The elements in the second column are placed in order by rearranging the array rows.

```
=ASORT(marray,4)
```

Note the difference in the order of the rows:

	Column 1	Column 2
Row 1	B	N
Row 2	G	A
Row 3	C	Z

<expN2>

If you include a starting element <expN1> you can also specify the number of elements in a one-dimensional array or rows in a two-dimensional array to sort. The number of elements or rows that are sorted is specified with <expN2>. For example, if the array is one-dimensional and <expN1> is 2 (start with the second array element) and <expN2> is 3 (use three elements), the second, third and fourth array elements are sorted. If <expN2> is -1 or is omitted, all array elements from the starting element <expN1> through the last element in the array are sorted.

If the array is two-dimensional, <expN2> designates the number of rows to sort, beginning with the row containing the starting element <expN1>. For example, if <expN1> is 2 and <expN2> is 3, the row containing the second array element and the following two rows are sorted. If <expN2> is -1 or is omitted, all array rows beginning with the row containing the starting element <expN1> through the last array row are sorted.

<expN3>

By default, array elements are sorted in ascending order. You can include <expN3> to specify that the array's sort order (ascending or descending). If <expN3> is 0, the array elements are sorted in ascending order. If <expN3> is 1 or any non-zero value, the array elements are sorted in descending order.

Example

The following example copies the CNO and COMPANY fields from the CUSTOMER table to an array named COMPANIES. The array is then sorted based on the elements in the first column starting with the fourth row. In other words, the array is sorted starting with element 7. A list containing the elements in the first column (CNO) of the COMPANIES array is displayed.

```
*** SELECT or USE customer ***

IF USED('customer')
    SELECT customer
ELSE
    USE LOCFILE('customer.dbf','DBF','Where is CUSTOMER.DBF?')
ENDIF

COUNT TO m.count

*** DIMENSION array large enough to hold data ***

DIMENSION companies[m.count,2]

COPY TO ARRAY companies FIELD cno,company  && Fill the array

= ASORT(companies, AELEMENT(companies,4,1))  && Sort array
WAIT WINDOW 'First three rows are not sorted' NOWAIT
@ 5,30 SAY 'List of Companies CNO:'
@ 7,30 GET m.choice ;
    FROM companies ;
    RANGE 1 ;
    SIZE 10,30 ;
    DEFAULT 1

READ  && Display and GET sorted company data

CLEAR
USE
```


ASUBSCRIPT()

Purpose	Returns the row or column subscript of an element from the element's number
Syntax	ASUBSCRIPT(<array>, <expN1>, <expN2>)
Parameters	<array> An array name <expN1> An element number <expN2> If 1, ASUBSCRIPT() returns the row subscript If 2, ASUBSCRIPT() returns the column subscript
Returns	Numeric
See Also	ADEL(), ADIR(), AELEMENT(), AFIELDS(), AINS(), ALLEN(), ASCAN(), ASORT(), DIMENSION, DISPLAY MEMORY

Description You can refer to elements in two-dimensional memory variable arrays in one of two ways. The first method uses two subscripts to specify the row and column position of the element in the array. The second method uses an element number. Use ASUBSCRIPT() to obtain an element's row or column subscript from the element's number.

In the following example, an array with two rows and three columns is created. DISPLAY MEMORY shows the contents of the elements of the array listed in element number order.

Example DIMENSION x(2,3)
 DISPLAY MEMORY LIKE x

X		Pub	A	
(	1,	1)	L	.F. (element number 1)
(	1,	2)	L	.F. (element number 2)
(	1,	3)	L	.F. (element number 3)
(	2,	1)	L	.F. (element number 4)
(	2,	2)	L	.F. (element number 5)
(	2,	3)	L	.F. (element number 6)

These commands both store the character string INVOICE to the same array element:

STORE 'INVOICE' TO X(2, 1) and STORE 'INVOICE' TO X(4)

In one-dimensional arrays, an element's number is identical to its single row subscript. It isn't necessary to use ASUBSCRIPT() with one-dimensional arrays.

Parameters <array>, <expN1>, <expN2>

If the array is one-dimensional, include the element number in <expN1> and 1 in <expN2>. ASUBSCRIPT() identically returns <expN1>.

If the array is two-dimensional you must include both the element number <expN1> and a value of 1 or 2 in <expN2>. Specifying 1 in <expN2> returns the row subscript of the element, and specifying 2 returns the column subscript.

For more information on how to reference elements in an array, refer to DIMENSION in this chapter.

AT()

Purpose	Returns the beginning numeric position of the first occurrence of a character expression or memo field within another character expression or memo field, counting from the leftmost character
Syntax	AT(<expC1>, <expC2> [, <expN>])
Parameters	<expC1> The character expression to search for <expC2> The character expression to be searched <expN> Specifies which occurrence of <expC1> in <expC2> to search for
Returns	Numeric
See Also	ATC(), LEFT(), RAT(), RIGHT(), SUBSTR()

Description AT() searches the second character expression for the first occurrence of the first character expression. It then returns an integer indicating the position of the first character in the found character expression. If the character expression isn't found, 0 is returned.

The search performed by AT() is case sensitive (upper and lower-case is respected). To perform a search that isn't case sensitive, use ATC().

Parameters <expC1>, <expC2>

FoxPro looks for the search character expression <expC1> in the second character expression <expC2>. The character expressions <expC1> and <expC2> can be memo fields of any size.

<expN>

If <expN> is included, AT() searches for the <expN>th occurrence of <expC1> in <expC2>. By default, AT() searches for the first occurrence of <expC1> (<expN> = 1). Including <expN> lets you search for additional occurrences of <expC1> in <expC2>.

AT()

Examples

```
STORE 'Now is the time for all good men' TO string
STORE 'is the' TO find_str
? AT(find_str,string)
5
```

```
STORE 'IS' TO find_str
? AT(find_str,string)
0
```

```
USE parts
STORE 'Kimono' to string
LIST ALL pno,descript FOR AT(string,descript) > 0
```

```
STORE 'AB' TO find_str
STORE 'ABABAB' TO string
? AT(find_str,string,2)
3
```

ATAN()

Purpose	Returns the arc tangent of a specified numeric expression
Syntax	ATAN(<expN>)
Returns	Numeric
See Also	ATN2(), RTOD(), SET DECIMALS, TAN()

Description This trigonometric function returns in radians the arc tangent of a numeric expression.

Parameters <expN>

ATAN() returns the arc tangent of the numeric expression <expN>. <expN> can be any value; the value returned by ATAN() can range from $-\pi/2$ through $+\pi/2$ (≈ -1.57079 to 1.57079). The number of decimal places displayed in the value returned by ATAN() is determined by SET DECIMALS.

The numeric value returned by ATAN() is in radians. Use RTOD() to convert this value to degrees.

Example

```
? ATAN(0)
0.00
STORE PI( )/2 to angle
? ATAN(angle)
1.00
? ATAN(PI( )/2)
1.00
? ATAN(DTOR(90))
1.00
```

ATC()

Purpose	Returns the beginning numeric position of the first occurrence of a character expression or memo field within another character expression or memo field, without regard for the case of these two expressions
Syntax	ATC(<expC1>, <expC2> [, <expN>])
Parameters	<expC1> The expression to search for <expC2> The expression to be searched <expN> Specifies which occurrence of <expC1> to look for
Returns	Numeric
See Also	AT(), ATCLINE(), LEFT(), RAT(), RIGHT(), SUBSTR()

Description ATC() searches the second character expression for the occurrence of the first character expression, without concern for the case (upper or lower) of the characters in either expression. Use AT() to perform a case-sensitive search.

ATC() returns an integer corresponding to the position where the first character of the character expression is found. If the character expression isn't found, 0 is returned.

Parameters **<expC1>, <expC2>**

ATC() searches <expC2> for <expC1>. The character expressions <expC1> and <expC2> can be memo fields of any size.

<expN>

If <expN> is included, ATC() searches for the <expN>th occurrence of <expC1> in <expC2>. By default, ATC() searches for the first occurrence of <expC1> (<expN> = 1). Including <expN> lets you search for additional occurrences of <expC1> in <expC2>.

Example

```
STORE 'Now is the time for all good men ...' TO string
STORE 'IS THE' TO find_str
? ATC(find_str,string)
5
STORE 'IS' TO find_str
? ATC(find_str,string)
5
? ATC('now',string)
1
```

ATCLINE()

Purpose	Returns the line number of the first occurrence of a character expression or memo field within another character expression or memo field; without regard for the case of the two expressions
Syntax	ATCLINE(<expC1>, <expC2>)
Parameters	<expC1> The expression to search for <expC2> The expression to be searched
Returns	Numeric
See Also	AT(), ATC(), ATLINE(), MLINE(), SET MEMOWIDTH

Description ATCLINE() searches the second character expression for the occurrence of the first character expression without concern for the case (upper or lower) of the characters in either expression. Use ATLINE() to perform a case-sensitive search.

If the search is successful, the number of the line containing the first character expression is returned. If the search is unsuccessful, 0 is returned.



The line number returned by ATCLINE() is determined by the value of SET MEMOWIDTH, even if <expC2> isn't a memo field. Refer to SET MEMOWIDTH in this chapter for additional information.

Parameters <expC1>, <expC2>

ATCLINE() searches <expC2> for <expC1>. The character expressions <expC1> and <expC2> can be memo fields of any size.

The line containing the matching character expression can be returned with MLINE().



ATCLINE() offers a convenient way to search memo fields.

Examples The following example locates the first time a character string occurs in a memo field and displays the city, state and line of the memo for the record.

```
CLOSE DATABASES
USE salesman
GO TOP
CLEAR
STORE 'JANET' TO findstring
LOCATE FOR ATCLINE(findstring, notes) != 0
? city
? state
? MLINE(notes, ATCLINE(findstring, notes))
```

The following example demonstrates how the memo width affects ATCLINE().

```
STORE '1234567890ABCDEFGHILJ' TO string
SET MEMOWIDTH TO 20
? ATCLINE ('AB', string)
1
SET MEMOWIDTH TO 10
? ATCLINE('AB', string)
2
```


ATLINE()

Purpose	Returns the line number of the first occurrence of a character expression or memo field within another character expression or memo field, counting from the first line
Syntax	ATLINE(<expC1>, <expC2>)
Parameters	<expC1> The expression to search for <expC2> The expression to be searched
Returns	Numeric
See Also	AT(), ATC(), ATCLINE(), MLINE(), SET MEMOWIDTH

Description ATLINE() searches the second character expression for the occurrence of the first character expression. The case (upper or lower) of the characters in either expression is respected. Use ATCLINE() to perform a search that isn't case sensitive.

If the search is successful, the number of the line where the match occurs is returned. If the search is unsuccessful, 0 is returned.



The line number returned by ATLINE() is determined by the value of SET MEMOWIDTH, even if <expC2> isn't a memo field. Refer to SET MEMOWIDTH in this chapter for additional information.

Parameters <expC1>, <expC2>

ATLINE() searches <expC2> for <expC1>. The character expressions <expC1> and <expC2> can be memo fields of any size.

The line containing the matching character expression can be returned as a character string with MLINE().



ATLINE() offers a convenient way to search memo fields.

ATLINE()

Examples

```
CLOSE DATABASES
USE salesman
GO TOP
CLEAR
STORE 'Janet' TO findstring
LOCATE FOR ATLINE(findstring, notes) != 0
? city
? state
? MLINE(notes, ATLINE(findstring, notes))
```

The following example demonstrates how the memo width affects ATLINE().

```
STORE '1234567890ABCDEFGHILJ' TO string
SET MEMOWIDTH TO 20
? ATLINE('AB', string)
1
SET MEMOWIDTH TO 10
? ATLINE('AB', string)
2
```

ATN2()

Purpose	Returns the arc tangent from specified sine and cosine values
Syntax	ATN2(<expN1>, <expN2>)
Parameters	<expN1> The <i>x</i> coordinate <expN2> The <i>y</i> coordinate
Returns	Numeric
See Also	ATAN(), DTOR(), RTOD(), SET DECIMALS, TAN()

Description This trigonometric function returns an arc tangent angle in all four quadrants. In ATN2() you specify a set of *x* and *y* coordinates. In ATAN() you specify a tangent value.

Parameters **<expN1>, <expN2>**

Specify the *x* coordinate with <expN1> and the *y* coordinate with <expN2>. The value of <expN1>/<expN2> must be between $-\pi$ and $+\pi$.

The value returned by ATN2() can be converted to degrees with RTOD(). The number of decimal places in the display of the result can be specified with SET DECIMALS.

Example

```

? PI( )
3.14
? ATN2(0,-1)
3.14
STORE COS(PI( )) TO X
STORE SIN(PI( )) TO Y
? ATN2(Y,X)
3.14
? ATN2(Y,X)/PI( )
1.00

```

AVERAGE

Purpose	Computes the arithmetic mean of numeric expressions or fields
Syntax	AVERAGE [<expr list>] [<scope>] [FOR <expL1>] [WHILE <expL2>] [TO <memvar list> TO ARRAY <array>] [NOOPTIMIZE]
See Also	CALCULATE, DIMENSION, SET HEADING, SUM

Description All numeric fields in the selected table are averaged unless you include an optional expression list. The result is displayed on the screen if SET TALK is ON. If SET HEADINGS is ON, the field names or expression involving the field names are displayed above the results.

Clauses **<expr list>**
These are expressions to be averaged. <expr list> can be a list of fields from the table separated by commas, or numeric expressions involving fields from the table.

<scope>
The scope clauses are: ALL, NEXT <expN>, RECORD <expN>, and REST. These are explained in the Overview of the FoxPro Language chapter of this manual. Commands which include <scope> operate only on the table in the active work area.

You can specify a scope of records to copy to average. Only the records that fall within the range of records specified by the scope are averaged.

The default scope for AVERAGE is ALL records.

FOR <expL1>
If FOR <expL1> is included, only the records that satisfy the logical condition <expL1> are included in the average. Including FOR lets you conditionally include records in the average, filtering out undesired records.

Rushmore optimizes a AVERAGE FOR query if <expL1> is an optimizable expression. For best performance, use an optimizable expression in the FOR clause. A discussion of Rushmore optimizable expressions appears in the Optimizing Your Application chapter in the *FoxPro Developer's Guide*.

WHILE <expL2>

If WHILE <expL2> is included, records are included in the average for as long as the logical expression <expL2> evaluates to true (.T.).

TO <memvar list> | TO ARRAY <array>

The results of AVERAGE can optionally be stored to a list of memory variables or array elements or to a one-dimensional ARRAY.

If TO ARRAY <array> is included, the results are stored in the one-dimensional array. The one-dimensional array can be created before the execution of AVERAGE.

If the array you include in AVERAGE doesn't exist, FoxPro will automatically create it. If the array exists and isn't large enough to contain all the results, the size of the array is automatically increased to accommodate the information.

NOOPTIMIZE

To disable Rushmore optimization of AVERAGE, include NOOPTIMIZE. For further information on NOOPTIMIZE, see SET OPTIMIZE in this manual and the Rushmore discussion in the Optimizing Your Application chapter in the *FoxPro Developer's Guide*.

Examples

```
USE invoices
AVERAGE itotal FOR BETWEEN(idate,{05/01/90},{05/31/90})
125 records averaged
itotal
    1250.02

AVERAGE itotal TO mavg
206 records averaged.
itotal
    1198.01
? mavg
    1198.01
```

BAR()

Purpose	Returns the number of the most recently chosen option from a popup defined with DEFINE POPUP or a FoxPro menu system popup
Syntax	BAR()
Returns	Numeric
See Also	ACTIVATE POPUP, CNTBAR(), DEFINE BAR, DEFINE POPUP, GETBAR(), MRKBAR(), ON BAR, ON SELECTION BAR, PRMBAR()

Description Each option on a popup is assigned a number with DEFINE BAR. When an option is chosen from the popup, BAR() returns the number assigned to that option. A program can branch to other routines based on the value returned by BAR().

BAR() returns 0 if there is no active popup or if Escape has been pressed to exit the menu.

Example This example uses BAR() to pass the number of a popup option to a procedure. The current system menu bar is saved to memory with SET SYSMENU SAVE, and then all system menu pads are removed with SET SYSMENU TO.

Two menu pads are created with DEFINE PAD, and DEFINE POPUP is used to create a popup for each menu pad. DEFINE BAR is used to create options on each popup. When a menu pad is chosen, ON PAD uses ACTIVATE POPUP to activate the corresponding popup.

When an option is chosen from a popup, ON SELECTION POPUP uses BAR() and POPUP() to pass the option number and popup name to the CHOICE procedure. CHOICE displays the number of the chosen option and the name of the popup containing the option. If the **Exit** option is chosen from the **Card Info** popup, the original FoxPro system menu is restored with SET SYSMENU TO DEFAULT.

```
*** Name this program BAR_EXAM.PRG ***
```

```
CLEAR
```

```
SET SYSMENU SAVE
SET SYSMENU TO
```

```

DEFINE PAD convpad OF _MSYSMENU ;
    PROMPT '\<Conversions' COLOR SCHEME 3 ;
    KEY ALT+C, ''
DEFINE PAD cardpad OF _MSYSMENU ;
    PROMPT 'Card \<Info' COLOR SCHEME 3 ;
    KEY ALT+I, ''

ON PAD convpad OF _MSYSMENU ACTIVATE POPUP conversion
ON PAD cardpad OF _MSYSMENU ACTIVATE POPUP cardinfo

DEFINE POPUP conversion MARGIN RELATIVE SHADOW COLOR SCHEME 4
DEFINE BAR 1 OF conversion PROMPT 'Ar\<ea' ;
    KEY CTRL+E, '^E'
DEFINE BAR 2 OF conversion PROMPT '\<Length' ;
    KEY CTRL+L, '^L'
DEFINE BAR 3 OF conversion PROMPT 'Ma\<ss' ;
    KEY CTRL+S, '^S'
DEFINE BAR 4 OF conversion PROMPT 'Spee\<d' ;
    KEY CTRL+D, '^D'
DEFINE BAR 5 OF conversion PROMPT '\<Temperature' ;
    KEY CTRL+T, '^T'
DEFINE BAR 6 OF conversion PROMPT 'T\<ime' ;
    KEY CTRL+I, '^I'
DEFINE BAR 7 OF conversion PROMPT 'Volu\<me' ;
    KEY CTRL+M, '^M'

*** Here is where the conversion menu pad uses the BAR( ) function
*** to pass a bar number to the procedure called choice below.

ON SELECTION POPUP conversion;
    DO choice IN bar_exam WITH BAR( ), POPUP( )

DEFINE POPUP cardinfo MARGIN RELATIVE SHADOW COLOR SCHEME 4
DEFINE BAR 1 OF cardinfo PROMPT '\<View Charges' ;
    KEY ALT+V, ''
DEFINE BAR 2 OF cardinfo PROMPT 'View \<Payments' ;
    KEY ALT+P, ''
DEFINE BAR 3 OF cardinfo PROMPT 'Vie\<w Users' ;
    KEY ALT+W, ''
DEFINE BAR 4 OF cardinfo PROMPT '\<-'
DEFINE BAR 5 OF cardinfo PROMPT '\<Charges '
DEFINE BAR 6 OF cardinfo PROMPT '\<-'
DEFINE BAR 7 OF cardinfo PROMPT 'E\<xit '

```

BAR()

*** Here is where the cardinfo menu pad uses the BAR() function
*** to pass a bar number to the procedure called choice below.

```
ON SELECTION POPUP cardinfo;  
    DO choice IN bar_exam WITH BAR( ), POPUP( )
```

*** The procedure choice uses the mbar parameter
*** to contain the value passed by the BAR() function.

```
PROCEDURE choice  
PARAMETERS mbar, mpopup
```

```
WAIT WINDOW 'You chose bar #' + LTRIM(STR(mbar)) +  
            ' from popup ' + mpopup NOWAIT
```

```
IF mbar = 7 AND mpopup = 'CARDINFO'  
    SET SYSMENU TO DEFAULT  
ENDIF
```


BETWEEN()

Purpose	Determines if the value of an expression lies between the values of two other expressions of the same date type
Syntax	BETWEEN(<expr1>, <expr2>, <expr3>)
Parameters	<expr1> The expression whose value is being tested <expr2> The expression whose value is the low end of the range <expr3> The expression whose value is the high end of the range
Returns	Logical
See Also	MAX(), MIN()

Description BETWEEN() returns a value of true (.T.) if the value of a character, numeric or date expression lies between the values of two other expressions of the same data type. If the value of the expression doesn't lie between the values of two other expressions, BETWEEN() returns false (.F.).

Parameters <expr1>, <expr2>, <expr3>
 True (.T.) is returned if the value of <expr1> is greater than or equal to the value of <expr2> and less than or equal to the value of <expr3>; otherwise, false (.F.) is returned.

Example This example scans the INVOICES table/DBF for all records with invoice numbers between 1250 and 1260 inclusive and then displays the invoices and their respective totals.

```
CLOSE DATABASES
USE invoices
SCAN FOR BETWEEN(ino,1250,1260 )
  ? 'Itotal for ' + ALLTRIM(STR(ino)) + ' is ' + ;
    ALLTRIM(STR(itotal))
ENDSCAN
```

BOF()

Purpose	Determines if the record pointer is positioned at the beginning of a table/.DBF
Syntax	BOF([<expN> <expC>])
Parameters	<expN> The work area number <expC> The table/.DBF or work area alias
Returns	Logical
See Also	EOF()

Description Use BOF() to test for a beginning-of-file condition for a table/.DBF. BOF() returns true (.T.) if you have tried to move the record pointer to a position before the first logical record in the table/.DBF.

Parameters **<expN> | <expC>**

BOF() supports optional parameters that specify the work area in which BOF() is tested. If you omit <expN> and <expC>, the record pointer position for the selected work area is returned.

You can test the beginning-of-file condition in a table/.DBF open in another work area by specifying the work area number <expN> or the table/.DBF alias <expC>.

If a table/.DBF isn't open in the indicated work area, BOF() returns false (.F.).

Example This example opens CUSTOMER.DBF and displays a small window with push buttons you can use to move the record pointer forward or backward through the table/.DBF. As you move the cursor forward or backward, FoxPro moves the record pointer. Depending on the direction the pointer moves, FoxPro checks to see if the record pointer is at the end of the file (EOF) or at the beginning of the file (BOF). If either condition is true, FoxPro displays the appropriate message and repositions the record pointer.

```
CLOSE DATABASES
SET TALK OFF
CLEAR
USE customer
ACTIVATE SCREEN
```

```

@ 2,1 SAY 'Company: ' + company
DEFINE WINDOW mskip FROM 9,10 TO 13,65 DOUBLE COLOR SCHEME 5
ACTIVATE WINDOW mskip

@ 1,5 GET mchoice FUNCTION '*HN \<Forward;\<Backward;\<Quit' ;
    DEFAULT 0 SIZE 1,14 VALID CHOICE( )

READ CYCLE
RELEASE WINDOW mskip

PROCEDURE choice
DO CASE
    CASE mchoice = 1
        SKIP
        IF EOF('CUSTOMER')
            WAIT WINDOW 'End of file encountered' NOWAIT
            SKIP -1
        ENDIF
    CASE mchoice = 2
        SKIP -1
        IF BOF('CUSTOMER')
            WAIT WINDOW 'Beginning of file encountered' NOWAIT
            GO TOP
        ENDIF
    CASE mchoice = 3
        CLEAR READ
ENDCASE
ACTIVATE SCREEN
@ 2,1 SAY 'Company: ' + company
RETURN .T.

```

BROWSE

Purpose	Opens the Browse window
Syntax	BROWSE [FIELDS <field list>] [FONT <expC1> [, <expN1>]] [STYLE <expC2>] [FOR <expL1>] [FORMAT] [FREEZE <field>] [KEY <expr1> [, <expr2>]] [LAST] [LEDIT] [REDIT] [LOCK <expN2>] [LPARTITION] [NOAPPEND] [NOCLEAR] [NODELETE] [NOEDIT NOMODIFY] [NOLGRID] [NORGRID] [NOLINK] [NOMENU] [NOOPTIMIZE] [NOREFRESH] [NORMAL] [NOWAIT] [PARTITION <expN3>] [PREFERENCE <expC3>] [REST] [SAVE] [TIMEOUT <expN4>] [TITLE <expC4>] [VALID [:F] <expL2> [ERROR <expC5>]] [WHEN <expL3>] [WIDTH <expN5>] [[WINDOW <window name1>] [IN [WINDOW] <window name2> IN SCREEN]] [COLOR SCHEME <expN6> COLOR <color pair list>]
Remarks	FONT and STYLE clauses, RGB color pairs — FoxPro for Window only
See Also	CHANGE, EDIT, SET SKIP, WTITLE()

Description BROWSE is one of the most useful commands available in FoxPro. Use BROWSE to open a Browse window and display records from a table/.DBF. You can then easily edit and append records. You can have a Browse window open for each work area if you have sufficient memory. For further information on the use of the Browse window, see the FoxPro *User's Guide*.

Navigating in a Browse Window

You can move from record to record with the Up and Down Arrow keys or the vertical scroll bar. You can move forward or backward through a group of records with the PgUp or PgDn key. You can move to the next or previous field by pressing Tab or Shift+Tab or by using the horizontal thumb.

For additional information on navigating in a Browse window, see the Database Menu chapter in the FoxPro *User's Guide*.

Saving Your Changes

In FoxPro for MS-DOS, you can save any changes you make to your data and exit the Browse window by pressing Ctrl+W or Ctrl+End, or by clicking on the close box. If you press Ctrl+Q or Escape, you can exit the Browse window without saving changes you make to the current field.

In FoxPro for Windows, you can save any changes you make to your data and exit BROWSE by pressing Ctrl+W or Ctrl+End, or by choosing **Close** from the Browse window **Control** menu. Press Escape to exit the Browse window without saving changes you make to the current field.

In a program, use DEACTIVATE WINDOW to save your changes and close a Browse window. Include the name of the Browse window in DEACTIVATE WINDOW. See WTITLE() in this chapter for more information on Browse window names.

You can edit memo fields in the Browse window by placing the cursor in the memo field and pressing Ctrl+PgDn or double-clicking on the memo field. To exit the memo field and save your changes, press Ctrl+W or click the on close box. To exit the memo field without saving your changes, press Escape.

BROWSE and Related Files

With FoxPro you can establish relationships between tables/.DBFs. If a relationship is established, when the record pointer is moved in the parent table/.DBF, the record pointer is automatically positioned on the corresponding record in the child table/.DBF. You can open a Browse window for the parent window and each of the child tables/.DBFs. When you move to a record in the parent Browse window, the corresponding record or records are displayed in the child Browse window.

In a single Browse window you can include fields from both the parent table/.DBF and any child tables. The **FIELDS** clause described in further detail below enables you to include a list of fields from the parent and child tables/.DBFs.

Select the work area of the parent table/.DBF. Then issue **BROWSE** with the **FIELDS** clause. When you include fields from child tables/.DBFs in the field list, preface the child field names with their table/.DBF alias and a period.

For additional information on creating relationships between tables/.DBFs, see **SET RELATION** in this chapter.

SET SKIP Support

SET SKIP lets you establish a one-to-many relationship between two tables/.DBFs. For each record in the parent table/.DBF, there can be multiple related records in the child table/.DBF. If you create a one-to-many relationship, you may use **BROWSE** to view records from both the parent and child tables/.DBFs.

For each record from the parent table/.DBF that FoxPro displays in the Browse window, all related records (if any) from the child table/.DBF are displayed. The parent record is displayed once along with the first matching record from the child table/.DBF. Any subsequent matching records are displayed in the rows following the parent record and first matching child record. In FoxPro for MS-DOS, shaded blocks are displayed in any column containing information from the parent table/.DBF beyond the first matching record. In FoxPro for Windows the fill character for repeated parent information depends on the current Browse window font.

If the record pointer is positioned on a parent record, you can move the pointer between each of the parent records in the Browse window by pressing **Ctrl+↓** to move to the next parent record or **Ctrl+↑** to move to the previous parent record. For additional information on creating one-to-many relationships, see **SET SKIP** in this chapter.

Example This example uses SET SKIP to create a one-to-many relationship between two tables/.DBFs. There is a single record in the parent table/.DBF (OFFICES) for each invoice created. The child table/.DBF (CUSTOMER) contains multiple records for each record in the parent table/.DBF. After the relationship is created, a Browse window displaying records from both the parent and child tables/.DBFs is opened.

The FIELDS clause fields list contains records from both the parent and child tables/.DBFs. The names of fields are prefaced with their table/.DBF alias (CUSTOMER or OFFICES) and a period.

```
CLEAR
CLOSE DATABASES
USE offices INDEX offices ORDER ono IN 0  && Parent table/.DBF
USE customer INDEX customer ORDER ono IN 0 && Child table/.DBF
SELECT offices          && Back to parent work area
SET RELATION TO ono INTO customer && Establish relationship
SET SKIP TO customer    && One-to-many relationship
WAIT WINDOW 'Scroll to see companies for each office' NOWAIT

BROWSE FIELDS offices.ono :H='Office Number', ;
        offices.city :H='Office City', customer.company
```

Useful Functions

Several FoxPro functions return useful information about a Browse window. VARREAD() can be used to determine the current field in a Browse window. When a Browse window is active, VARREAD() returns the name of the field the cursor is positioned in.

RECNO() returns the record number of the selected record. ROW() and COL() return the row and column position of the cursor in the Browse window.

BROWSE supports numerous options. A list of the available options and a brief description of each follow.

Clauses FIELDS <field list>

If FIELDS is included, the listed fields are displayed in the order specified in <field list>. If FIELDS isn't included, all fields in the table/.DBF are displayed in the order they appear in the table/.DBF structure.

You can include fields from other related tables/.DBFs in the field list. When you include a field from a related table/.DBF, preface the field name with its table/.DBF alias and a period.

The field list can specify any combination of fields or calculated fields.

The syntax of the field list is:

```
<field1> [:R] [:column width]
[:V = <expr1> [:F] [:E = <expC1>]]
[:P = <expC2>]
[:B = <expr2>, <expr3> [:F]]
[:H = <expC3>]
[:W = <expL1>]
```

```
[, <field2> [:R] ... ]
```

Calculated Fields

The field list can contain statements for creating calculated fields. A calculated field contains read-only data created with an expression. This expression can take any form, but it must be a valid FoxPro expression.

The format of the statement you use to create a calculated field is:

```
<calculated field name> = <expr>
```

This example creates a calculated field called LOCATION:

```
CLOSE DATABASES
USE customer
```

```
BROWSE FIELD LOCATION = ALLTRIM(city) + ', ' + state
```

CITY and STATE are the names of fields from the selected table/.DBF.

The FIELDS clause fields list includes eight options, which enable special handling of fields displayed in a Browse window:

:R

The read-only option (:R) specifies that the field is read-only — the data it contains can be viewed but not edited.

In the following example, a Browse window is opened with the CNO and COMPANY fields. The CNO (customer number) field is read-only and cannot be changed.

```
CLOSE DATABASES
USE customer
BROWSE FIELD cno:R, company
```


:<column width>

The width clause specifies the display size for a field in columns. The value of this option doesn't affect the size of the field in the table/.DBF; it alters only the way the field is displayed in the Browse window.

:V = <expr1>

The verify option (:V) lets you perform field-level data validation within the Browse window. If <expr1> evaluates to true (.T.) when you move the cursor from a field, the data input into the field is considered correct and the cursor moves to the next field.

If <expr1> evaluates to false (.F.), the data input is considered incorrect, the cursor remains in the field and a message is displayed. If <expr1> evaluates to 0, the data input is considered incorrect and the cursor remains in the field but no error message is displayed.

By default, <expr1> is evaluated only when the field is modified. To force verification, include the :F option.

You can display your own error message by including the :E option described below. The verify option isn't executed for memo fields.

:F

The forced validation option (:F) determines if the expression in the verify option is evaluated when you move the cursor out of a field. If :F is not included, <expr1> is evaluated only if changes are made to the field. If :F is included, <expr1> is evaluated even if the field isn't modified.

:E = <expC1>

If the validation expression :V = <expr1> evaluates to true, the cursor leaves the field normally. If the expression evaluates to false, the cursor remains in the field, and the system error message "Invalid input" is displayed.

If the error option (:E) is included, <expC1> is displayed instead of the system error message. <expC1> is displayed only if SET NOTIFY is ON. The bell is sounded if BELL is SET ON.

If :V = <expr1> evaluates to 0, no message is displayed and the cursor remains in the field being validated. This lets you display your own error messages in validation routines.

The following example opens the CUSTOMER table/.DBF and displays the CNO and COMPANY fields. The CNO (customer number) field is a character field which will accept up to 5 characters. For purposes of this example, we consider a customer number less than 14000 to be invalid.

:V specifies the validation criteria. :F forces the validation check be performed whether the data is changed or not. :E replaces the FoxPro system error message with a user-defined error message. In FoxPro for Windows the error message is displayed in the status bar at the bottom of the main FoxPro window.

Press Escape to close the Browse window.

```
CLOSE DATABASES
IF _WINDOWS
    SET STATUS BAR ON
ENDIF
USE customer
BROWSE FIELDS cno :V = cno > '14000' ;
    :F ;
    :E = 'The customer number must be greater than 14000', ;
    company
```

:P = <expC2>

If you include a FIELDS clause, you can also specify a picture option (:P) for each field in the list. The picture option lets you create a list of codes that controls the display and input of data for each field in a Browse window. <expC2> is the list of codes.

For further information on the use of picture editing codes, see @ ... SAY and @ ... GET. With the exception of the M code, all of the function and picture codes available in @ ... SAY and @ ... GET can be used in PICTURE and FUNCTION clauses.

The following example uses the picture option to allow only numeric data in a specific format to be entered in the YTDPURCH field:

```
CLOSE DATABASES
USE customer

BROWSE FIELDS ytdpurch :P = '999999.99'
```

:B = <expr2>, <expr3> [:F]

With this option, you can specify a set of boundaries between which data in a field must fall. The boundary expressions <expr2> and <expr3> must match the data type of the field and *cannot* be the names of user-defined functions. If the data entered doesn't fall between <expr2> and <expr3>, a system error message is displayed indicating the range between which the data must fall.

By default, the data you enter is checked against the boundary values only if you make a change to the contents of the field. To force checking against the boundary values, include the forced validation option (:F).

The following example ensures that the YTDSALES data is between 100 and 99,999. Press Escape to close the Change window.

```
CLOSE DATABASES
USE salesman
BROWSE FIELDS ytdsales :B = 100, 99999 ;
      :F, name
```

:H = <expC3>

By default, field names are used as the column headings in the Browse window. Including the heading option (:H) replaces the default field names with your own headings, which you specify with <expC3>.

The following example provides user-defined headings for the displayed fields.

```
CLOSE DATABASES
USE customer
BROWSE FIELDS ytdpurch :H = 'Purchases:', ;
      company :H = 'Company:'
```

:W = <expL1>

You can conditionally prohibit moving the cursor to a field based on the value of the logical expression <expL1>. If <expL1> evaluates to false (F.), moving the cursor to the field is prohibited. If <expL1> evaluates to true (T.), the cursor can be moved to the field. User-defined functions are supported in <expL1>.

If moving the cursor to all fields is prohibited, the current field is marked read-only. This occurs only when every field contains a WHEN clause that evaluates to false.

FONT <expC1> [, <expN1>] — FoxPro for Windows only

The character expression <expC1> is the name of the font, and the numeric expression <expN1> is the font size. For example, the following clause specifies 16 point Roman font for the fields displayed in a Browse window:

```
FONT 'ROMAN',16
```

If you include the FONT clause but omit the font size <expN1>, a 10 point font is used in the Browse window. If you omit the FONT clause, 8 point MS Sans Serif is used.

If the font you specify is not available, Windows substitutes a font with similar font characteristics.

STYLE <expC2> — FoxPro for Windows only

In FoxPro for Windows, include the STYLE clause to specify a font style for fields displayed in the Browse window. The styles that are available for a font are determined by Windows. If the font style you specify is not available, Windows substitutes a font style with similar characteristics.

The font style is specified with <expC2>. If the STYLE clause is omitted, the standard font style is used.

Character	Font Style
B	Bold
I	Italic
N	Normal
O	Outline
S	Shadow
—	Strikeout
U	Underline

You can include more than one character to specify a combination of font styles. The following example open a Browse window and uses a 15 point underlined System font:

```
CLOSE DATABASES
USE customer
IF _WINDOWS
    BROWSE FIELDS contact FONT 'System', 15 STYLE 'NU'
ENDIF
```

FOR <expL1>

Including the FOR clause lets you conditionally display records in a Browse window, filtering out undesired records. If you include the FOR clause, only records for which the logical expression <expL1> is true are displayed in the Browse window.

Rushmore optimizes a query specified with a BROWSE FOR if <expL1> is an optimizable expression. For best performance, use an optimizable expression in the FOR clause. A discussion of Rushmore optimizable expressions appears in the Optimizing Your Application chapter in the *FoxPro Developer's Guide*.

FORMAT

Include the FORMAT clause to use a format file to control the display and data entry format in a Browse window. A format file must first be opened with SET FORMAT. The following information is extracted from the format file and applied to the Browse window:

- The list of fields to browse
- All VALID clauses
- All WHEN clauses
- All RANGE clauses
- Field sizes (as specified in PICTURE clauses)
- All SAY expressions (included as calculated BROWSE fields)

The following example uses a format file to validate data entered into a Browse window. Locations specified with @ ... GET can be the same and, in the format file, are ignored.

The first line creates a BROWSE field (CNO) that is 5 characters wide and allows the entry of letters and digits only. The second line creates a BROWSE field (COMPANY) that cannot contain a blank value and can contain a maximum of 20 alphabetic characters.

The third line creates a BROWSE field (CONTACT) into which you enter data only when the field is blank. The fourth line creates a BROWSE field (YTDPURCH) that displays the year-to-date purchases. The last line creates a calculated field (indicated by @ ... SAY) that displays the contents of the CITY and STATE fields with all blanks removed in a field 20 characters wide.

```
USE customer
SET FORMAT TO custentr.fmt
BROWSE FORMAT
```

Here are the contents of the CUSTENTR.FMT format file, which is used to validate data entered into CUSTOMER table/.DBF:

```
@ 3,0 GET cno PICTURE 'NNNN'
@ 3,0 GET company VALID company != SPACE(40) ;
    PICTURE 'AAAAAAAAAAAAAAAAAAAA'
@ 3,0 GET contact WHEN contact = SPACE(40)
@ 3,0 SAY ytdpurch
@ 3,0 SAY ALLTRIM(city) + ', ' + ALLTRIM(state) ;
    PICTURE 'XXXXXXXXXXXXXXXXXXXX'
```

FREEZE <field>

Include the FREEZE clause to permit changes to be made to only one field in the Browse window. You specify this field with <field name>. The remaining fields are displayed and cannot be edited.

```
CLOSE DATABASES
USE customer
BROWSE FIELDS ytdpurch :H = 'Purchases:' , ;
    company :H = 'Company:' ;
    FREEZE ytdpurch
CLOSE DATABASES
```

KEY <expr1> [, <expr2>]

Use the KEY clause to limit the scope of records that are displayed in the Browse window. With KEY you can specify an index key value (<expr1>) or a range of key values (<expr1>, <expr2>) for the records that are displayed in the Browse window. The table/.DBF you browse must be indexed, and the index key value or values included in the KEY clause must be the same data type as the index expression of the master index file or master tag.

For example, the CUSTOMER table/.DBF includes a character field containing zip codes. If the table/.DBF is indexed on the zip code field, you can specify a range of zip codes in the KEY clause.

In this example, only records with zip codes falling within the range of 43000 to 43999 are displayed in the Browse window:

```
CLOSE DATABASE
USE customer
SET ORDER TO zip
BROWSE KEY '43000', '43999'
```

LAST

Any configuration changes made to the appearance of a Browse window are saved in the FOXUSER resource file when the Browse window is closed. The field list, the size of each field, and the location and size of the Browse window are stored in the FOXUSER file. For additional information this file, see SET RESOURCE in this chapter.

If you issue BROWSE with the LAST clause, the Browse window opens in the same configuration that was last saved in the FOXUSER file if RESOURCE is SET ON. This restores the previous Browse window configuration created with the last BROWSE command. If the last BROWSE issued in the Command window included a long list of clauses, issue BROWSE with the LAST option to avoid having to retype the command.

If the last Browse window was opened with a BROWSE that included a PREFERENCE clause, BROWSE LAST won't restore the preference.

Any Browse window configuration changes you make in the current session aren't saved if you exit BROWSE by pressing Ctrl+Q.

**LEDIT
REDIT**

Information in a Browse window can be viewed in two modes: Change or Browse. In the default Browse mode, fields are displayed horizontally. In Change mode, fields are displayed vertically.

You can split a Browse window into left and right partitions by moving the window splitter with the mouse, by choosing **Resize Partitions** from the **Browse** menu or by including PARTITION in the BROWSE. Each partition can be viewed in Change or Browse mode independently of the other partition.

If LEDIT or REDIT is included in BROWSE, the left or right partition is in Change mode when the Browse window is displayed. Include both key words to open both partitions in Change mode. Each partition can be toggled between Change and Browse modes by choosing **Browse** and **Change** from the **Browse** menu popup.

```
CLOSE DATABASE
USE customer
BROWSE PARTITION 30 LEDIT
```

LOCK <expN2>

Include **LOCK** to specify the number of fields displayed in the left partition of the Browse window. The number of fields displayed is specified with <expN2>. The first <expN2> fields (from left to right) in the Browse window are placed in their own partition.

For more information about Browse window partitions and the features that are available for a split Browse window, see the *FoxPro User's Guide*.

LPARTITION

A Browse window can be split into left and right partitions by including the **PARTITION** clause described later. By default, the cursor is placed in the first field in the right partition when the Browse window is opened. If you include **LPARTITION**, the cursor is placed in the first field in the left Browse window partition.

The cursor is placed in the right Browse window partition if you include **LPARTITION** without the **PARTITION** clause.

NOAPPEND

If you include **NOAPPEND** in **BROWSE**, you can't add records to the table/.DBF by pressing Ctrl+N or choosing **Append Record** from the **Browse** menu.



Including **NOAPPEND** doesn't prevent you from appending a record from within a routine (created with **VALID**, **WHEN** or **ON KEY LABEL**) while in the Browse window.

NOCLEAR

If **NOCLEAR** is included in **BROWSE** and you close the Browse window, an image of the Browse window remains displayed. This image isn't an active Browse window. Use **CLEAR** to remove the Browse window image.

NOCLEAR is ignored in FoxPro for Windows.

NODELETE

Including NODELETE prevents records from being marked for deletion from within a Browse window. By default, a record can be marked for deletion by pressing Ctrl+T, choosing **Toggle Delete** from the **Browse** menu, or clicking in the leftmost column of the record.



Including NODELETE doesn't prevent you from marking a record for deletion from within a routine (created with VALID, WHEN or ON KEY LABEL) while in the Browse window.

NOEDIT | NOMODIFY

Including NOEDIT or NOMODIFY prevents you from modifying the table/.DBF. NOEDIT and NOMODIFY are identical. If you include either clause, you can browse or search the table/.DBF, but you cannot edit it. However, you can append and delete records. Include NOAPPEND and NODELETE to prohibit the addition and deletion of records.

NOLGRID NORGRID

The fields in a Browse window are separated by vertical lines. You can remove the vertical field grid lines from the left or right partitions of the Browse window by including NOLGRID or NORGRID in BROWSE. To remove the field grid lines in the left partition of the Browse window, include NOLGRID. To remove the grid lines in the right partition, include NORGRID.

You can also toggle grid lines on or off by choosing **Grid** from the **Browse** menu.

NOLINK

By default, the left and right partitions of the Browse window are linked together: when you scroll through one partition, the other partition scrolls. You can unlink the partitions by including NOLINK. You can also unlink partitions by choosing **Unlink Partitions** from the **Browse** menu.

NOMENU

Including NOMENU in BROWSE prevents access to the **Browse** menu and its options by removing the **Browse** menu pad from the system menu bar.

NOOPTIMIZE

To disable Rushmore optimization of BROWSE, include NOOPTIMIZE. For further information on the NOOPTIMIZE clause, see SET OPTIMIZE in this chapter, or consult the *Optimizing Your Application* chapter in the FoxPro *Developer's Guide*.

NOREFRESH

Including NOREFRESH prevents the Browse window from being refreshed. Browse windows are refreshed at the rate determined by SET REFRESH. NOREFRESH is useful with read-only files and improves performance.

NORMAL

When the current output window is a user-defined window, the Browse window derives its colors, size, position, title and control options (GROW, FLOAT, ZOOM, and so on) from the user-defined window. If NORMAL is specified, the Browse window is opened normally with its default settings as if there were no active user-defined window.

NOWAIT

When BROWSE is issued from within a program, a Browse window is opened and program execution pauses until the Browse window is closed. Including NOWAIT continues program execution immediately after the Browse window is opened. The program doesn't wait for the Browse window to be closed but continues executing on the program line immediately following the program line containing BROWSE NOWAIT.

This option is available only from within a program. Including NOWAIT in BROWSE has no effect when the command is issued in the Command window.

PARTITION <expN3>

A Browse window can be split into left and right partitions by including PARTITION in BROWSE. The numeric expression <expN3> specifies the column in which the window splitter is located. For example, if <expN3> is 20, the window splitter is placed in column 20 of the Browse window.

A Browse window can also be split into left and right partitions by moving the window splitter with the mouse or by choosing **Resize Partitions** from the **Browse** menu.

PREFERENCE <expC3>

Include PREFERENCE in BROWSE to save a Browse window's attributes and options for later use. Unlike LAST, which restores the Browse window as it appeared in the previous session, PREFERENCE saves a Browse window's attributes indefinitely in the FOXUSER resource file. Preferences can be retrieved at any time. For additional information about the FOXUSER resource file, see SET RESOURCE in this chapter.

Issuing BROWSE with the specified preference name <expC3> for the first time creates an entry in the FOXUSER file that saves the Browse window configuration. Issuing BROWSE later with the same preference name restores the Browse window to that preference state. When the Browse window is closed, the preference is updated.



Once you have a preference the way you like it, you can prevent it from being changed. Close the Browse window, issue SET RESOURCE OFF, open the FOXUSER file as a table/.DBF, and change the field containing the preference to read-only by changing the value of the logical field READONLY to true (.T.).

If you exit a Browse window by pressing Ctrl+Q, any Browse window changes aren't saved to the resource file.

REST

By default, BROWSE positions the record pointer at the top of the table/.DBF. Include REST to prevent the record pointer from being moved from its current position to the top of the table/.DBF.

SAVE

Include SAVE with BROWSE to keep the Browse window and any of its memo field text editing windows active and visible (open). You can then return to the Browse window after cycling through other open windows with the keyboard or the mouse.

This option is available only from within a program. SAVE has no effect when included with BROWSE in the Command window because BROWSE SAVE is always the default in the interactive mode.

TIMEOUT <expN4>

Include TIMEOUT to specify how long a Browse window waits for input. The numeric expression <expN4> specifies how many seconds can elapse without any input before the Browse window automatically closes.

TIMEOUT is available only from within a program; it has no effect when issued in the Command window.

TITLE <expC4>

By default, the name of the table/.DBF being browsed appears in the top border of the Browse window. If you issue BROWSE WINDOW <window name1> and the window has a title, the Browse window assumes <window name1> as its title.

You can overwrite the table/.DBF name or window title with your own title by including a TITLE clause. The character expression <expC4> is the new title you want to display in the top border of your Browse window.

```
DEFINE WINDOW mybrowse FROM 1,1 TO 24,40 ;
    CLOSE ;
    GROW ;
    COLOR SCHEME 10
```

```
CLOSE DATABASES
USE customer
BROWSE WINDOW mybrowse ;
    FIELDS ytdpurch :H = 'Purchases' , ;
    company :H = 'Company:' ;
    TIMEOUT 3
```

```
BROWSE WINDOW mybrowse ;
    TITLE 'Mybrowse Window' ;
    FIELDS ytdpurch :H = 'Purchases' , ;
    company :H = 'Company:' ;
    TIMEOUT 3
```

```
RELEASE WINDOW mybrowse
```

VALID [:F] <expL2> [ERROR <expC5>]

Including VALID lets you perform record-level validation in a Browse window. VALID is executed only if a change is made to the record and you move the cursor to another record. The VALID clause is *not* executed if the only change is to a memo field.

If VALID returns a true value (.T.), you can move the cursor to another record.

If VALID returns a false value (.F.), the cursor remains in the current field, and the system message “Invalid input” is displayed. You can display your own error message when VALID returns false by including the ERROR clause. The character expression <expC5> is displayed as the error message.

If VALID returns 0, the cursor remains in the current field, and an error message isn’t displayed.

You can force the VALID clause to execute before you move the cursor to the next record by including :F after the VALID key word. In this case, VALID is executed even if the record isn’t changed.

The VALID clause shouldn’t be confused with the verify option (:V), which enables *field-level* validation.

WHEN <expL3>

The WHEN clause is executed when you move the cursor to another record. If <expL3> evaluates to a true value (.T.), you can make changes to the record you move the cursor to. If <expL3> evaluates to false or 0, the record you move the cursor to becomes read-only and cannot be modified.

WIDTH <expN5>

The WIDTH clause limits the number of characters displayed for *all* fields in a Browse window to <expN5>. The contents of a field can be scrolled horizontally using the Left and Right Arrow keys or the horizontal scroll bar. Including the WIDTH clause doesn’t change the size of fields in the table/DBF; it alters only the way the fields are displayed in the Browse window. If a width has been specified for an individual field with the FIELDS clause, it overrides the width specified with the WIDTH clause for that field.

WINDOW <window name1>

If you include the WINDOW clause in BROWSE, the Browse window takes on the characteristics of the window specified with <window name1>. For example, if the window is defined with the FLOAT clause, the Browse window can be moved. The specified window doesn’t have to be active or visible, but it must be defined. In FoxPro for MS-DOS, if the window is defined with a footer, the footer doesn’t appear; the Browse window scroll bar takes its place.

IN [WINDOW] <window name2>

If the IN WINDOW clause is included, the Browse window is opened within the parent window specified with <window name2>. The Browse window doesn't assume the characteristics of the parent window. A Browse window activated inside a parent window cannot be moved outside the parent window. If the parent window is moved, the Browse window moves with it.

To access the Browse window, the parent window must first be defined with DEFINE WINDOW and must be active and visible.

IN SCREEN

By including IN SCREEN, you can explicitly place a Browse window on the desktop in FoxPro for MS-DOS or in the main window in FoxPro for Windows when a user-defined window is active.

COLOR SCHEME <expN6>**COLOR <color pair list>**

The color of a Browse window can be specified by including the number of an existing color scheme in the COLOR SCHEME clause or a set of color pairs in the COLOR clause.

A color scheme is a set of 10 predefined color pairs. The color pairs in a color scheme can be changed with SET COLOR OF SCHEME. In FoxPro for MS-DOS the color pairs in a color scheme can also be changed in the Color Picker.

A color pair is a set of two letters separated by a forward slash. The first color letter specifies the foreground color and the second letter specifies the background color.

For example, this color pair specifies a red foreground on a white background:

R/W

For a list of colors and their corresponding color letters, see the SET COLOR commands section in this chapter or the Color Table by Color Pair topic in the online help facility.

In FoxPro for Windows, a color pair can also be specified with a set of 6 RGB (Red Green Blue) color values separated by commas. The first 3 color values specify the foreground color and the second 3 color values specify the background color. The color values can range from 0 through 255.

In FoxPro for Windows, the R/W color pair in the example above can also be specified with this RGB color pair:

RGB(255,0,0,255,255,255)

In FoxPro for MS-DOS, the Browse window assumes the colors specified with COLOR SCHEME 10 (Browse) by default.

In FoxPro for Windows, the Browse window assumes the color scheme established using the Color option in the Microsoft Windows Control Panel.

The following table shows what element each color pair controls in FoxPro for MS-DOS.

Color Pair Number	Browse Window Element
1	Other records (In FoxPro for Windows, color pair 1 controls the color of the current record and field)
2	Current field
3	Border
4	Title color when Browse window is active
5	Title color when another window is active
6	Selected text
7	Current record
8	Shadow

In FoxPro for MS-DOS, the second color specified in color pair number 1 designates the color of the grid background. The second color specified in color pair number 3 is used for the grid foreground.

BUILD APP

Purpose	Creates an application file from a project file
Syntax	BUILD APP <app file> FROM <project file>
See Also	BUILD EXE, BUILD PROJECT, CREATE PROJECT, MODIFY PROJECT
Description	BUILD APP uses information from a project file to create an application file with an .APP file name extension. You create a project file with CREATE PROJECT or MODIFY PROJECT. A project file is a table/.DBF which uses the file extensions .PJX (table/.DBF file) and .PJT (memo file). Before using BUILD APP, be sure that the project file contains all of the files needed in the application. Unresolved references or other compile-time errors encountered during the build process are reported and/or stored in an error file. The first 8 characters of the error file name are the same as your project name and it has the file extension .ERR.
Clauses	FROM <project> FROM <project> specifies the name of the project from which the custom application is built. <app file> Specify the application file name with <app file>. The default file name extension is .APP. For more information on building applications, refer to the <i>FoxPro User's Guide</i>.

BUILD EXE

Purpose	For owners of the FoxPro 2.5 for MS-DOS Distribution Kit
Syntax	BUILD EXE <exe file> FROM <project> [STANDALONE] [EXTENDED]
Remarks	FoxPro for MS-DOS only
See Also	BUILD APP, BUILD PROJECT

Description BUILD EXE is used to create a compact or stand-alone executable .EXE file from a project. This command is available only to owners of the FoxPro Distribution Kit. The Distribution Kit must be installed in the same directory as FoxPro.

For additional information about creating executable files and BUILD EXE, see the FoxPro Distribution Kit manual.

Clauses **<exe file>**

<exe file> specifies the name of the .EXE file to create. If an APP application file with the same root name as the stand-alone .EXE file exists, it's deleted. Note that if an .EXE file exists and you create an APP file with the same name, the .EXE file is deleted.

FROM <project>

FROM <project> specifies the name of the project from which the executable file is built.

STANDALONE

STANDALONE creates a Standard version .EXE file that is completely self-contained and has the same requirements as the Standard version of FoxPro.

EXTENDED

EXTENDED creates an 32-bit Extended version .EXE file that's completely self-contained and has the same system requirements as the 32-bit Extended version of FoxPro. Creating an Extended version .EXE file requires considerably more time than does creating a Standard version .EXE file.

If you don't include either STANDALONE or EXTENDED, a compact .EXE file is created. To execute a compact .EXE file, the FoxPro .EXE Support Library must be located on the MS-DOS path.

BUILD PROJECT

Purpose	Creates and builds a project file
Syntax	BUILD PROJECT <project file> FROM <program> <menu> <report> <label> <screen> <library> [, <program> <menu> <report> <label> <screen> <library> ...]
See Also	BUILD APP, BUILD EXE, CREATE PROJECT, EXTERNAL, MODIFY PROJECT

Description BUILD PROJECT automatically creates a project table with a .PJX file name extension by opening and processing one or more program, menu, report, label, screen and library files that you specify. You can use the project file to create one of two program types: an application .APP file or an executable .EXE file. The project table/.DBF keeps track of all the files required to create an application, as well as the dependencies, references and connections among the files. Once you specify the pieces in the project, FoxPro makes sure that the application is based on the latest source files. For more information on building projects, refer to the Project Manager chapter in the *FoxPro User's Guide*.

When FoxPro encounters a program, menu or screen file while creating a project file from BUILD PROJECT, it finds its compiled file and compares the time and date stamp of the two files. If the time and date stamp of the source file is later than that of the compiled file, FoxPro recompiles the source file.

Each project file contains a time and date stamp so you can refresh the project file when you make changes to files in the project or when dependencies change. This helps to guarantee that any applications created from a project file will always use the most recent source files. To refresh a project file, issue BUILD PROJECT without the optional FROM clause and FoxPro updates the specified project.

When you issue BUILD PROJECT, unresolved references and other errors are reported but do not prevent the project file from being created. This enables you to build a project when all the necessary pieces aren't actually created or available at the time the project is built. Unresolved references or other problems can then be corrected by refreshing the project file at a later date or by manually modifying the information stored in the project file with MODIFY PROJECT.

Clauses <project file>

The name of the project table to be created is specified with <project file>.

FROM <program> | <menu> | <report> | <label> | <screen> | <library>
 [, <program> | <menu> | <report> | <label> | <screen> | <library>] ...

The FROM clause is optional and specifies the files to include in the project. You can specify one or more program, menu, report, label, screen and library files and the project will keep track of these files, as well as the dependencies, references and connections among them.

The first executable program or menu file in the FROM clause is the master program file in the project by default.

CALCULATE

Purpose	Performs financial and statistical operations on fields or expressions involving fields
Syntax	CALCULATE <expr list> [<scope>] [FOR <expL1>] [WHILE <expL2>] [TO <memvar list> TO ARRAY <array>] [NOOPTIMIZE]
See Also	AVERAGE, COUNT, DIMENSION, FV(), MAX(), MIN(), PAYMENT, PV(), SUM

Description Use **CALCULATE** to perform financial and statistical operations on fields in a table/.DBF or on expressions involving fields.

Clauses <expr list>

The expression list can contain any combination of the following functions:

AVG(<expN>)
 CNT()
 MAX(<expr>)
 MIN(<expr>)
 NPV(<expN1>, <expN2> [, <expN3>])
 STD(<expN>)
 SUM(<expN>)
 VAR(<expN>)

Functions in the expression list <expr list> are separated by commas. These functions are specific to **CALCULATE**. These functions are described in detail later in this section. They should not be confused with similarly named independent functions. For example, **CALCULATE** MIN() is not the same as MIN().

<scope>

The scope clauses are: ALL, NEXT <expN>, RECORD <expN>, and REST. These are explained in the Overview of the FoxPro Language chapter of this manual. Commands which include <scope> operate only on the table/.DBF in the active work area.

You can specify a scope of records used in the calculation. Only the records that fall within the range of records specified by the scope are included in the calculation.

The default scope for **CALCULATE** is ALL records.

FOR <expL1>

If the FOR <expL1> clause is included, only the records that satisfy the logical condition <expL1> are included in the calculation. Including a FOR clause conditionally includes records in the calculation, filtering out undesired records.

Rushmore optimizes a CALCULATE ... FOR query if <expL1> is an optimizable expression. For best performance, use an optimizable expression in the FOR clause. A discussion of Rushmore optimizable expressions appears in the Optimizing Your Application chapter in the *FoxPro Developer's Guide*.

WHILE <expL2>

If the WHILE clause is included, records are included in the calculation as long as the logical expression <expL2> evaluates to true (.T.).

TO <memvar list> | TO ARRAY <array>

Results of the calculation can be stored to one or more memory variables or to an array. If you include the TO clause to store a result to a memory variable, a new memory variable is created if one of the specified memory variables doesn't exist.

If you include the TO clause to store a result to an array, FoxPro automatically creates it if an array with the specified name doesn't exist. If the array exists and isn't large enough to contain all the results, the size of the array is automatically increased to accommodate the information. If an existing array is larger than necessary, additional elements are left unchanged. Results are stored to the array elements in the order they are specified in the CALCULATE command.

NOOPTIMIZE

To cause Rushmore not to optimize CALCULATE, include NOOPTIMIZE. For further information about NOOPTIMIZE, see SET OPTIMIZE.

Functions AVG <expN>

AVG() computes the arithmetic mean of <expN>.

Only records meeting the <scope> and/or optional FOR or WHILE conditions are included in the result.

CNT()

CNT() returns the number of records in the table/.DBF.

Only records meeting the <scope> and/or optional FOR or WHILE conditions are included in the result.

MAX(<expr>)

MAX() returns the highest or latest value of <expr>. Within the MAX() clause, you can specify any numeric, character or date field, or any expression using fields of these types.

Only records meeting the <scope> and/or optional FOR or WHILE conditions are included in the result.

MIN(<expr>)

MIN() returns the lowest or earliest value of <expr>. Any numeric, character or date field, or any valid expression using fields of these types can be included in <expr>.

Only records meeting the <scope> and/or optional FOR or WHILE conditions are included in the result.

NPV(<expN1>, <expN2> [, <expN3>])

NPV() computes the net present value of a series of future cash flows discounted at a constant periodic interest rate.

<expN1> is the interest rate expressed as a decimal value.

<expN2> is a field, field expression or a numeric expression representing a series of cash flows. Each cash flow can be either positive or negative. In cases where <expN2> is a field, the value in that field for each record in the table/.DBF is considered a cash flow.

<expN3> is an optional initial investment. If the initial investment isn't included, then the initial investment is assumed to occur at the end of the first period. This initial investment is the first record in the field and is negative to represent a cash outflow.

Only records meeting the <scope> and/or optional FOR or WHILE conditions are included in the result.

STD(<expN>)

STD() computes the standard deviation of <expN>. The standard deviation measures the degree to which the values of fields or expressions involving fields differ from the average of all the values. The smaller the standard deviation, the less the values vary from the average.

Only records meeting the <scope> and/or optional FOR or WHILE conditions are included in the result.

SUM(<expN>)

SUM() totals the values of <expN>.

Only records meeting the <scope> and/or optional FOR or WHILE conditions are included in the result.

VAR(<expN>)

VAR() computes the variance from the average of <expN>. The variance is the standard deviation squared. The smaller the variance, the less the values vary from the average.

Only records meeting the <scope> and/or optional FOR or WHILE conditions are included in the result.

Examples

```
CLOSE DATABASES
SET TALK ON
USE invoices
CALCULATE AVG(itotal), MIN(itotal), MAX(itotal)
      AVG(itotal) MIN(itotal) MAX(itotal)
      1198.01      15.48      3321.66

CALCULATE STD(itotal), VAR(itotal) TO mstd, mvar
?mstd,mvar
721.20  520123.68
```

CALL

Purpose	Executes a binary file that has been placed in memory with LOAD
Syntax	CALL <file> [WITH <expC> WITH <memvar>] [SAVE NOSAVE]
See Also	LOAD, RELEASE MODULE, SET LIBRARY TO

Description CALL works in conjunction with LOAD. With CALL you can run customized binary routines directly from memory.

LOAD places the binary file into memory. CALL executes the loaded binary file.

Clauses <file>

The binary file to be called is specified with <file>. It isn't necessary to specify an extension in the file name, since the extension is stripped from the file when it's loaded.

WITH <expC>| WITH <memvar>

WITH can be used to pass a character string or memory variable parameter to the binary subroutine. The memory variable can be of any data type.

When the called binary file executes, the code segment initially points to the beginning of the module. The DS and BX register pair points to the first byte of the parameter passed using WITH. If no parameter is passed, BX contains 0.

SAVE | NOSAVE

CALL and LOAD both support the SAVE key word for use with binary routines that write to the screen. NOSAVE is the default. You may include SAVE when loading a binary routine with CALL, or SAVE and NOSAVE when loading a binary routine in memory with LOAD. If specified with CALL, SAVE or NOSAVE overrides the SAVE or NOSAVE setting specified with LOAD.

If `CALL ... SAVE` is specified, FoxPro copies the current contents of video RAM into the FoxPro desktop upon return from the binary routine (assuming that the binary routine is writing directly into video RAM). This means that anything written by the binary routine is known to FoxPro and is treated as if FoxPro had written it. In particular, if `SAVE` is in effect, such externally written data will not be erased the first time a FoxPro object is dragged over it.

Unless it is needed, the `SAVE` clause isn't desirable since it takes FoxPro some time to scan and save the screen contents upon each return from the binary routine.



Carefully read the following assembly language interface warnings:

When using string variables in the binary file, do not change the length of the string. It is possible that the free memory list could be corrupted if the length of a string variable is changed by even one byte.

When saving a string address from one call to the next, you should also reset the string address you will be storing to just prior to using the string variable.

CANCEL

Purpose	Cancels execution of the current program file
Syntax	CANCEL
See Also	QUIT, RESUME, RETURN, SUSPEND

Description CANCEL aborts execution of a FoxPro program. FoxPro returns control to the Command window or to MS-DOS in FoxPro for MS-DOS, or to the Command window or to Windows in FoxPro for Windows.

If a program is executing in a runtime version of FoxPro for MS-DOS, the program terminates and control returns to MS-DOS. If a program is executing in the development version of FoxPro for MS-DOS, control returns to the Command window.

If a program is executing in a runtime version of FoxPro for Windows, the program terminates and control returns to Windows. If a program is executing in the development version of FoxPro for Windows, control returns to the Command window.

Executing CANCEL releases all private memory variables.

Example In this example, the memory variable MPASS is tested to see if it contains the character string FOXPRO. If an invalid password is entered, the program is terminated with CANCEL.

```
SET TALK OFF
DO WHILE .T.
    CLEAR
    STORE SPACE(6) TO mpass
    @ 8,29 SAY 'Password '
    @ 8,39 GET mpass
    READ

    IF UPPER(mpass) = 'FOXPRO'
        @ 18,25 SAY 'Password correct'
        CANCEL
    ELSE
        WAIT WINDOW 'Sorry! Password not accepted, Try "FoxPro"'
    ENDIF
ENDDO
```

CAPSLOCK()

Purpose	Returns the current mode of the CapsLock key or sets the CapsLock key mode on or off
Syntax	CAPSLOCK([<expL>])
Returns	Logical
See Also	INSMODE(), NUMLOCK()

Description CAPSLOCK() returns the current mode of the CapsLock key or toggles the CapsLock key mode on or off. Issuing CAPSLOCK() with no argument returns true (.T.) if CapsLock is on, or false (.F.) if CapsLock is off.

Parameter **<expL>**
 <expL> can be included to turn the CapsLock key on or off. CAPSLOCK(.T.) turns CapsLock on and CAPSLOCK(.F.) turns CapsLock off. A logical value is returned corresponding to the CapsLock setting *before* CAPSLOCK(.T.) or CAPSLOCK(.F.) is issued.

Example The following code stores the state of CAPSLOCK() to a memory variable. The = command executes the CAPSLOCK() function to set CapsLock on. Then the = command executes the CAPSLOCK() function to set CapsLock to its previous state.

```
oldlock = CAPSLOCK( )    && Set original setting
=CAPSLOCK(.T.)          && Turn it on

*** perform any number of statements ***

=CAPSLOCK(oldlock)      && Return to original setting

*** or, toggle CapsLock to the opposite value and back ***

=CAPSLOCK(!CAPSLOCK( ))
=CAPSLOCK(!CAPSLOCK( ))
```

CDOW()

Purpose	Returns the day of the week from a given date expression
Syntax	CDOW(<expD>)
Returns	Character
See Also	DAY(), DOW(), SYS()

Description CDOW(), the character day of the week function, returns from a date expression the name of the day of the week, for example, Sunday or Monday.

Parameter **<expD>**
Include the date expression <expD> for which CDOW() returns the day.

Examples STORE {02/16/56} TO bdate
 ? CDOW(bdate)
 Thursday

CDX()

Purpose	Returns the names of the open compound index file that has the specified index position number
Syntax	CDX(<expN1> [, <expN2> <expC>])
Parameters	<expN1> <expN2> <expC> A number corresponding to an index position The work area number for a table/.DBF The table/.DBF or work area alias
Returns	Character
See Also	INDEX, MDX(), NDX(), SET INDEX, SET FULLPATH, SYS(14), SYS(21), SYS(22), SYS(2021), TAG(), USE

Description CDX() returns the names of open .CDX files. The CDX() function is identical to the MDX() function.

A .CDX (compound) index consists of one physical file containing many tags. Each tag is an index order reference for the associated table/.DBF.

There are two types of .CDX files. A standard compound index (.CDX) may have a different name from its associated table/.DBF and can reside in a different directory than its associated table/.DBF. A table/.DBF can have multiple compound index files. You open a compound index with the INDEX clause of USE or with SET INDEX.

A structural .CDX must have the same name as its associated table/.DBF and reside in the same directory. A table/.DBF can have only one structural index file. Structural .CDXs are automatically opened and updated when the associated table/.DBF is opened with USE.

Any .IDX (FoxBASE+ and FoxPro 1.0 compatible index) files specified in USE or SET INDEX are ignored by the CDX() function.

Use TAG() to return individual tag names contained in a .CDX, and NDX() to return the name of open .IDX files.

Parameters <expN1>

For a table/.DBF with a structural compound index and one or more compound indexes:

- If <expN1> is 1, the name of the structural index file (which is always the same as the name of the table/.DBF) is returned.
- If <expN1> is 2, the name of the first compound index file name specified in the INDEX clause of USE or in SET INDEX is returned.
- If <expN1> is 3, the second compound index file name, if any, is returned and so on.
- If <expN1> is greater than the number of open .CDX files, the null string is returned.

For a table/.DBF with no structural compound index and one or more compound indexes:

- If <expN1> is 1, the name of the first compound index file name specified in the INDEX clause of USE or in SET INDEX is returned.
- If <expN1> is 2, the second compound index file name, if any, is returned and so on.
- If <expN1> is greater than the number of open .CDX files, the null string is returned.

<expN2> | <expC>

You can return the names of compound index files open in work areas other than the current work area by specifying either the work area number <expN2> or the table/.DBF or work area alias <expC>.

If you omit the optional <expN2> and <expC> arguments, names of compound index files are returned for the current work area.

CEILING()

Purpose	Returns the nearest integer that is greater than or equal to the specified numeric expression
Syntax	CEILING(<expN>)
Returns	Numeric
See Also	FLOOR(), ROUND()

Description CEILING() returns the nearest integer that is greater than or equal to a numeric expression. CEILING rounds a positive number with a fractional portion up to the next highest integer, and rounds a negative number with a fractional portion up to the integer portion of the number.

Parameter **<expN>**
Include a numeric expression <expN> in CEILING().

Example

```
STORE 10.1 TO num1
STORE -10.9 TO num2
STORE 10.0 TO num3
STORE -10.0 TO num4
```

```
? CEILING(num1)
11
? CEILING(num2)
-10
? CEILING(num3)
10
? CEILING(num4)
-10
```

CHANGE

Purpose	Displays fields for editing
Syntax	CHANGE [FIELDS <field list>] [<scope>] [FOR <expL1>] [WHILE <expL2>] [FONT <expC1> [, <expN1>]] [STYLE <expC2>] [FREEZE <field name>] [KEY <expr1> [, <expr2>]] [LAST] [LEDIT] [REDIT] [LPARTITION] [NOAPPEND] [NOCLEAR] [NODELETE] [NOEDIT NOMODIFY] [NOLINK] [NOMENU] [NOOPTIMIZE] [NORMAL] [NOWAIT] [PARTITION <expN2>] [PREFERENCE <expC3>] [REST] [SAVE] [TIMEOUT <expN3>] [TITLE <expC4>] [VALID [:F] <expL3> [ERROR <expC5>]] [WHEN <expL4>] [WIDTH <expN4>] [[WINDOW <window name1>] [IN [WINDOW] <window name2> IN SCREEN]] [COLOR SCHEME <expN5> COLOR <color pair list>]
Remarks	FONT and STYLE clauses, RGB color pairs — FoxPro for Windows only
See Also	BROWSE, EDIT, WTITLE()

Description CHANGE allows editing of the selected table/.DBF within a window. CHANGE functions identically to EDIT.

The Change window is also available from within the Browse window. For a complete description of using the Browse window in Change mode, see the *FoxPro User's Guide*.

Navigating in a Change Window

You can move from record to record with the Up and Down Arrow keys or move forward or backward through a group of records with the PgUp or PgDn key. To move to the next or previous field, press Tab or Shift+Tab.

Saving Your Changes

In FoxPro for MS-DOS, you can save any changes you make to your data and exit the Change window by pressing Ctrl+W or Ctrl+End, or by clicking on the close box. If you press Ctrl+Q or Escape, you can exit the Change window without saving changes you make to the current field.

In FoxPro for Windows, you can save any changes you make to your data and exit the Change window by pressing Ctrl+W or Ctrl+End, or by choosing **Close** from the Change window **Control** menu. Press Escape to exit the Change window without saving changes you made to the current field.

In a program, use DEACTIVATE WINDOW to save your changes and close a Change window. Include the name of the Change window in DEACTIVATE WINDOW. See WTITLE() in this chapter for more information on Change window names.

You can edit memo fields in the Change window by placing the cursor in the memo field and pressing Ctrl+PgDn or double-clicking on the memo field. To exit the memo field and save your changes, press Ctrl+W or click the close box. To exit the memo field without saving your changes, press Escape.

SET SKIP Support

SET SKIP lets you establish a one-to-many relationship between two tables/.DBFs. For each record in the parent table/.DBF, there can be multiple related records in the child table/.DBF. If you create a one-to-many relationship, you can use CHANGE to view records from both the parent and child tables/.DBFs.

For each record from the parent table/.DBF that FoxPro displays in the Change window, all related records (if any) from the child table/.DBF are displayed. The parent record is displayed once along with the first matching record from the child table/.DBF. Any subsequent matching records are displayed in the rows following the parent record and first matching child record. In FoxPro for MS-DOS, shaded blocks are displayed in any column containing information from the parent table/.DBF beyond the first matching record. In FoxPro for Windows the fill character for repeated parent information depends on the current Browse window font.

For additional information on creating one-to-many relationships, see SET SKIP in this chapter.

Example

This example uses SET SKIP to create a one-to-many relationship between two tables/.DBFs. There is a single record in the parent table/.DBF (OFFICES) for each invoice created. The child table/.DBF (CUSTOMER) contains multiple records for each record in the parent table/.DBF.

After the relationship is created, a Change window displaying records from both the parent and child tables/.DBFs is opened.

```
CLEAR
CLOSE DATABASES
USE offices INDEX offices ORDER ono IN 0 && Parent table/.DBF
USE customer INDEX customer ORDER ono IN 0 && Child table/.DBF
SELECT offices      && Back to parent work area
SET RELATION TO ono INTO customer && Establish relationship
SET SKIP TO customer && One-to-many relationship
WAIT WINDOW 'Scroll to see companies for each office' NOWAIT
CHANGE FIELDS offices.ono :H='Office Number', ;
      offices.city :H='Office City', customer.company
```

COL() and ROW() Support

Use COL() and ROW() to return the current screen row and column position of the cursor in a Change window. If a Change window is opened on the desktop in FoxPro for MS-DOS or the main FoxPro window in FoxPro for Windows, the returned cursor position is relative to the desktop or the main FoxPro window, not to the Change window itself. If a Change window is opened in a user-defined window, COL() and ROW() return the cursor position relative to the user-defined window.

CHANGE supports numerous options. A list of the available options and a brief description of each follow.

Clauses **FIELDS <field list>**

If FIELDS is included in CHANGE, the listed fields are displayed in the order they appear in <field list>. If FIELDS isn't included, all fields in the table/.DBF are displayed in the order they appear in the table/.DBF structure.

The field list can specify any combination of fields or calculated fields, including fields from tables/.DBFs open in other work areas. The syntax of the field list is:

```
<field1> [:R]
[:V = <expr1> [:F] [:E = <expC1>]]
[:P = <expC2>]
[:B = <expr2>, <expr3> [:F]]
[:H = <expC3>]
[:W = <expL1>]

[, <field2> [:R] ... ]
```

Calculated Fields

The field list can contain statements for creating calculated fields. A calculated field contains read-only data created with an expression. This expression may take any form but it must be a valid FoxPro expression.

The syntax of the statement you use to create a calculated field is:

```
<calculated field name> = <expr>
```

This example creates a calculated field called ADDRESS:

```
CLOSE DATABASES
USE customer

*** ADDRESS, CITY, and STATE ***
*** are the names of fields from the table/.DBF ***
CHANGE FIELD ADDRESS = ALLTRIM(address) + ' ' + ALLTRIM(city) ;
    + ', ' + state :H='Street,City,State'
```

The FIELDS clause fields list includes eight options, which enable special handling of fields displayed in the Change window.

:R

The read-only option (:R) specifies that the field included in the Change window is read-only — the data it contains can be viewed but not edited.

```
CLOSE DATABASES
USE customer
CHANGE FIELD cno:R, company
```

:V = <expr1>

Use the verify option (:V) to perform field-level data validation within the Change window. If <expr1> evaluates to true (.T.) when you move the cursor from the field, the data input into the field is considered correct and the cursor moves to the next field.

If <expr1> evaluates to false (.F.), the data input is considered incorrect, the cursor remains in the field and a message is displayed. If <expr1> evaluates to 0, the data input is considered incorrect, the cursor remains in the field, but no error message is displayed.

By default, <expr1> is evaluated only when the field is modified. To force verification, include the :F option.

You can display your own error message by including the :E option.

The verify option isn't executed for memo fields.

:F

The forced validation option (:F) determines if the expression in the verify option (<expr1>) is evaluated when you move the cursor from a field. If :F is not included, <expr1> is evaluated only if changes are made to the field. If :F is included, <expr1> is evaluated, even if the field isn't modified.

:E = <expC1>

If the validation expression :V = <expr1> evaluates to true, the cursor leaves the field normally. If the expression evaluates to false, the cursor remains in the field and the system error message "Invalid input" is displayed.

If the Error option (:E) is included, <expC1> is displayed instead of the system error message. <expC1> is displayed only if SET NOTIFY is ON. The bell is sounded if BELL is SET ON.

If the validation expression :V = <expr1> evaluates to 0, no message is displayed and the cursor remains in the field being validated. This lets you display your own error messages in validation routines.

Example

The following example opens SALESMAN.DBF and displays the CITY and YTDSALES fields. Enter a value of 0 in the YTDSALES field to perform the field validation.

:V is the validation criteria. :F forces the validation check to be performed whether the data is changed or not. :E replaces the FoxPro system error message with a user-defined error message.

In FoxPro for Windows the error message is displayed in the status bar at the bottom of the main FoxPro window.

Press Escape to close the Change window.

```
CLOSE DATABASES
USE salesman
CHANGE FIELD ono :R, city, ytdsales :V = ytdsales > 0 ;
      :F :E = 'Ytdsales must be greater than 0'
```

:P = <expC2>

If you include a FIELDS clause, you can also specify a picture option (:P) for each field in the list. The picture option lets you create an editing template specified with <expC2> that controls the display and input of data for each field in a Change window.

For further information on using picture editing codes, see @ ... SAY and @ ... GET. With the exception of the M code, all of the function and picture codes available in @ ... SAY and @ ... GET can be used in the PICTURE and FUNCTION clauses.

The following example uses the picture option to allow only numeric data in a specific format to be entered in the YTDSALES field:

```
CLOSE DATABASES
USE salesman
CHANGE FIELDS name, ytdsales :P = '999,999.99'
```

:B = <expr2>, <expr3> [:F]

With this option, you can specify a set of boundaries between which data must fall. The boundary expressions <expr2> and <expr3> must match the data type of the field and *cannot* be the names of user-defined functions. If the data entered doesn't fall between <expr2> and <expr3>, a system message is displayed indicating the range between which the data must fall.

By default, the data you enter is checked against the boundary values only if you make a change to the contents of the field. To force checking against the boundary values, include the forced validation option (:F).

The following example ensures that the YTDSALES data is between 100 and 99,999. Enter a value less than 100 in the YTDSALES field to check the value against the boundary values. Press Escape to close the Change window.

```
CLOSE DATABASES
USE salesman
CHANGE FIELDS ytdsales :B = 100, 99999 ;
:F, name
```

:H = <expC3>

By default, field names are placed to the left side of the fields in the Change window. Including the heading option (:H) replaces the default field names with your own headings, which you specify with <expC3>.

The following example provides user-defined headings for the displayed fields.

```
CLOSE DATABASES
USE customer
CHANGE FIELDS ytdpurch :H = 'Purchases:', ;
company :H = 'Company:'
```

:W = <expL1>

You can conditionally prohibit moving the cursor to a field based on the value of the logical expression <expL1>. The when option (:W) evaluates <expL1>. If <expL1> evaluates to false (.F.), you can't move the cursor to the field. If <expL1> evaluates to true (.T.), you can move the cursor to the field. User-defined functions are supported in <expL1>.

Moving the cursor to all fields is prohibited, the current field is marked read-only. This occurs only when every field contains a WHEN clause that evaluates to false.

<scope>

The scope clauses are: ALL, NEXT <expN>, RECORD <expN>, and REST. These are explained in the Overview of the FoxPro Language chapter of this manual. Commands which include <scope> operate only on the table/.DBF in the active work area.

You can specify a scope of records that are displayed in the Change window. Only the records that fall within the range of records specified by the scope appear in the Change window.

The default scope for CHANGE is ALL records.

FOR <expL1>

Including the FOR clause lets you conditionally display records in the Change window, filtering out undesired records. If you include the FOR clause, only the records that satisfy the logical condition <expL1> are displayed in the Change window.

Rushmore optimizes a CHANGE FOR query if <expL1> is an optimizable expression. For best performance, use an optimizable expression in the FOR clause. A discussion of Rushmore optimizable expressions appears in the Optimizing Your Application chapter in the *FoxPro Developer's Guide*.

WHILE <expL2>

If the WHILE clause is included in CHANGE, records are displayed in the Change window for as long as the logical expression <expL2> evaluates to true (.T.).

FONT <expC1> [, <expN1>] — FoxPro for Windows only

The character expression <expC1> is the name of the font, and the numeric expression <expN1> is the font size. For example, the following clause specifies 16 point Roman font for the fields displayed in a Change window:

```
FONT 'ROMAN',16
```

If you include the FONT clause but omit the font size <expN1>, a 10 point font is used in the Change window. If you omit the FONT clause, 8 point MS Sans Serif is used.

If the font you specify is not available, Windows substitutes a font with similar font characteristics.

STYLE <expC2> — FoxPro for Windows only

In FoxPro for Windows, include the STYLE clause to specify a font style for fields displayed in the Change window. The styles that are available for a font are determined by Windows. If the font style you specify is not available, Windows substitutes a font style with similar characteristics.

The font style is specified with <expC2>. If the STYLE clause is omitted, the standard font style is used.

Character	Font Style
B	Bold
I	Italic
N	Normal
O	Outline
S	Shadow
—	Strikeout
U	Underline

You can include more than one character to specify a combination of font styles. The following example open a Change window and uses a 15 point underlined System font:

```
CLOSE DATABASES
USE customer
IF _WINDOWS
    CHANGE FIELDS contact FONT 'System', 15 STYLE 'NU'
ENDIF
```

FREEZE <field name>

Include the FREEZE clause to allow changes to be made to a single field in the Change window. You specify this field with <field name>. The remaining fields are displayed and cannot be edited.

KEY <expr1> [,<expr2>]

Use the KEY clause to limit the scope of records that are displayed in the Change window. With KEY, you can specify an index key value (<expr1>) or a range of key values (<expr1>, <expr2>) for the records displayed in the Change window. The table/.DBF must be indexed, and the index key value or values included in the KEY clause must be the same data type as the index expression of the master index file or master tag.

For example, the CUSTOMER table/.DBF includes a character field containing zip codes. If the table/.DBF is indexed on the zip code field, you can specify a range of zip codes in the KEY clause. In this example, only records with zip codes falling within the range 43000 to 43999 are displayed in the Change window:

```
USE customer
SET ORDER TO zip
CHANGE FIELD company, zip KEY '43000','43999'
```


LAST

Any configuration changes made to the appearance of a Change window are saved in the FOXUSER resource file when the Change window is closed. The field list, the size of each field, and the location and size of the Change window are stored in the FOXUSER file. For additional information on this file, see SET RESOURCE in this chapter.

If you issue CHANGE with the LAST clause, the Change window opens in the same configuration that was last saved in the FOXUSER file. This restores the previous Change window configuration created with the last CHANGE. If the last CHANGE command issued in the Command window included a long list of clauses, issue CHANGE LAST to avoid having to retype the command.

Any Change window configuration changes you make in the current session aren't saved if you exit CHANGE by pressing Ctrl+Q.

**LEDIT
REDIT**

Information in a Change window can be viewed in two modes: Change or Browse. In the default Change mode, fields are displayed vertically. In Browse mode, fields are displayed horizontally.

You can split a Change window into left and right partitions by moving the window splitter with the mouse, by choosing **Resize Partitions** from the **Browse** menu or by including PARTITION in CHANGE. Each partition can be viewed in Change or Browse mode independently of the other partition.

If LEDIT or REDIT is included in CHANGE, the left or right partition is in Browse mode when the Change window is displayed. Include both key words to open both partitions in Browse mode. Each partition can be toggled between Change and Browse modes by choosing **Browse** and **Change** from the **Browse** menu popup.

```
USE customer
CHANGE PARTITION 30 LEDIT
```

LPARTITION

A Change window can be split into left and right partitions by including the PARTITION clause described later. By default, the cursor is placed in the first field in the right partition when the Change window is opened. If you include LPARTITION, the cursor is placed in the first field in the left Change window partition.

The cursor is placed in the right Change window partition if you include LPARTITION without the PARTITION clause.

NOAPPEND

If you include NOAPPEND in CHANGE, you can't add records to the table/.DBF by pressing Ctrl+N or choosing **Append Record** from the **Browse** menu.



Including NOAPPEND doesn't prevent you from appending a record from within a routine (created with VALID, WHEN or ON KEY LABEL) while in the Change window.

NOCLEAR

If NOCLEAR is included and you close the Change window, an image of the Change window remains displayed. This image isn't an active Change window. Use CLEAR to remove the Change window image.

NOCLEAR is ignored in FoxPro for Windows.

NODELETE

Including NODELETE prevents records from being marked for deletion from within a Change window. By default, a record can be marked for deletion by pressing Ctrl+T, choosing **Toggle Delete** from the **Browse** menu or clicking in the leftmost column of the record to be deleted.



Including NODELETE doesn't prevent you from marking a record for deletion from within a routine (created with VALID, WHEN or ON KEY LABEL) while in the Change window.

NOEDIT | NOMODIFY

Including NOEDIT or NOMODIFY prevents you from modifying the table/.DBF. NOEDIT and NOMODIFY are identical. If you include either clause, you can browse or search the table/.DBF, but you cannot be edit it. However, you can append and delete records. Include NOAPPEND and NODELETE to prohibit the addition and deletion of records.

NOLINK

By default, the left and right partitions of the Change window are linked; when you scroll through one partition, the other partition scrolls. You can unlink the partitions by including **NOLINK**. You can also unlink partitions by choosing **Partitions** from the **Browse** menu.

NOMENU

Including **NOMENU** in **CHANGE** prevents access to the **Browse** menu and its options by removing the **Browse** menu pad from the system menu bar.

NOOPTIMIZE

To disable Rushmore optimization of **CHANGE**, include **NOOPTIMIZE**. For further information on the **NOOPTIMIZE** clause, see **SET OPTIMIZE** in this chapter, or consult the *Optimizing Your Application* chapter in the *FoxPro Developer's Guide*.

NORMAL

When the current output window is a user-defined window, the Change window derives its colors, size, position, title and control options (**GROW**, **FLOAT**, **Zoom**, and so on) from this window. If **NORMAL** is specified, the Change window is opened normally with its default settings as if there were no active user-defined window.

NOWAIT

When **CHANGE** is issued from within a program, a Change window is opened and program execution pauses until the Change window is closed. Including **NOWAIT** continues program execution immediately after the Change window is opened. The program doesn't wait for the Change window to be closed but continues executing on the program line immediately following the program line containing **CHANGE NOWAIT**.

This option is available only from within a program. Including **NOWAIT** in **CHANGE** has no effect when the command is issued in the Command window.

PARTITION <expN2>

A Change window can be split into left and right partitions by including PARTITION in CHANGE. The numeric expression <expN2> specifies the column in which the window splitter is located. For example, if <expN2> is 20, the window splitter is placed in column 20 of the Change window.

A Change window can also be split into left and right partitions by moving the window splitter with the mouse or by choosing **Resize Partitions** from the **Browse** menu.

PREFERENCE <expC3>

Include PREFERENCE in CHANGE to save a Change window's attributes and options for later use. Unlike LAST, which restores the Change window as it appeared in the previous session, PREFERENCE saves a Change window's attributes indefinitely in the FOXUSER resource file. Preferences can be retrieved at any time. For additional information about the FOXUSER resource file, see SET RESOURCE in this chapter.

Issuing CHANGE with the specified preference name <expC3> for the first time creates an entry in the FOXUSER file that saves the Change window configuration. Issuing CHANGE later with the same preference name restores the Change window to that preference state. When the Change window is closed, the preference state is updated.

If you exit a Change window by pressing Ctrl+Q, any Change window changes aren't saved to the resource file.

REST

By default, CHANGE positions the record pointer at the top of the table/.DBF. Include REST to prevent the record pointer from being moved from its current position to the top of the table/.DBF.

SAVE

Include SAVE with CHANGE to keep the Change window and any of its memo field text editing windows active and visible (open). You can then return to the Change window after cycling through other open windows with the keyboard or the mouse.

This option is available only from within a program. SAVE has no effect when included with CHANGE in the Command window because CHANGE SAVE is always the default in the interactive mode.

TIMEOUT <expN3>

Include TIMEOUT to specify how long a Change window waits for input. The numeric expression <expN3> specifies how many seconds can elapse without any input before the Change window automatically closes.

TIMEOUT is available only from within a program; it has no effect when issued in the Command window. In the following example the Change window is closed if no input occurs in 10 seconds.

```
DEFINE WINDOW mybrowse FROM 1,1 TO 24,40 ;
    CLOSE ;
    GROW ;
    COLOR SCHEME 10

CLOSE DATABASES
USE customer
CHANGE WINDOW mybrowse ;
    FIELDS ytdpurch :H = 'Purchases:' , ;
    company :H = 'Company:' ;
    TIMEOUT 10
RELEASE WINDOW mybrowse
```

TITLE <expC4>

By default, the name of the table/.DBF being browsed appears in the top border of the Change window. If you issue CHANGE WINDOW <window name1> and the window has a title, the Change window assumes <window name1> as its title.

You can overwrite the table/.DBF name or window title with your own title by including a TITLE clause. The character expression <expC4> is the new title you want to display in the top border of your Change window.

VALID [:F] <expL2> [ERROR <expC5>]

Including VALID lets you perform record-level validation in a Change window. VALID is executed only if a change is made to the record and you move the cursor to another record. The VALID clause is *not* executed if the only change is to a memo field.

If VALID returns a true value (.T.), you can move the cursor to another record.

If VALID returns a false value (.F.), the cursor remains in the current field, and the system message “Invalid input” is displayed. You can display your own error message when VALID returns false by including the ERROR clause. The character expression <expC5> is displayed as the error message.

If VALID returns 0, the cursor remains in the current field, and an error message isn’t displayed.

You can force the VALID clause to execute before you move the cursor to the next record by including :F after the VALID key word. In this case, VALID is executed even if the record isn’t changed.

The VALID clause shouldn’t be confused with the verify option (:V), which enables *field-level* validation.

WHEN <expL4>

The WHEN clause is executed when you move the cursor to another record. If <expL4> evaluates to a true value (.T.), you can make changes to the record you move the cursor to. If <expL4> evaluates to false or 0, the record you select becomes read-only and cannot be modified. The cursor moves to the next field.

WIDTH <expN4>

The WIDTH clause limits the number of characters displayed for *all* fields in a partition of the Change window to <expN4>. Including the WIDTH clause doesn’t change the size of fields in the table/.DBF itself; it alters only the way the fields are displayed in the Change window.

WINDOW <window name1>

If you include the WINDOW clause in CHANGE, the Change window takes on the characteristics of the window specified with <window name1>. For example, if the window is defined with the FLOAT clause, the Change window can be moved. The specified window doesn’t have to be active or visible, but it must be defined.

IN [WINDOW] <window name2>

If the IN WINDOW clause is included, the Change window is opened within the parent window specified with <window name2>. The Change window doesn’t assume the characteristics of the parent window. A Change window activated inside a parent window cannot be moved outside the parent window. If the parent window is moved, the Change window moves with it.

To access the Change window, the parent window must first be defined with DEFINE WINDOW and must be active and visible.

IN SCREEN

By including IN SCREEN, you can explicitly place a Change window on the desktop in FoxPro for MS-DOS or in the main window in FoxPro for Windows when a user-defined window is active.

COLOR SCHEME <expN5>**COLOR <color pair list>**

The color of a Change window can be specified by including the number of an existing color scheme in the COLOR SCHEME clause or a set of color pairs in the COLOR clause.

A color scheme is a set of 10 predefined color pairs. The color pairs in a color scheme can be changed with SET COLOR OF SCHEME. In FoxPro for MS-DOS the color pairs in a color scheme can also be changed in the Color Picker.

A color pair is a set of two letters separated by a forward slash. The first color letter specifies the foreground color and the second letter specifies the background color.

For example, this color pair specifies a red foreground on a white background:

R/W

For a list of colors and their corresponding color letters, see the SET COLOR commands section in this chapter or the Color Table by Color Pair topic in the online help facility.

In FoxPro for Windows, a color pair can also be specified with a set of 6 RGB (Red Green Blue) color values separated by commas. The first 3 color values specify the foreground color and the second 3 color values specify the background color. The color values can range from 0 through 255.

In FoxPro for Windows, the R/W color pair in the example above can also be specified with this RGB color pair:

RGB(255,0,0,255,255,255)

In FoxPro for MS-DOS, the Change window assumes the colors specified with COLOR SCHEME 10 (Browse) by default.

In FoxPro for Windows, the Change window assumes the color scheme established using the Color option in the Microsoft Windows Control Panel.

The following table shows what element each color pair controls in FoxPro for MS-DOS.

Color Pair Number	Change Window Element
1	Other records (In FoxPro for Windows, color pair 1 controls the color of the current record and field)
2	Current field
3	Border
4	Active title
5	Idle title
6	Selected text
7	Current record
8	Shadow

In FoxPro for MS-DOS, the second color specified in color pair number 1 designates the color of the grid background. The second color specified in color pair number 3 is used for the grid foreground.

CHR()

Purpose	Returns the character associated with the specified numeric ASCII code
Syntax	CHR(<expN>)
Returns	Character
See Also	ASC(), INKEY()

Description CHR() returns a single character corresponding to a numeric expression. CHR() can be used to send printer control codes to a printer.

Parameter <expN>

Include in CHR() a numeric expression <expN> that evaluates to a number in the range of 0 to 255. The numeric expression is evaluated and the equivalent ASCII character is returned.

Use ASC() to return the ASCII code for a specified character.

Examples

? CHR(65)

A

? CHR(97)

a

? CHR(7) && Rings the bell

The following example puts an HP[®] LaserJet[®] Series II printer into landscape mode:

```
??? CHR(27) + '&l10'
```

CHRS AW()

Purpose	Determines whether or not a character is present in the keyboard buffer
Syntax	CHRS AW([<expN>])
Returns	Logical
See Also	INKEY(), READKEY()

Description CHRS AW() returns true (.T.) if a character is present in the keyboard buffer, and false (.F.) if not. CHRS AW() doesn't affect the keyboard buffer contents.

Parameter <expN>

If the <expN> is included, CHRS AW() waits <expN> seconds before checking the keyboard buffer. If <expN> is omitted, the keyboard buffer is checked immediately.

Including <expN> lets you use CHRS AW() for a variety of timed activities. For example, your program can close an application if a key hasn't been pressed for a specific number of seconds.

Example In this example, the system displays a window containing input fields created with @ ... GETs, and waits 5 seconds for keyboard input. If a key isn't pressed, CHRS AW() returns false (.F.) and the program terminates.

```
SET TALK OFF
DEFINE WINDOW menter FROM 7,10 to 17,70 PANEL
ACTIVATE WINDOW menter

@ 1,3 SAY 'Customer: ' GET mcustomer DEFAULT SPACE(40)
@ 3,3 SAY 'Address: ' GET maddress DEFAULT SPACE(40)
@ 5,3 SAY 'City: ' GET mcity DEFAULT SPACE(24)
@ 7,3 SAY 'State: ' GET mstate DEFAULT SPACE(2)
@ 7,18 SAY 'Zip: ' GET mzip DEFAULT SPACE(10)
@ 1,14 SAY ''
WAIT WINDOW 'Waiting for input' NOWAIT
```

```
IF NOT CHRS AW(5)
    DEACTIVATE WINDOW menter
    CLEAR GETS
ELSE
    READ
    DEACTIVATE WINDOW menter
ENDIF

RELEASE WINDOW menter
WAIT CLEAR
```

CHRTRAN()

Purpose	Replaces each character in a character expression that matches a character in a second character expression with the corresponding character in a third character expression
Syntax	CHRTRAN(<expC1>, <expC2>, <expC3>)
Parameters	<expC1> The expression to be translated <expC2> The first translation expression <expC3> The second translation expression
Returns	Character
See Also	SYS(15)

Description CHRTRAN() translates the character expression <expC1> using the translation expressions <expC2> and <expC3>.

Parameters <expC1>, <expC2>, <expC3>

If a character from <expC1> is found in <expC2>, the character in <expC1> is replaced by the character from <expC3> that's in the same position in <expC3> as is the character in <expC2>.

If <expC3> has fewer characters than <expC2>, the additional characters in <expC2> are translated to null characters. If <expC3> has more characters than <expC2>, the additional characters are ignored.

Example

```
? CHRTRAN('abcdef', 'ace', 'xyz')
xbydzf
```

```
? CHRTRAN('abcd', 'abc', 'yz')
yzd
```

```
? CHRTRAN('abcdef', 'ace', 'xyzqrst')
xbydzf
```

CLEAR

Purpose	Clears the specified item
Syntax	CLEAR CLEAR ALL CLEAR FIELDS CLEAR GETS CLEAR MACROS CLEAR MEMORY CLEAR MENUS CLEAR POPUPS CLEAR PROGRAM CLEAR PROMPT CLEAR READ CLEAR TYPEAHEAD CLEAR WINDOWS
See Also	Individual CLEAR commands below

Description **CLEAR**

CLEAR erases the desktop in FoxPro for MS-DOS, the main FoxPro window in FoxPro for Windows or the current user-defined window, and releases all pending @ ... GET statements. You can include CLEAR in format files.

See also: @ ... SAY, @ ... GET, @ ... CLEAR, CLEAR GETS

CLEAR ALL

CLEAR ALL releases from memory all memory variables and arrays and the definitions of all user-defined menu bars, popups and windows; it also closes any open tables/.DBFs, including all associated index, format and memo files, and selects work area 1. CLEAR ALL doesn't release system memory variables.

See also: CLEAR MEMORY, CLOSE, RELEASE ALL

CLEAR FIELDS

CLEAR FIELDS releases a list created with SET FIELDS and executes SET FIELDS OFF. CLEAR FIELDS differs from SET FIELDS TO in that it releases all field lists in all work areas, not just the fields list in the current work area. Also, SET FIELDS TO doesn't implicitly issue a SET FIELDS OFF.

See also: SET FIELDS

CLEAR GETS

CLEAR GETS releases all pending @ ... GET statements. Issuing CLEAR also releases all pending @ ... GET statements.

See also: @ ... SAY, @ ... GET

CLEAR MACROS

CLEAR MACROS releases from memory all keyboard macros, including any SET FUNCTION key assignments. Macros can be saved to a macro file or to a memo field with SAVE MACROS and restored later with RESTORE MACROS. You can also restore the default macros with RESTORE MACROS.

See also: PLAY MACRO, RESTORE MACROS, SAVE MACROS

CLEAR MEMORY

CLEAR MEMORY releases from memory all public and private memory variables and arrays. System memory variables aren't released.

See also: RELEASE ALL

CLEAR MENUS

CLEAR MENUS releases all menu bar definitions from memory.

See also: ACTIVATE MENU, CREATE MENU, DEFINE MENU,
HIDE MENU, SHOW MENU

CLEAR POPUPS

CLEAR POPUPS releases all popup definitions established with DEFINE POPUP from memory.

See also: ACTIVATE POPUP, CREATE MENU, DEFINE POPUP,
HIDE POPUP, SHOW POPUP

CLEAR PROGRAM

CLEAR PROGRAM clears the compiled program buffer. FoxPro keeps a buffer of the most recently executed programs. In rare cases, FoxPro may not recognize changes made to program files on disk. CLEAR PROGRAM forces FoxPro to read the programs from disk, rather than from the program buffer. The most common reason FoxPro may not recognize changes made to program files is your using an external or TSR (Terminate and Stay Resident) editor to modify a program file. With this exception, you should not have to use CLEAR PROGRAM.

CLEAR PROMPT

CLEAR PROMPT releases menu prompts created with @ ... PROMPT.

See also: @ ... PROMPT

CLEAR READ [ALL]

CLEAR READ exits the active READ, and returns control to the previous READ level if the active READ is nested. READs can be nested up to five levels. When CLEAR READ is issued, the READ isn't immediately terminated. Any commands that remain to be executed before returning to the READ are executed.

FoxPro supports nested READs. A nested READ is created by using a VALID or WHEN clause with @ ... GET to call a UDF (user-defined function) containing additional @ ... GETs and a READ. READs can be nested in this manner to five levels. CLEAR READ terminates the READ on the current level and returns control to the previous READ level.

Including ALL terminates all READs on all READ levels.

CLEAR READ will not terminate a READ issued with no previous @ ... GET statements. In this case, READ VALID is executed when CLEAR READ is issued. If the READ follows @ ... GET statements and has a VALID clause, the VALID clause is *not* executed when CLEAR READ is issued. For more information on commands issued with no previous @ ... GET statements, see the discussion of Foundation READs in the READ topic in this chapter.

Example

```
SET TALK OFF
DEFINE WINDOW entry FROM 6,14 TO 14,65 ;
    TITLE 'Input Window' FLOAT SHADOW SYSTEM COLOR SCHEME 8
ACTIVATE WINDOW entry
@ 2,2 SAY 'First Name:'
@ 4,2 SAY 'Last Name:'
@ 2,16 GET firstname SIZE 1,24 DEFAULT ''
@ 4,16 GET lastname SIZE 1,24 DEFAULT ''
@ 6,14 GET button FUNCTION '*HN \? \<Clear;!\<Exit' ;
    SIZE 1,9,2 DEFAULT '' VALID clearexit( )
READ CYCLE
RELEASE WINDOW entry
```

CLEAR

```
FUNCTION clearexit
DO CASE
    CASE button = 'Clear'
        firstname = SPACE(10)
        lastname = SPACE(10)
        _CUROBJ = 1
        SHOW GETS
        WAIT WINDOW 'Enter Name Again' NOWAIT
    CASE button = 'Exit'
        CLEAR READ
        WAIT WINDOW 'Entry Complete' NOWAIT
ENDCASE
```

See also: CLEAR GETS, RDLEVEL(), READ

CLEAR TYPEAHEAD

CLEAR TYPEAHEAD clears the keyboard typeahead buffer. CLEAR TYPEAHEAD is useful when you want to prevent input into a field or prevent a response to a prompt before the field or prompt is displayed.

See also: SET TYPEAHEAD

CLEAR WINDOWS

CLEAR WINDOWS releases all user-defined window definitions from memory and clears the windows from the desktop in FoxPro for MS-DOS, the main FoxPro window in FoxPro for Windows, or the active user-defined window. Windows definitions can be saved with SAVE WINDOW to a file or memo field for later use.

See also: ACTIVATE WINDOW, DEFINE WINDOW, HIDE WINDOW, RELEASE WINDOWS, SAVE WINDOW, SHOW WINDOW

CLOSE

Purpose	Closes various file types
Syntax	CLOSE ALL CLOSE ALTERNATE CLOSE DATABASES CLOSE FORMAT CLOSE INDEXES CLOSE PROCEDURE
See Also	Individual CLOSE commands below

Commands **CLOSE ALL**

CLOSE ALL closes all files in all work areas and selects work area 1. CLOSE ALL also closes any files opened with the FCREATE() and FOPEN() low-level file functions.

CLOSE ALL also closes the following windows:

Label
Project
Report
RQBE
Screen

CLOSE ALL doesn't close these windows:

Command
Debug
Desk Accessories (for example, Calculator, and Puzzle)
Help
Trace

CLOSE ALTERNATE

CLOSE ALTERNATE closes the alternate file you opened with SET ALTERNATE.

See also: SET ALTERNATE

CLOSE DATABASES

CLOSE DATABASES closes all open table/.DBF, index, memo and format files and selects work area 1.

See also: USE

CLOSE FORMAT

CLOSE FORMAT closes the format file open in the current work area.

See also: SET FORMAT

CLOSE INDEXES

CLOSE INDEXES closes all open index files (both .IDX and .CDX files) in the current work area. A structural index (a .CDX file with the same root name as the open table/.DBF) isn't closed.

See also: INDEX, SET INDEX

CLOSE PROCEDURE

CLOSE PROCEDURE closes the file you opened with SET PROCEDURE.

See also: SET PROCEDURE

CLOSE MEMO

Purpose	Closes one or more memo editing windows
Syntax	CLOSE MEMO <memo field1> [, <memo field2> ...] ALL
See Also	BROWSE, MODIFY MEMO

Description CLOSE MEMO closes memo editing windows opened with MODIFY MEMO, or during a BROWSE, CHANGE or EDIT. CLOSE MEMO saves changes you made to the memo fields.

In FoxPro for MS-DOS, you can also exit a memo editing window and save changes you made to the memo field by pressing Ctrl+W or clicking the close box. To exit the memo window without saving any changes, press Escape or Ctrl+Q.

In FoxPro for Windows, you can also exit a memo editing window and save any changes made to the memo field by pressing Ctrl+W or choosing **Close** from the windows's **Control** menu. To exit the memo window without saving any changes, press Escape or Ctrl+Q. If you close the table/.DBF containing the memo field or fields, your changes are saved and the memo windows are closed.

Clauses <memo field1> [, <memo field2> ...] | ALL

Include the name of the memo field whose memo editing window you want to close. To close a set of memo editing windows, include a list of memo field names with the names separated from each other with a comma. You can close the memo editing window of a memo field in a table/.DBF open in another work area by including the table/.DBF alias.

Use ALL instead of listing memo field names to close all memo editing windows for all memo fields in tables/.DBFs open in every work area.

CMONTH()

Purpose	Returns the name of the month from a given date expression
Syntax	CMONTH(<expD>)
Returns	Character
See Also	DMY(), MDY(), MONTH()

Description CMONTH(), the character month function, returns the name of the month as a string in proper noun format.

Parameter <expD>
The date expression <expD> can be a function that returns a date, a date type memory variable, array element or field.

Examples ? CMONTH({9/25/92})
September

STORE {6/25/92} TO due_date
? 'Your payment was due in', CMONTH(due_date)
Your payment was due in June

STORE due_date+60 TO finaldate
? 'You must pay by', CMONTH(finaldate)
You must pay by August

CNTBAR()

Purpose	Returns the number of bars on a popup defined with DEFINE POPUP or on a FoxPro system popup
Syntax	CNTBAR(<expC>)
Returns	Numeric
See Also	ACTIVATE POPUP, CNTPAD(), CREATE MENU, DEFINE BAR, DEFINE POPUP, GETBAR(), MRKBAR(), ON BAR, ON SELECTION BAR, PRMBAR()

Description If the popup bars are defined with DEFINE POPUP ... PROMPT, FoxPro evaluates the prompts when you issue ACTIVATE POPUP. For such a popup, CNTBAR() returns a meaningful value only after you activate the popup. If the popup bars are defined with DEFINE BAR, CNTBAR() can determine the number of bars before you issue ACTIVATE POPUP.

Parameter **<expC>**
Specify the name of the popup with <expC>. See MENU – System Menu Names in this chapter for a list of FoxPro system menus.

Example In the following example, CNTBAR.PRG adds a menu pad to the system menu. The popup ENVIRO is defined with four bars. Each bar sets the item indicated on or off. The program places a mark character by each bar when the associated item is turned on, and removes the mark when the associated item is turned off.

```
*** You must name this program CNTBAR.PRG ***
CLEAR
SET TALK OFF
DEFINE PAD enviro OF _MSYSMENU PROMPT 'E\<environment';
    KEY ALT+V, 'ALT+V'
ON PAD enviro OF _MSYSMENU ACTIVATE POPUP enviro

DEFINE POPUP enviro MARGIN RELATIVE SHADOW COLOR SCHEME 4
DEFINE BAR 1 OF enviro  PROMPT '\<Status Bar'
DEFINE BAR 2 OF enviro  PROMPT '\<Clock'
DEFINE BAR 3 OF enviro  PROMPT '\<Extended Video'
DEFINE BAR 4 OF enviro  PROMPT 'St\<icky'
```

```

ON SELECTION POPUP enviro DO enviropop IN cntbar.prg

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

PROCEDURE enviropop
DO CASE
CASE PROMPT( ) = 'Status'
    IF mrkbar('enviro', bar( ))
        DO CASE
            CASE _WINDOWS
                SET STATUS BAR OFF
            CASE _DOS
                SET STATUS OFF
            OTHERWISE
                ENDCASE
            SET MARK OF BAR bar( ) OF enviro TO .F.
        ELSE
            DO CASE
                CASE _WINDOWS
                    SET STATUS BAR ON
                CASE _DOS
                    SET STATUS ON
                OTHERWISE
                    ENDCASE
            SET MARK OF BAR bar( ) OF enviro TO .T.
        ENDIF
    
```

```

CASE PROMPT( ) = 'Clock'
  IF mrkbar('enviro', bar( ))
    SET CLOCK OFF
    SET MARK OF BAR bar( ) OF enviro TO .F.
  ELSE
    DO CASE
      CASE _WINDOWS
        SET STATUS BAR ON
        SET CLOCK STATUS
      CASE _DOS
        SET CLOCK ON
      OTHERWISE
    ENDCASE
    SET MARK OF BAR bar( ) OF enviro TO .T.
  ENDIF
CASE PROMPT( ) = 'Extended Video'
  IF MRKBAR('enviro', bar( ))
    SET DISPLAY TO VGA25
    SET MARK OF BAR bar( ) OF enviro TO .F.
  ELSE
    SET DISPLAY TO VGA50
    SET MARK OF BAR bar( ) OF enviro TO .T.
  ENDIF
CASE PROMPT( ) = 'Sticky'
  IF MRKBAR('enviro', bar( ))
    DO CASE
      CASE _WINDOWS
        WAIT WINDOW 'STICKY is always ON in FoxProw'
      CASE _DOS
        SET STICKY OFF
      OTHERWISE
    ENDCASE
    SET MARK OF BAR bar( ) OF enviro TO .F.
  ELSE
    DO CASE
      CASE _WINDOWS
        WAIT WINDOW 'STICKY is always ON in FoxProw'
      CASE _DOS
        SET STICKY ON
      OTHERWISE
    ENDCASE
    SET MARK OF BAR bar( ) OF enviro TO .T.
  ENDIF
ENDCASE

```

CNTPAD()

Purpose	Returns the number of pads on a user-defined menu bar or the FoxPro system menu bar
Syntax	CNTPAD(<expC>)
Returns	Numeric
See Also	ACTIVATE MENU, CNTBAR(), CREATE MENU, DEFINE PAD, DEFINE MENU, GETPAD(), MRKPAD(), ON PAD, ON SELECTION PAD, PRMPAD()

Description CNTPAD() returns the number of pads in a menu.

Parameter **<expC>**
Specify the name of the menu with <expC>.

Example The following line of code uses CNTPAD() to count the number of pads in the FoxPro system menu bar.

```
?CNTPAD( '_MSYSMENU' )
```


COL()

Purpose	Returns the current column position of the cursor
Syntax	COL()
Returns	Numeric
See Also	PCOL(), PROW(), ROW(), WCOLS(), WROWS()

Description COL() is especially useful for directing screen output to a column position relative to the current column position of the cursor.

The special operator \$ can be used in place of COL().

Example

```
@ 5,5 SAY ''  
@ ROW( ),COL( )+12 SAY 'Contact person'  
@ ROW( ),$+12 SAY 'Contact person'
```

COMPILE

Purpose	Compiles one or more files and creates an object file for each source file
Syntax	COMPILE <file> <skel> [ENCRYPT] [NODEBUG]
See Also	BUILD APP, BUILD PROJECT, MODIFY COMMAND, MODIFY PROJECT, SET LOGERRORS

Description COMPILE compiles a source file or files and creates corresponding object files. FoxPro executes only object files, so if the source file hasn't been compiled the file is automatically compiled when you run the program.

The source file remains unchanged. A separate compiled file is created with the same root name as the source file, but with a different extension. This table lists the source and compiled file extensions for each file type:

File Type	Source Extension	Compiled Extension
Program file	PRG	FXP
Screen code	SPR	SPX
Menu code	MPR	MPX
Query	QPR	QPX
Format	FMT	PRX

The compiler detects any syntax errors in the source file. Compilation error messages are saved to a text file if SET LOGERRORS is ON when a file is compiled. The error log file has the same root name as the compiled file and an .ERR extension. If SET LOGERRORS is OFF, the error log file isn't created.

Clauses **<file> | <skel>**

You can compile a single file <file>, or a set of files that match a file skeleton <skel> that contains wildcard characters. For example, to compile all program files with the extension .PRG in the current directory, issue `COMPILE *.PRG`.

You can also compile program files by choosing **Compile...** from the **Program** menu. For more information on compiling program files, refer to the Program Menu chapter in the *FoxPro User's Guide*.

ENCRYPT

Including `ENCRYPT` encrypts your compiled FoxPro programs. This prevents any access to your original source programs. For additional source code protection, users of the FoxPro Distribution Kit should always include this option when compiling programs intended for distribution.

NODEBUG

Include `NODEBUG` to reduce the size of your compiled file by two bytes per source file line. These two bytes in the compiled file are a reference to the corresponding line in the source file. Removing these two bytes doesn't affect the program's performance, but it does reduce your compiled file size, and conserves disk space.

When you include the `NODEBUG` clause, you can neither view a program's execution in the Trace window nor use `MESSAGE(1)` to return the source code for a line that causes an error.

CONTINUE

Purpose	Continues the previous LOCATE
Syntax	CONTINUE
See Also	EOF(), FOUND(), LOCATE, SEEK

Description CONTINUE is used after LOCATE succeeds in finding a record, to continue the LOCATE operation. CONTINUE moves the record pointer to the next record for which the logical expression specified in the previous LOCATE evaluates to true (.T.).

You can continue LOCATE interactively by choosing **Continue** from the **Record** menu or by pressing Ctrl+K.

CONTINUE can be repeated until the end of the file is encountered, or until the end of the scope you specified with LOCATE is reached.

If CONTINUE succeeds in finding a record, RECNO() returns the record number of the record, FOUND() returns a value of true (.T.) and EOF() returns a value of false (.F.).

If CONTINUE doesn't succeed in finding a record, RECNO() returns the number of records in the table/.DBF plus one, FOUND() returns the value false (.F.) and EOF() returns the value true (.T.).

Example In this example, all customers from New York are counted and the total is displayed. All records are found using one LOCATE command followed by CONTINUE commands.

```
CLOSE DATABASES
SET TALK OFF
USE customer
STORE 0 TO mcount
LOCATE FOR ALLTRIM(UPPER(state)) = 'NY'
DO WHILE FOUND( )
    mcount = mcount + 1
    CONTINUE
ENDDO
? 'Total customers from New York state: ' + LTRIM(STR(mcount))
```

COPY FILE

Purpose	Duplicates any type of file
Syntax	COPY FILE <file1> TO <file2>
See Also	DELETE FILE, FILER, RENAME

Description COPY FILE creates an exact duplicate of the file whose name is specified in <file1>. You can use COPY FILE to copy any type of file. The file to be copied cannot be open. You must include the extensions for both the source file name <file1> and destination file name <file2>.

If you use COPY FILE to create a backup of a table that has a memo field and/or structural index, be sure to copy the .FPT and .CDX files also.

COPY INDEXES

Purpose	Copies single-entry index files to a compound index file
Syntax	COPY INDEXES <index file list> ALL [TO <cdx file>]
See Also	CDX(), COPY TAG, DELETE TAG, INDEX, TAG(), USE

Description Use COPY INDEXES to copy single-entry index files to a compound index file. A compound index file is an index file containing any number of separate index entries called *tags*. Each tag is identified by its unique tag name. The default extension for a compound index file is .CDX, and the default extension for a single-entry index file is .IDX.

You must first open the table and single-entry index files before using COPY INDEXES. By default, the new tags are added to the structural compound index file, which is automatically opened when the table/.DBF is opened. If a structural compound index file doesn't exist for the table/.DBF, COPY INDEXES creates one.

Clauses **<index file list>**

The single-entry index file names in the <index file list> must be separated from each other by a comma. For each index file in the index file list, COPY INDEXES creates a tag with the same index expression as the associated index file. The name assigned to each tag is the root name of the corresponding single-entry index file name. If you try to copy an index file that has the same name as an existing tag, a dialog is displayed (if SAFETY is ON) asking if you would like to overwrite the tag.

You can create a single-entry index file from a tag with COPY TAG.

ALL

If you include ALL instead of a specific index file list, all open single-entry index files are added to the compound index file.

TO <cdx file>

Include TO <cdx file> to add tags to a compound index file other than the structural compound index file. Specify the name of the compound index file in <cdx file>. If a compound index file with the specified name doesn't exist, it is automatically created. The default file name extension for the compound file is .CDX.

COPY MEMO

Purpose	Copies the contents of a memo field to a text file
Syntax	COPY MEMO <memo field> TO <file> [ADDITIVE]
See Also	APPEND MEMO, COPY FILE, COPY TO, MODIFY MEMO

Description Use COPY MEMO to export the contents of the specified memo field in the current record to a text file.

Clauses **<memo field> TO <file>**
 <memo field> specifies the name of the memo field to be copied, and <file> specifies the name of a new or existing text file. If you do not provide an extension in <file>, a .TXT extension is assigned. You can also include a path with the file name.

ADDITIVE

If you include the ADDITIVE clause, the contents of the memo field is appended to the end of the text file. If you do not include ADDITIVE, the memo field contents replaces the contents of the text file.

Example In this example, the contents of the memo field named NOTES are copied to a file called TEST.TXT. The memo field is then copied again and appended to the end of the text file.

```
CLOSE DATABASES
USE salesman
COPY MEMO notes TO test.txt
WAIT WINDOW 'Memo contents now in test.txt' NOWAIT
MODIFY FILE test.txt
COPY MEMO notes TO test.txt ADDITIVE
WAIT WINDOW 'Memo contents added again to test.txt' NOWAIT
MODIFY FILE test.txt
DELETE FILE test.txt
```

COPY STRUCTURE

Purpose	Copies the structure of the current table/.DBF to a new table/.DBF
Syntax	COPY STRUCTURE TO <file> [FIELDS <field list>] [[WITH] CDX [WITH] PRODUCTION]]
See Also	COPY STRUCTURE EXTENDED, CREATE, DISPLAY STRUCTURE

Description COPY STRUCTURE creates an empty table/.DBF named <file> with the same structure as the current table/.DBF.

Clauses **FIELDS <field list>**
If the FIELDS <field list> clause is included, only the fields whose names are specified in the <field list> are copied to the new table/.DBF. If the FIELDS <field list> clause is omitted, all fields are copied to the new table/.DBF.

[WITH] CDX | [WITH] PRODUCTION
If the original table/.DBF has a structural index file, you can create a structural index file for the new table/.DBF. Include CDX or PRODUCTION to create an identical structural index file for the new table/.DBF. The tags and index expressions from the original structural index file are copied to the new structural index file. The CDX and PRODUCTION clauses have the same effect.

Example In this example, the CUSTOMER table/.DBF is opened, its structure is copied to a table/.DBF called BACKUP and BACKUP is opened. APPEND FROM then appends records to the BACKUP table/.DBF from the CUSTOMER table/.DBF and browses the BACKUP table/.DBF.

```
CLOSE DATABASES
USE customer
COPY STRUCTURE TO backup
USE backup
APPEND FROM customer FOR state = 'OH'
BROWSE FIELDS state, company
USE
DELETE FILE backup.dbf
```


COPY STRUCTURE EXTENDED

Purpose	Copies the structure of the current table/.DBF to records in a new table/.DBF
Syntax	COPY STRUCTURE EXTENDED TO <file> [FIELDS <field list>]
See Also	CREATE FROM

Description Information about each field in the current table/.DBF is copied to one record in the new table/.DBF with this command. The structure of the new table/.DBF is fixed in format and consists of four fields. The names of the four fields and their contents are:

Field	Contents	
FIELD_NAME	Field names from selected table	
FIELD_TYPE	C = Character N = Numeric F = Float D = Date	L = Logical M = Memo G = General P = Picture
FIELD_LEN	Width of Field	
FIELD_DEC	Number of decimal places in numeric fields	

You can modify the newly created table/.DBF and then use CREATE FROM to create a new table/.DBF that has the new structure.

Clause **FIELDS <field list>**

If FIELDS <field list> is included, only the fields specified in the list have a record in the new table/.DBF. If a field list is omitted, all fields have a record in the new table/.DBF.

Example The following example displays the structure of the INVOICES table/.DBF, copies the structure extended to a TEMP table/.DBF, browses TEMP, creates a BACKUP table/.DBF from TEMP, and displays the structure of BACKUP.

```
CLOSE DATABASES  
CLEAR
```

```
USE invoices  
DISPLAY STRUCTURE  
WAIT WINDOW 'Structure of the invoices table/.DBF' NOWAIT  
COPY STRUCTURE EXTENDED TO temp
```

```
USE temp  
WAIT WINDOW 'The temp table/.DBF - 1 row per field in invoices'  
NOWAIT  
BROWSE
```

```
CREATE backup FROM temp  
USE backup  
DISPLAY STRUCTURE  
WAIT WINDOW 'Backup.dbf has the same structure as invoices' NOWAIT  
USE  
DELETE FILE temp.dbf  
DELETE FILE backup.dbf
```

COPY TAG

Purpose	Creates a single-entry index file from a tag in a compound index file
Syntax	COPY TAG <tag name> [OF <cdx file>] TO <index file>
See Also	CDX(), COPY INDEX, DELETE TAG, INDEX, TAG(), USE

Description A compound index (.CDX) file is a single-entry index file that consists of any number of separate tags. Each tag represents one index, and is identified by a unique tag name. A compound index file lets you have a large number of “indexes” available with only one index file open.

With COPY TAGS you can copy a tag from a compound index file to create a new single-entry (.IDX) index file. The compound index file from which you copy a tag must be open. If you are copying a tag from a structural compound index file (a file with the same root name as the associated table/.DBF), the structural compound index file is opened automatically when you open its table/.DBF.

Additional compound index files can be opened with USE ... INDEX or SET INDEX. For further information on compound index files, see INDEX in this chapter.

Single-entry (.IDX) index files can be added as tags to compound index files with COPY INDEX.

Clauses <tag name>

The <tag name> is a unique name that identifies a tag (index entry) in a compound index file. Use TAG() to obtain a tag name.

OF <cdx file>

If the OF <cdx file> clause isn't included, FoxPro looks first for the tag in the structural index file. If it isn't found there, all other compound index files open for the current table/.DBF are searched. If duplicate tag names occur in open compound index files, include the OF <cdx file> clause to specify which compound index file contains the specified tag.

TO <index file>

The TO <index file> clause specifies the name of the single-entry file to create.

COPY TO

Purpose	Copies the contents of the current table/.DBF to a new file
Syntax	COPY TO <file> [FIELDS <field list>] [<scope>] [FOR <expL1>] [WHILE <expL2>] [[WITH] CDX] [[WITH] PRODUCTION] [NOOPTIMIZE] [TYPE] [FOXPLUS DIF MOD SDF SYLK WK1 WKS WR1 WRK XLS DELIMITED [WITH <delimiter> WITH BLANK WITH TAB]]
See Also	APPEND FROM, COPY FILE, IMPORT, EXPORT
Description	Use this command to copy records from the current table/.DBF to a new file. If an index order is set, records are copied in master index order.
Clauses	<file> The name of the new file to which COPY TO copies data is specified with <file>. If you do not include an extension with the file name, the default extension for the specified file type is assigned. If you do not specify a file type, COPY TO creates a new table/.DBF and assigns the table/.DBF file name the default extension .DBF. FIELDS <field list> If you include FIELDS and a field list, you can specify which fields are copied to the new file. If the FIELDS clause is omitted, all fields are copied to the file. Memo fields aren't copied to the new file even if memo field names are included in the field list. <scope> The scope clauses are: ALL, NEXT <expN>, RECORD <expN>, and REST. These are explained in the Overview of the FoxPro Language chapter in this manual. Commands which include <scope> operate only on the table/.DBF in the active work area. You can specify a scope of records copied to a file. Only the records that fall within the range of records specified by the scope are copied. The default scope for COPY TO is ALL records.

FOR <expL1>

If the FOR <expL1> clause is included, only the records for which the logical condition <expL1> evaluates to true (.T.) are copied to the file. Include FOR <expL1> to conditionally copy records, filtering out undesired records.

Rushmore optimizes COPY TO with a FOR <expL1> clause if <expL1> is an optimizable expression. For best performance, use an optimizable expression in the FOR <expL1> clause. A discussion of Rushmore optimizable expressions appears in the Optimizing Your Application chapter in the *FoxPro Developer's Guide*.

WHILE <expL2>

If WHILE <expL2> is included, records are copied as long as the logical expression <expL2> evaluates to true (.T.).

[WITH] CDX | [WITH] PRODUCTION

If the table/.DBF you copy from has a structural index file, you can create a structural index file for the new table/.DBF. Including CDX or PRODUCTION creates an identical structural index file for the new table/.DBF. The tags and index expressions from the original structural index file are copied to the new structural index file. The CDX and PRODUCTION clauses have the same effect.

Do not include CDX or PRODUCTION if you are copying to a file other than a new FoxPro table/.DBF.

NOOPTIMIZE

Include NOOPTIMIZE to cause Rushmore to not optimize COPY TO. For further information on NOOPTIMIZE, see SET OPTIMIZE, and the discussion of Rushmore in the Optimizing Your Application chapter in the *FoxPro Developer's Guide*.

TYPE

If the file you are creating isn't a FoxPro table/.DBF, you must specify its file type. Although you must specify a file type, you need not include the key word TYPE. You can create a wide variety of different file types including DELIMITED ASCII text files in which you can specify a field delimiter.

FOXPLUS

FoxPro memo files have a different structure than FoxBASE+ memo files. If your source FoxPro table/.DBF contains a memo field, include the FOXPLUS clause to create a table/.DBF that can be used in FoxBASE+. The FoxPro memo field cannot contain binary data because FoxBASE+ does not support binary data in memo fields.

DIF

Use DIF to create a .DIF (Data Interchange Format) file, used by VisiCalc. Fields from the FoxPro table/.DBF become vectors (columns) and records become tuples (rows). The new file name is assigned a .DIF extension if you do not include an extension in <file>.

MOD

A FoxPro table/.DBF can be copied to a new file in Microsoft Multiplan version 4.01 MOD format. The new Microsoft Multiplan file name is assigned a .MOD extension if an extension isn't included.

SDF

An SDF (System Data Format) file is an ASCII text file in which records have a fixed length and end with a carriage return and line feed. Fields aren't delimited. The SDF file name is assigned a .TXT file extension if you do not include an extension.

SYLK

A SYLK file is a Symbolic Link interchange format (used in Microsoft MultiPlan) in which fields from the FoxPro table/.DBF become columns in the spreadsheet and records become rows. SYLK file names have no extension.

WK1

Data from a FoxPro table/.DBF can be copied to a new Lotus 1-2-3 spreadsheet by including the WK1 clause. Each field from the selected table/.DBF becomes a column in the spreadsheet; each table/.DBF record becomes a row in the spreadsheet. A .WK1 file name extension is assigned to the new spreadsheet for use in Lotus 1-2-3 revision 2.x.

WKS

A FoxPro table/.DBF can be copied to a new Lotus 1-2-3 spreadsheet with the WKS clause. Each field from the current table/.DBF becomes a column in the spreadsheet and the each record becomes a row in the spreadsheet. A .WKS file name extension is assigned to the new spreadsheet for use in Lotus 1-2-3 revision 1-A.

WR1

A FoxPro table/.DBF can be copied to a new Lotus Symphony spreadsheet with this the WR1 clause. Each field from the current table/.DBF becomes a column in the spreadsheet and the each record becomes a row in the spreadsheet. A .WR1 file name extension is assigned to the new spreadsheet for use in Symphony version 1.1 or 1.2.

WRK

A table/.DBF can be copied to a new Lotus Symphony spreadsheet with the WRK clause. Each field from the current table/.DBF becomes a column in the spreadsheet, and each record becomes a row in the spreadsheet. A .WR1 file name extension is assigned to the new spreadsheet for use in Lotus Symphony version 1.0.

XLS

The data in a FoxPro table/.DBF can be copied to a Microsoft Excel version 2 spreadsheet by including the XLS clause. Each field from the current table/.DBF becomes a column in the spreadsheet and each record becomes a row in the spreadsheet. A .XLS extension is assigned to the newly created spreadsheet if you do not include a file extension.

**DELIMITED [WITH <delimiter> | WITH BLANK | WITH TAB]
DELIMITED [WITH] | WITH BLANK | WITH TAB**

A DELIMITED file is an ASCII text file in which each record ends with a carriage return and line feed. The default field separator is a comma. Since character data may include commas, character fields are additionally delimited with double quotation marks. In the following example

```
"Smith",9999999,"TELEPHONE"
```

there are 2 character fields ("Smith" and "TELEPHONE") delimited with double quotation marks. There is one numeric field which is not delimited since numeric data does not contain commas. The 3 fields are separated with commas.

The DELIMITED WITH <delimiter> option can be used to replace the double quotation marks with one of your choice.

The comma is not truly a delimiter. However, the DELIMITED WITH BLANK or DELIMITED WITH TAB clauses allow you to replace the comma field separator with either a space or a tab. The data should not contain embedded spaces or tabs.

You cannot combine the WITH <delimiter> clause with either the WITH BLANK or WITH TAB clauses.

Unless you specify otherwise, a .TXT extension is assigned to all newly created DELIMITED files.

Example

In this example, CUSTOMER.DBF is opened and the next three records are copied to a new DELIMITED data file called TEMP.TXT.

```
CLOSE DATABASES
USE customer
COPY NEXT 3 TO temp TYPE DELIMITED
WAIT WINDOW 'This is the delimited text file' NOWAIT
MODIFY FILE temp.txt
DELETE FILE temp.txt
```


COPY TO ARRAY

Purpose	Copies data from the current table/.DBF to an array
Syntax	COPY TO ARRAY <array> [FIELDS <field list>] [<scope>] [FOR <expL1>] [WHILE <expL2>] [NOOPTIMIZE]
See Also	APPEND FROM ARRAY, DECLARE, DIMENSION, GATHER, PUBLIC, SCATTER, STORE

Description COPY TO ARRAY and SCATTER are similar. COPY TO ARRAY copies multiple records to an array while SCATTER copies just one record into an array or a set of memory variables. Both COPY TO ARRAY and SCATTER create a new array if an array with the name you specify doesn't exist.

To copy a single record to an array you can use a one-dimensional array. The one-dimensional array should have the same number of elements as fields in the table/.DBF not counting memo fields. Memo fields are ignored in COPY TO ARRAY.

In a one-dimensional array, the first field of a record is stored to the first element of the array, the second field is stored to the second element of the array and so on. If the one-dimensional array has more elements than the table/.DBF has fields, any remaining elements remain unchanged. If the array has fewer elements than the table/.DBF has fields, any remaining fields are ignored.

To copy multiple records or an entire table/.DBF to an array, use a two-dimensional array. The first subscript in the array (array rows) is the number of records the array can hold, the second subscript (array columns) is the number of fields the array can hold.

Each record is stored in one row of the array, and each field of the record is stored in one column of the array. For each record, the first field is stored to the first column of the array, the second field is stored to the second column of the array and so on. If the array has more columns than the table/.DBF has fields, any remaining columns aren't changed. If the array has fewer columns than the table/.DBF has fields, any remaining fields aren't stored to the array.

Each successive row in the array is filled with the contents of the next record in the table/.DBF. If the array has more rows than the table/.DBF has records, any remaining rows aren't changed. If the array has fewer rows than the table/.DBF has records, any remaining records aren't stored to the array.

Data can be copied from arrays to table/.DBF records with APPEND FROM ARRAY. Data can also be copied from either an array or memory variables to a table/.DBF with GATHER.

Clauses

<array>

The array to which data is copied.

FIELDS <field list>

If FIELDS <field list> is included, only the fields specified in the list are stored to the array. If the FIELDS <field list> clause is omitted, all fields are stored to the array if the array has enough columns.

<scope>

The scope clauses are: ALL, NEXT <expN>, RECORD <expN>, and REST. These are explained in the Overview of the FoxPro Language chapter in this manual. Commands which include <scope> operate only on the table/.DBF in the active work area.

You can specify a scope of records copied to the array. Only the records that fall within the range of records specified by the scope are copied.

The default scope for COPY TO ARRAY is ALL records.

FOR <expL1>

If the FOR <expL1> clause is included, only the records that satisfy the logical condition <expL1> are copied to the array. Including FOR lets you conditionally copy records to the array, filtering out undesired records.

Rushmore optimizes a COPY TO ARRAY query that includes FOR <expL1> if <expL1> is an optimizable expression. For best performance, use an optimizable expression in the FOR clause. A discussion of Rushmore optimizable expressions appears in the Optimizing Your Application chapter in the *FoxPro Developer's Guide*.

WHILE <expL2>

If the WHILE <expL2> clause is included, records are copied to the array as long as the logical expression <expL2> evaluates to true (.T.).

NOOPTIMIZE

To cause Rushmore to not optimize COPY TO ARRAY, include NOOPTIMIZE. For further information on NOOPTIMIZE, see SET OPTIMIZE, and the Optimizing Your Application chapter in the *FoxPro Developer's Guide*.

Example

In this example, the CUSTOMER table/.DBF is opened. A two-dimensional array is then created and the first three records from CUSTOMER are copied to the array. DISPLAY MEMORY shows the data stored in the array.

```
CLOSE DATABASES
USE customer
DIMENSION temp(3,10)
COPY NEXT 3 TO ARRAY temp
DISPLAY MEMORY LIKE temp
```

COS()

Purpose	Returns the cosine of a numeric expression
Syntax	COS(<expN>)
Returns	Cosine
See Also	ACOS(), DTOR(), RTOD(), SET DECIMALS, SIN()

Description This trigonometric function returns the cosine of a numeric expression.

Parameter **<expN>**

COS() returns the cosine of <expN>. <expN> can be any value, and the value returned by COS() ranges between -1 and 1.

<expN> is given in radians. To convert an angle from degrees to radians, use DTOR(). The number of decimal places returned by COS() can be specified with SET DECIMALS.

Examples

```
? COS(0)
1.00

? COS(PI( ))
-1.00

? COS(DTOR(180))
-1.00

STORE PI( ) * 3 TO angle
? COS(angle)
-1.00
```

COUNT

Purpose	Counts table/.DBF records
Syntax	COUNT [<scope> [FOR <expL1> [WHILE <expL2> [TO <memvar> [NOOPTIMIZE]
See Also	CALCULATE, SET DELETED, SET TALK

Description This command counts the records within a scope of records for which the FOR or WHILE conditions are true. If SET TALK is ON, the record count is displayed in this form:

`nnnnnn records`

Records marked for deletion are included in the count if SET DELETE is OFF.

Clauses <scope>

The scope clauses are: ALL, NEXT <expN>, RECORD <expN>, and REST. These are explained in the Overview of the FoxPro Language chapter in this manual. Commands which include <scope> operate only on the table/.DBF in the active work area.

You can specify a scope of records that are counted. Only the records that fall within the range of records specified by the scope are counted.

The default scope for COUNT is ALL records.

FOR <expL1>

If you include the FOR clause, only the records that satisfy the logical condition <expL1> are counted. Including FOR lets you conditionally count records, filtering out undesired records.

Rushmore will optimize a COUNT FOR query if <expL1> is an optimizable expression. For best performance, use an optimizable expression in the FOR clause. A discussion of Rushmore optimizable expressions appears in the Optimizing Your Application chapter in the FoxPro *Developer's Guide*.

WHILE <expL2>

If you include the WHILE clause, records are counted for as long as the logical expression <expL2> evaluates to true (.T.).

TO <memvar>

The number of records counted is stored to the memory variable or array element specified in <memvar>. A memory variable is created, if it isn't defined.

NOOPTIMIZE

To disable Rushmore optimization of COUNT, include NOOPTIMIZE. For further information on NOOPTIMIZE, see SET OPTIMIZE in this manual and the Optimizing Your Application chapter in the *FoxPro Developer's Guide*.

Example

The following example counts and displays the number of customers in Seattle.

```
CLOSE DATABASES
USE customer
COUNT TO mcount FOR city = 'Seattle'
WAIT WINDOW 'Total customers from Seattle is '+ LTRIM(STR(mcount))
```

CREATE

Purpose	Creates a new table/.DBF
Syntax	CREATE [<file> ?]
See Also	MODIFY STRUCTURE

Description CREATE builds a new FoxPro table/.DBF.

Specify the name of the table/.DBF to create in <file>. If you don't specify a table/.DBF name or if you include the question mark (?), a dialog is displayed so you can enter the name of the table/.DBF to create. You cannot create a table/.DBF with the name of an MS-DOS device (CON, NUL, PRN, COM1 ...). You should avoid using hyphens in a table/.DBF name because hyphenated table/.DBF names don't appear in the View window, and can cause confusion with the alias pointer (->).

A table/.DBF is created by defining the name, type and size of each field. Once the structure of the table/.DBF is created, you can then add records to the table/.DBF.

For more information on creating and creating and modifying tables/.DBFs, see the Database Menu chapter in the *FoxPro User's Guide*.

CREATE COLOR SET

Purpose	Creates a color set from the current color settings
Syntax	CREATE COLOR SET <color set name>
See Also	SET COLOR OF SCHEME, SET COLOR SET, SET COLOR TO
Description	CREATE COLOR SET creates a new color set from the current color settings. Every color pair in every color scheme is saved in the color set you create. A color set name can be up to 10 characters long, and can contain numbers and underscores, but it cannot begin with a number. Once you have created a color set, you can load it with SET COLOR SET. In FoxPro for MS-DOS, you can also load the color set in the Color Picker. Color sets are saved in the FoxPro resource file. If a color set exists with the same name you specify, it is overwritten.

CREATE CURSOR

Purpose	Creates a temporary table/.DBF
Syntax	<pre>CREATE CURSOR <dbf_name> (<fname1> <type> [(<precision> [, <scale>]) [, <fname2> ...]]) FROM ARRAY <array></pre>
See Also	AFIELDS(), CREATE, CREATE QUERY, CREATE TABLE - SQL, INSERT - SQL, MODIFY QUERY, SELECT - SQL

Description CREATE CURSOR creates a temporary table/.DBF that exists only until it is closed. A temporary table/.DBF created with CREATE CURSOR can be manipulated like any other table/.DBF — it can be browsed and indexed, and you can append and delete records.

The temporary table/.DBF is opened in the lowest available work area, and can be accessed by its alias.

Each field in the temporary table/.DBF is defined with a name, type, precision and scale. These definitions can be obtained from the command itself or from an array. The temporary table/.DBF is opened exclusively regardless of the setting of SET EXCLUSIVE.

Clauses **<dbf_name>**

<dbf_name> specifies the name of the temporary table/.DBF to create. <dbf_name> can include a path and can be a name expression.

(<fname1> <type> [(<precision> [, <scale>] [, <fname2> ...])

In this clause, <fname1> and <fname2> are names of fields in the temporary table/.DBF. Each <fname> can be a name expression. <type> is a single letter indicating the data type for the field. Some data types require that you specify a <precision> (field width) and <scale> (number of decimal places).

<type>, <precision> and <scale> can be:

<type>	<precision>	<scale>	Description
C	n	–	Character string of width n
D	–	–	Date
F	n	d	Float of width n with d decimal places
L	–	–	Logical
M	–	–	Memo
N	n	d	Numeric of width n with d decimal places
G	–	–	General
P	–	–	Picture

<precision> and <scale> are ignored for D, L, M, G and P types. If a <scale> isn't included for N or F types, the <scale> defaults to zero (no decimal places).

FROM ARRAY <array>

In this clause, <array> is the name of an existing array whose contents are the name, type, precision and scale for each field in the temporary table/DBF. The contents of the array can be defined with the AFIELDS() function.

Examples The following example creates a cursor with the alias EMPLOYEE.

```
CLOSE DATABASES
CLEAR
CREATE CURSOR employee ;
    (name C(20), addr C(30), city C(30), zip C(5), salary N(8,2), ;
    comments M)
? ALIAS( )
DISPLAY STRUCTURE
```

CREATE FROM

Purpose	Creates a table/.DBF from a COPY STRUCTURE EXTENDED file
Syntax	CREATE [<file1>] FROM [<file2>]
See Also	COPY STRUCTURE EXTENDED

Description This variation of CREATE assumes that the table/.DBF specified in <file2> has been created either with COPY STRUCTURE EXTENDED or manually. A new table/.DBF <file1> is created with the structure described in <file2>. The newly created table/.DBF becomes the active table/.DBF. The structure of the new table/.DBF is fixed in format and consists of four fields. The names of the four fields and their contents are:

Field	Contents	
FIELD_NAME	Field names from selected table/.DBF	
FIELD_TYPE	C = Character N = Numeric F = Float D = Date	L = Logical M = Memo G = General P = Picture
FIELD_LEN	Width of field	
FIELD_DEC	Number of decimal places in numeric fields	

If you don't include either <file1> or <file2> or both, a dialog is displayed. In this dialog, you can specify the file to be created or the FROM file or both.

Example The following example displays the structure of the INVOICES table/.DBF, copies the structure extended to a TEMP table/.DBF, browses TEMP, creates a BACKUP table/.DBF from TEMP, and displays the structure of BACKUP.

```
CLOSE DATABASES
```

```
CLEAR
```

```
USE invoices
```

```
DISPLAY STRUCTURE
```

```
WAIT WINDOW 'Structure of the invoices table/.DBF' NOWAIT
```

```
COPY STRUCTURE EXTENDED TO temp
```

```
USE temp
```

```
WAIT WINDOW 'Temp table/.DBF has 1 row per field in INVOICES'
```

```
NOWAIT
```

```
BROWSE
```

```
CREATE backup FROM temp
```

```
USE backup
```

```
DISPLAY STRUCTURE
```

```
WAIT WINDOW 'Backup.dbf has the same structure as INVOICES' NOWAIT
```

```
USE
```

```
DELETE FILE temp.dbf
```

```
DELETE FILE backup.dbf
```

CREATE LABEL

Purpose	Opens a Label Design window in FoxPro for MS-DOS and the New Label dialog in FoxPro for Windows
Syntax	CREATE LABEL [<file> ?] [NOWAIT] [SAVE] [[WINDOW <window name1>] [IN [WINDOW] <window name2> IN SCREEN]]
See Also	LABEL, MODIFY LABEL

Description In FoxPro for MS-DOS, this command opens a Label Design window. In FoxPro for Windows, this command opens the New Label dialog, and after you choose a label format, the Report Layout window is opened. Now you can create and print labels in standard label formats or design your own custom labels.

You can see how your labels will look before they are printed using Page Preview.

Clauses <file> | ?

If a file name is included, the label file is created with that name. If an extension isn't included with the file name, an LBX extension is automatically assigned. The Open dialog appears when the question mark (?) is included. A list of existing label files is displayed. Choose one of the existing labels or enter the name of a new labels to create.

NOWAIT

When CREATE LABEL is issued in a program, a Label Design or Report Layout window is opened and program execution pauses until the Label Design or Report Layout window is closed.

The NOWAIT option continues program execution after the Label Design Window or Report Layout window has been opened. The program doesn't wait for the Label Design or Report Layout window to be closed, but continues execution on the program line immediately following that containing CREATE LABEL NOWAIT. Including the NOWAIT option has no effect on the CREATE LABEL command, when it is issued from the Command window.

SAVE

During program execution, SAVE leaves the Label Design or Report Layout window open after another window is brought forward. Including the SAVE option has no effect, when issued from the Command window.

In FoxPro for MS-DOS, you can close a Label Design window opened with the SAVE option by pressing Ctrl+W, Ctrl+End, Ctrl+Q or Escape, by clicking the close box or by choosing the **Close** option from the **Edit** menu.

In FoxPro for Windows, you can close a Report Layout window created with SAVE by pressing Ctrl+W, Ctrl+End or Escape, or by choosing **Close** from the Browse window **Control** menu or the **Edit** menu.

WINDOW <window name1>

If you include WINDOW <window name1>, the Label Design or Report Layout window takes on the characteristics of the specified window. For example, if the window is created with the FLOAT option of DEFINE WINDOW, the Label Design or Report Layout window can be moved. The window doesn't have to be active or visible, but it must be defined.

The Label Design or Report Layout window has a default size that can be larger than the window from which it takes its characteristics. In this case, the Label Design or Report Layout window still assumes the characteristics of the window in which it is placed. The upper left corner of the Label Design or Report Layout window is placed at the same coordinates as the upper left corner of the window, and can extend beyond the window's borders.

IN [WINDOW] <window name2>

If IN WINDOW <window name2> is included, the Label Design or Report Layout window is opened within a parent window. It doesn't assume the characteristics of the parent window in which it is placed. A Label Design or Report Layout window activated inside a parent window cannot be moved outside the parent window. If the parent window is moved, the Label Design or Report Layout window moves with it.

The parent window must first be defined with DEFINE WINDOW, and must be visible to access the Label Design or Report Layout window.

IN SCREEN

You can place a Label Design or Report Layout window on the desktop in FoxPro for MS-DOS or in the main window in FoxPro for Windows by including IN SCREEN. This clause lets you explicitly place a Label Design or Report Layout window on the desktop or main window in FoxPro for Windows when a user-defined window is active. Label Design or Report Layout windows are placed on the desktop or main window in FoxPro for Windows by default.

CREATE MENU

Purpose	Opens a Menu Design window
Syntax	CREATE MENU [<file> ?] [NOWAIT] [SAVE] [[WINDOW <window name1>] [IN [WINDOW] <window name2> IN SCREEN]]
See Also	DEFINE MENU, MODIFY MENU

Description CREATE MENU opens a Menu Design window where user-defined menus and popups are easily created with the Menu Builder. After you have created your menus and popups, the menu and popup definition information is stored in a FoxPro table with an MNX extension.

Issuing CREATE MENU without any additional arguments opens a new Menu Design window. The name UNTITLED is temporarily assigned to the menu definition. When you exit the Menu Design window, you can save the menu definition under a different name.

Clauses <file>
If a file name is included, the menu table is created with that name. If an extension isn't included with the table name, an .MNX extension is automatically assigned.

?
The Open dialog appears when the question mark (?) is included. A list of existing report files is displayed. Choose one of the existing reports or enter the name of a new report to create.

NOWAIT
When CREATE MENU is issued in a program, a Menu Design window is opened and program execution pauses until the Menu Design window is closed.

The NOWAIT option continues program execution after the Menu Design window has been opened. The program doesn't wait for the Menu Design window to be closed, but continues execution on the program line immediately following that containing CREATE MENU NOWAIT. Including the NOWAIT option has no effect on CREATE MENU, when it is issued in the Command window.

SAVE

During program execution, SAVE leaves the Menu Design window open after another window is brought forward. Including the SAVE option has no effect, when issued in the Command window.

In FoxPro for MS-DOS, you can close a Menu Design window opened with the SAVE option by pressing Ctrl+W, Ctrl+End, Ctrl+Q or Escape, by clicking the close box or by choosing the **Close** option from the **Edit** menu.

In FoxPro for Windows, you can close a Menu Design window created with SAVE by pressing Ctrl+W, Ctrl+End or Escape, or by choosing **Close** from the Browse window **Control** menu or the **Edit** menu.

WINDOW <window name1>

If you include WINDOW <window name1>, the Menu Design window takes on the characteristics of the specified window. For example, if the window is created with the FLOAT option of DEFINE WINDOW, the Menu Design window can be moved. The window need not be active or visible, but it must be defined.

The Menu Design window has a default size that can be larger than the window from which it takes its characteristics. In this case, the Menu Design window still assumes the characteristics of the window in which it is placed. The upper left corner of the Menu Design window is placed at the same coordinates as the upper left corner of the window, and can extend beyond the window's borders.

IN [WINDOW] <window name2>

If IN WINDOW <window name2> is included, the Menu Design window is opened within a parent window. It doesn't assume the characteristics of the parent window in which it is placed. A Menu Design window activated inside a parent window cannot be moved outside the parent window. If the parent window is moved, the Menu Design window moves with it.

The parent window must first be defined with DEFINE WINDOW, and must be visible, to access the Menu Design window.

IN SCREEN

You can place a Menu Design window on the desktop in FoxPro for MS-DOS or in the main window in FoxPro for Windows by including IN SCREEN. This clause lets you explicitly place a Menu Design window on the desktop or main window when a user-defined window is active.

CREATE PROJECT

Purpose	Creates a project
Syntax	CREATE PROJECT [<file> ?] [NOWAIT] [SAVE] [WINDOW <window name1>] [IN [WINDOW] <window name2> IN SCREEN]
See Also	BUILD APP, BUILD PROJECT, MODIFY PROJECT

Description CREATE PROJECT opens a Project window. A *project* is a table that keeps track of all files that are required to create an application, as well as all dependencies, references and connections among the files. A project table has a .PJX extension and an associated memo file with a .PJT extension. In a project, you specify all the pieces that are required for the final application, then FoxPro ensures that compiled files are based on the latest source files.

A project table can be opened with USE and manipulated like any other FoxPro table.

Issuing the command CREATE PROJECT without any additional arguments opens a new Project window. The name UNTITLED is assigned to the project. When you exit the Project window, you can save the project definition with a different name.

For more information about creating a project, refer to the Project Manager chapter in the FoxPro *User's Guide*.

Clauses <file>

The Open dialog appears when the question mark (?) is included. A list of existing report files is displayed. Choose one of the existing reports or enter the name of a new report to create.

?

The Open dialog appears when the question mark (?) is included. A list of existing report files is displayed. Choose one of the existing reports or enter the name of a new report to create.

NOWAIT

When CREATE PROJECT is issued in a program, a Project window is opened and program execution pauses until the Project window is closed.

The NOWAIT option continues program execution after the Project window has been opened. The program doesn't wait for the Project window to be closed, but continues execution on the program line immediately following the line containing CREATE PROJECT NOWAIT. Including the NOWAIT option has no effect on the CREATE PROJECT command, when it is issued in the Command window.

SAVE

During program execution, SAVE leaves the Project window open after another window is brought forward. Including the SAVE option has no effect, when issued in the Command window.

In FoxPro for MS-DOS, you can close a Project window opened with the SAVE option by pressing Ctrl+W, Ctrl+End, Ctrl+Q or Escape, by clicking the close box or by choosing the **Close** option from the **Edit** menu.

In FoxPro for Windows, you can close a Project window created with SAVE by pressing Ctrl+W, Ctrl+End or Escape, or by choosing **Close** from the Browse window **Control** menu or the **Edit** menu.

WINDOW <window name1>

If you include WINDOW <window name1>, the Project window takes on the characteristics of the specified window. For example, if the window is created with the FLOAT option of DEFINE WINDOW, the Project window can be moved. The window need not be active or visible, but it must be defined.

The Project window has a default size that can be larger than the window from which it takes its characteristics. In this case, the Project window still assumes the characteristics of the window in which it is placed. The upper left corner of the Project window is placed at the same coordinates as the upper left corner of the window, and can extend beyond the window's borders.

IN [WINDOW] <window name2>

If `IN WINDOW <window name2>` is included, the Project window is opened within a parent window. It doesn't assume the characteristics of the parent window in which it is placed. A Project window activated inside a parent window cannot be moved outside the parent window. If the parent window is moved, the Project window moves with it.

The parent window must first be defined with `DEFINE WINDOW`, and must be visible to access the Project window.

IN SCREEN

You can place a Project window on the desktop in FoxPro for MS-DOS or in the main window in FoxPro for Windows by including `IN SCREEN`. This clause lets you explicitly place a Project window on the desktop or main window in FoxPro for Windows when a user-defined window is active. Project windows are placed on the desktop or main FoxPro window, by default.

CREATE QUERY

Purpose	Opens an RQBE window
Syntax	CREATE QUERY [<file> ?]
See Also	MODIFY QUERY

Description CREATE QUERY opens an RQBE window so you can interactively create a query (SQL SELECT command). All you need to do is specify settings in the RQBE window and FoxPro takes care of the rest.

SELECT is used to retrieve data from tables/.DBFs. SELECT is very powerful and can replace a series of FoxPro commands. Because one SQL SELECT performs the function of a series of FoxPro commands, SELECT optimizes program performance.

Think of SELECT as a way to pose a query to FoxPro to obtain information from tables/.DBFs. SELECT allows you to specify the information you want without telling FoxPro how to retrieve the information. FoxPro determines the best way to retrieve the information.

After you create a query, the query is stored as a FoxPro program file with a .QPR extension. A query program can be executed with DO. You must include the query file extension when executing a query with DO. For example,

```
DO my_query.qpr
```

Issuing the command CREATE QUERY without any additional arguments opens a new RQBE query window. The name UNTITLED is assigned to the query. When you exit the Query window, you can save the query with a different name.

For more information about creating queries, refer to the File Menu chapter in the *User's Guide* or the lesson on creating queries in the *FoxPro Getting Started* manual.

Clauses **<file>**

If a <file> name is included, the query is created with that name. If an extension isn't included with the file name, a .QPR extension is automatically assigned.

?

The Open dialog appears when the question mark (?) is included. A list of existing report files is displayed. Choose one of the existing reports or enter the name of a new report to create.

CREATE REPORT

Purpose	Opens a report in a Report Design window
Syntax	CREATE REPORT [<file> ?] [NOWAIT] [SAVE] [[WINDOW <window name1>] [IN [WINDOW] <window name2> IN SCREEN]]
See Also	CREATE REPORT – Quick Report, MODIFY REPORT, REPORT
Description	This form of CREATE REPORT opens a Report Design window. For more information on creating reports, refer to the reports chapter in the <i>FoxPro User's Guide</i>. CREATE REPORT can also be used to generate a quick report without opening the Report Design window. Refer to the next topic for information on creating a quick report with CREATE REPORT.
Clauses	<file> Issuing CREATE REPORT without any additional arguments opens a new Report Design window. The name UNTITLED is assigned to the report definition. When you exit the Report Design window you are prompted to save the report definition with a different name. If a <file> name is included, the report is saved under that name. An .FRX extension is automatically assigned. If a report file with that name already exists, you are asked if you want to overwrite the existing file (if SET SAFETY is ON). ? The Open dialog appears when the question mark (?) is included. A list of existing report files is displayed. Choose one of the existing reports or enter the name of a new report to create.

NOWAIT

When CREATE REPORT is issued in a program, a Report Design window is opened and program execution pauses until the Report Design window is closed.

The NOWAIT option continues program execution after the Report Design window has been opened. The program doesn't wait for the Report Design window to be closed, but continues execution on the program line immediately following that containing CREATE REPORT NOWAIT. Including the NOWAIT option has no effect on the CREATE REPORT command, when it is issued from the Command window.

SAVE

During program execution, SAVE leaves the Report Design window open after another window is brought forward. Including the SAVE option has no effect, when issued in the Command window.

In FoxPro for MS-DOS, you can close a Report Design window opened with the SAVE option by pressing Ctrl+W, Ctrl+End, Ctrl+Q or Escape, by clicking the close box or by choosing the **Close** option from the **Edit** menu.

In FoxPro for Windows, you can close a Report Design window created with SAVE by pressing Ctrl+W, Ctrl+End or Escape, or by choosing **Close** from the Browse window **Control** menu or the **Edit** menu.

WINDOW <window name1>

If you include WINDOW <window name1>, the Report Design window takes on the characteristics of the specified window. For example, if the window is created with the FLOAT option of DEFINE WINDOW, the Report Design window can be moved. The window need not be active or visible, but it must be defined.

The Report Design window has a default size that can be larger than the window from which it takes its characteristics. In this case, the Report Design window still assumes the characteristics of the window in which it is placed. The upper left corner of the Report Design window is placed at the same coordinates as the upper left corner of the window, and can extend beyond the window's borders.

IN [WINDOW] <window name2>

If `IN WINDOW <window name2>` is included, the Report Design window is opened within a parent window. It doesn't assume the characteristics of the parent window in which it is placed. A Report Design window activated inside a parent window cannot be moved outside the parent window. If the parent window is moved, the Report Design window moves with it.

The parent window must first be defined with `DEFINE WINDOW`, and must be active and visible to access the Report Design window.

IN SCREEN

You can place a Report Design window on the desktop in FoxPro for MS-DOS or in the main window in FoxPro for Windows by including `IN SCREEN`. This clause lets you explicitly place a Report Design window on the desktop or main window when a user-defined window is active. Report Design windows are placed on the desktop or main window in FoxPro for Windows, by default.

CREATE REPORT – Quick Report

Purpose	Programmatically creates a report
Syntax	CREATE REPORT <file1> ? FROM <file2> [FORM COLUMN] [FIELDS <field list>] [ALIAS] [NOOVERWRITE] [WIDTH <expN>]
See Also	CREATE REPORT, MODIFY REPORT, REPORT

Description This form of CREATE REPORT creates a quick report without opening a Report Design window. The report is created as if you chose the **Quick Report...** option while the Report Design window is open.

Another form of CREATE REPORT, discussed in the previous topic, opens the Report Writer in a Report Design window to let you interactively create a report. The Report Writer has a Page Preview feature to show you how your report looks before it is printed.

For more information on interactively creating quick reports, refer to the reports chapter in the *FoxPro User's Guide*.

Clauses

<file1>

When a file name <file1> is included, the report file is saved under that name. An FRX extension is automatically assigned. If a report file with that name already exists and the NOOVERWRITE option is omitted, you are asked if you want to overwrite the existing file (if SAFETY is SET ON).

?

The Open dialog appears when the question mark (?) is included. A list of existing report files is displayed. Choose one of the existing reports or enter the name of a new report to create.

FROM <file2>

Specify the name of the table/.DBF from which the quick report is created with <file2>. The table/.DBF doesn't have to be open.

FORM | COLUMN

If you include FORM, the quick report is created with the fields and their names arranged from top to bottom in the Detail band.

If you include COLUMN, the quick report is created with the fields arranged from left to right across the page in the Detail band. The field names are placed in the Page Header band. If you do not include FORM or COLUMN, the quick report defaults to the COLUMN format.

FIELDS <field list>

Including FIELDS lets you specify the fields from the table/.DBF that appear in the quick report. Separate the fields in the <field list> with commas.

ALIAS

Include ALIAS to add the table/.DBF alias to the field names in the quick report.

NOOVERWRITE

NOOVERWRITE prevents an existing report from being overwritten. If a report already exists with the name <file1>, the report isn't created.

WIDTH <expN>

The width of the report page can be specified by including WIDTH <expN>. The number of columns in the report is specified by <expN>.

CREATE SCREEN

Purpose	Opens a Screen Design window
Syntax	CREATE SCREEN [<file> ?] [NOWAIT] [SAVE] [WINDOW <window name1>] [IN [WINDOW] <window name2> IN SCREEN]
See Also	CREATE SCREEN – Quick Screen, MODIFY SCREEN

Description CREATE SCREEN opens a screen layout window.

For more information on creating screens, refer to the Screen Builder chapter in the *FoxPro User's Guide*. CREATE SCREEN can also be used to generate a quick screen without opening the screen layout window. Refer to the next topic for information on creating a quick screen with CREATE SCREEN.

Clauses <file>

Issuing CREATE SCREEN without any additional arguments opens a new Screen Design window. The name UNTITLED is assigned to the screen definition. When you exit the Screen Design window you are prompted to save the screen definition with a different name.

If a <file> name is included, the screen is saved under that name. An .SCX extension is automatically assigned. If a screen file with that name already exists, you are asked if you want to overwrite the existing file (if SAFETY is SET ON).

?

The Open dialog appears when the question mark (?) is included. A list of existing report files is displayed. Choose one of the existing reports or enter the name of a new report to create.

NOWAIT

When CREATE SCREEN is issued in a program, a Screen Design window is opened and program execution pauses until the Screen Design window is closed.

The NOWAIT option continues program execution after the Screen Design window has been opened. The program doesn't wait for the Screen Design window to be closed, but continues execution on the program line immediately following that containing CREATE SCREEN NOWAIT. Including the NOWAIT option has no effect on the CREATE SCREEN command, when it is issued in the Command window.

SAVE

During program execution, SAVE leaves the Screen Design window open after another window is brought forward. Including the SAVE option has no effect when issued from the Command window.

In FoxPro for MS-DOS, you can close a Screen Design window opened with the SAVE option by pressing Ctrl+W, Ctrl+End, Ctrl+Q or Escape, by clicking the close box or by choosing the **Close** option from the **Edit** menu.

In FoxPro for Windows, you can close a Screen Design window opened with the SAVE option by pressing Ctrl+W, Ctrl+End or Escape, or by choosing **Close** from the Browse window **Control** menu or the **Edit** menu.

WINDOW <window name1>

If you include WINDOW <window name1>, the Screen Design window takes on the characteristics of the specified window. For example, if the window is created with the FLOAT option of DEFINE WINDOW, the Screen Design window can be moved. The window need not be active or visible, but it must be defined.

The Screen Design window has a default size that can be larger than the window from which it takes its characteristics. In this case, the Screen Design window still assumes the characteristics of the window in which it is placed. The upper left corner of the Screen Design window is placed at the same coordinates as the upper left corner of the window, and can extend beyond the window's borders.

IN [WINDOW] <window name2>

If **IN WINDOW <window name2>** is included, the Screen Design window is opened within a parent window. It doesn't assume the characteristics of the parent window in which it is placed. A Screen Design window activated inside a parent window cannot be moved outside the parent window. If the parent window is moved, the Screen Design window moves with it.

The parent window must first be defined with **DEFINE WINDOW**, and must be active and visible to access the Screen Design window.

IN SCREEN

You can place a Screen Design window on the desktop in FoxPro for MS-DOS or in the main window in FoxPro for Windows by including **IN SCREEN**. This clause lets you explicitly place a Screen Design window on the desktop or main window in FoxPro for Windows when a user-defined window is active. Screen Design windows are placed on the desktop or main window in FoxPro for Windows, by default.

CREATE SCREEN – Quick Screen

Purpose Programmatically creates a screen

Syntax **CREATE SCREEN** <file1> | ? **FROM** <file2>
[ROW | COLUMN] [FIELDS <field list>] [ALIAS]
[NOOVERWRITE] [SIZE <expN1>, <expN2>] [SCREEN]]

See Also CREATE SCREEN, MODIFY SCREEN

Description This form of CREATE SCREEN creates a quick screen without opening a screen layout window. The screen is created as if you chose the **Quick Screen...** option while the Screen Design window is open.

Another form of CREATE SCREEN, discussed in the previous topic, opens the Screen Builder to let you interactively create a screen in a layout window.

For more information on interactively creating quick screens, refer to the Designing Screens with the Screen Builder in the *FoxPro User's Guide*.

Clauses **<file1>**

When a file name <file1> is included, the screen file is saved under that name. An .SCX extension is automatically assigned. If a screen file with that name already exists and the NOOVERWRITE option isn't included, you are asked if you want to overwrite the existing file (if SAFETY is SET ON).

?

The Open dialog appears when the question mark (?) is included. A list of existing report files is displayed. Choose one of the existing reports or enter the name of a new report to create.

FROM <file2>

Specify the name of the table/.DBF from which quick screen is created with <file2>. The table/.DBF doesn't have to be open.

ROW | COLUMN

If you include ROW, the quick screen is created with the fields and their names arranged from top to bottom. If you do not include ROW or COLUMN, the quick screen defaults to this format.

If you include COLUMN, the quick screen is created with the fields and their names arranged from left to right across the screen. If the total width of the fields is greater than the width of the screen, the fields may be displayed in the ROW format.

FIELDS <field list>

Including FIELDS lets you specify the fields from the table/.DBF that appear in the quick screen. Separate the fields in the <field list> with commas.

ALIAS

Include ALIAS to add the table/.DBF alias to the field names in the quick screen.

NOOVERWRITE

NOOVERWRITE prevents an existing screen from being overwritten. If a screen already exists with the name <file1>, the screen isn't created.

SIZE <expN1>, <expN2>

You can designate the window height and width by including the SIZE clause. The window height in rows is specified by <expN1>, the window width in columns by <expN2>.

SCREEN

Including SCREEN creates a quick screen that is placed on the desktop in FoxPro for MS-DOS or in the main window in FoxPro for Windows. This clause lets you explicitly place quick screen on the desktop or main FoxPro window when a user-defined window is active. Quick screens are placed on the desktop or main FoxPro window, by default.

CREATE TABLE

Purpose	Creates a table/.DBF having the specified fields
Syntax	CREATE TABLE DBF <dbf_name> (<fname1> <type> [(<precision> [, <scale>]) [, <fname2> ...]]) FROM ARRAY <array>
See Also	AFIELDS(), CREATE, CREATE QUERY, INSERT – SQL, MODIFY QUERY, SELECT – SQL, SQL
Description	CREATE TABLE creates a table/.DBF. Each new field is defined with a name, type, precision and scale. These definitions can be obtained from the command itself or from an array. The new table/.DBF is opened in the lowest available work area, and can be accessed by its alias. The new table/.DBF is opened exclusively, regardless of the current setting of SET EXCLUSIVE.
Clauses	CREATE TABLE DBF <dbf_name> In this clause, <dbf_name> specifies the name of the table/.DBF to create. The <dbf_name> can include a path and can be a name expression. The table/.DBF is opened in a new work area (if a table/.DBF is open in the current work area) and the new work area is selected. (<fname1> <type> [(<precision> [, <scale>] [, <fname2> ...]) In this clause, <fname1> and <fname2> are names of fields in the new table/.DBF. Each <fname> can be a name expression. <type> is a single letter indicating the data type for the field. Some data types require that you specify a <precision> (field width) and <scale> (number of decimal places).

<type>, <precision> and <scale> may be:

<type>	<precision>	<scale>	Description
C	n	–	Character string of width n
N	n	d	Numeric of width n with d decimal places
F	n	d	Float of width n with d decimal places
D	–	–	Date
L	–	–	Logical
M	–	–	Memo
G	–	–	General
P	–	–	Picture

<precision> and <scale> are ignored for D, L, M, G and P types. If a <scale> isn't included for N or F types, the <scale> defaults to zero (no decimal places).

FROM ARRAY <array>

In this clause, <array> is the name of an existing array whose contents are the name, type, precision and scale for each field in the table/.DBF. The contents of the array can be defined with the AFIELDS() function.

Examples This example creates a table/.DBF and displays its structure.

```
CLOSE DATABASES
CREATE TABLE employee ;
    (name C(20), addr C(30), city C(30), zip C(5), salary N(8,2), ;
    comments M)
DISPLAY STRUCTURE
```

This example dimensions and fills an array then creates a table/.DBF from the array.

```
CLOSE DATABASES
CLEAR
USE invoices
DISPLAY STRUCTURE
COPY STRUCTURE EXTENDED TO temp
USE temp
DIMENSION cur_copy[1,4]
SCATTER TO cur_copy
CREATE DBF backup FROM ARRAY cur_copy
DISPLAY STRUCTURE
```

CREATE VIEW

Purpose	Creates a view file from the FoxPro environment
Syntax	CREATE VIEW <file>
See Also	SET VIEW

Description CREATE VIEW builds a new view <file> containing information about the FoxPro environment. SET VIEW restores the environment saved to a view file by CREATE VIEW. View files created with CREATE VIEW are assigned a .VUE extension.

The information saved in a view file includes:

- All tables/.DBFs, index, alternate and format files currently open in all work areas
- All fields contained in the SET FIELDS list
- All established relations between open tables/.DBFs
- All filters in effect for open tables/.DBFs
- The DEFAULT and PATH settings
- The procedure file setting
- The current help file
- The current resource file
- The SET SKIP status
- The status bar state (ON or OFF)

View files are useful both in programs and while debugging. Only one command, SET VIEW TO <view file>, needs to be executed to establish the entire environment needed to complete a task. This prevents you from having to retype the command. While debugging, the environment settings can be saved in a view file, testing can be performed, and the environment can be restored to continue program execution.

CTOD()

Purpose	Converts a character expression to a date expression
Syntax	CTOD(<expC>)
Returns	Date
See Also	DTOC(), SET CENTURY, SET DATE, SYS(1), SYS(10), SYS(11)

Description CTOD(), the character to date function, returns a date type value from a character expression.

Parameter <expC>

The character expression <expC> is converted to a date value by CTOD(). <expC> must contain a valid date from 1/1/100 to 12/31/9999.

The default format for <expC> is mm/dd/yy. You can use SET DATE and SET CENTURY to change the default format. If the century isn't specified when entering a date (as in the character expression 12/25/91), the twentieth century is assumed.

Examples

```
STORE '7/4/1776' TO the_4th
? CTOD(the_4th)
07/04/76

STORE {09/15/92} TO today
STORE CTOD('12/25/92') TO christmas
STORE STR(christmas-today,3) TO days_left
? 'There are', days_left, 'days until Christmas'
There are 101 days until Christmas
```

CURDIR()

Purpose	Returns the current MS-DOS directory
Syntax	CURDIR([<expC>])
Parameters	<expC> The drive
Returns	Character
See Also	FULLPATH()

Description CURDIR() returns, as a character string, the current MS-DOS directory on a specified drive.

Parameter **<expC>**

The current directory is returned for the drive specified by <expC>. If the drive is omitted, the current default drive is assumed (the FoxPro SET DEFAULT drive). The null string is returned if the specified drive doesn't exist.

CURDIR() ignores the FoxPro SET PATH values.

Examples The following example stores the current directory to a variable, sets the default to the directory FoxPro was started in, displays the new directory, returns the default to the original directory, and displays the original directory.

```
CLEAR
? 'Current directory: ', CURDIR( )
old_dir = SET('DEFAULT') + SYS(2003)
SET DEFAULT TO (SYS(2004))
? 'FoxPro directory: ', CURDIR( )
SET DEFA TO (old_dir)
? 'Current directory:', CURDIR( )
```

DATE()

Purpose	Returns the current system date
Syntax	DATE()
Returns	Date
See Also	SET CENTURY, SET DATE, SET MARK TO, SYS()

Description DATE() returns the current system date, which is controlled by the operating system. No FoxPro commands or functions can directly change the system date.

The format of the character string returned by DATE() can be changed with SET CENTURY, SET DATE and SET MARK TO.

Examples The following example displays the current system date and the date with CENTURY on.

```
? DATE( )
```

```
SET CENTURY ON
```

```
? MDY(DATE( ))
```

DAY()

Purpose	Returns the numeric day of the month for a given date expression
Syntax	DAY(<expD>)
Returns	Numeric
See Also	CDOW(), DOW(), SYS()

Description DAY() returns a number from 1 through 31.

Parameter <expD>

The date expression <expD> can be a date literal, a date type memory variable, an array element or a field.

Example

```
STORE {05/03/92} TO bdate
```

```
? CDOW(bdate)
```

```
Sunday
```

```
? DAY(bdate)
```

```
3
```

```
? 'That date is', CMONTH(bdate),STR(DAY(bdate),2)
```

```
That date is May 3
```

DBF()

Purpose	Returns a table/.DBF name from a work area or a table/.DBF alias
Syntax	DBF([<expC> <expN>])
Parameters	<expC> The work area number <expN> The table/.DBF alias
Returns	Character
See Also	CDX(), FIELD(), NDX(), SET FULLPATH

Description DBF() returns the name of a table/.DBF open in a specified work area or the table/.DBF name from a table/.DBF alias.

Parameters <expC> | <expN>

<expN> is the work area number, and <expC> is the table/.DBF alias.

If you omit <expC> and <expN>, the name of the table/.DBF open in the current work area is returned.

The null string is returned by DBF() if a table/.DBF isn't open in the work area you specify. If a table/.DBF doesn't have the alias you specify with <expC>, the message "Alias not found" is displayed.

For information on creating an alias for a table/.DBF, see USE in this chapter.

Example The following example returns the name of a table/.DBF from its work area and its alias, and returns the null string after all tables/.DBFs have been closed.

```

CLOSE DATABASES
USE salesman IN 2 ALIAS salesper
? DBF(2)
? DBF('salesper')
CLOSE DATABASES
? DBF( )    && the null string is returned

```


DDEAbortTrans()

Purpose	Aborts an asynchronous DDE transaction
Syntax	DDEAbortTrans(<expN>)
Returns	Logical
Remarks	FoxPro for Windows only
See Also	DDEExecute(), DDELastError(), DDEPoke(), DDERequest()

Description Use DDEAbortTrans() to abort an asynchronous transaction. An *asynchronous transaction* allows FoxPro program execution to continue without waiting for the server application.

DDEExecute(), DDEPoke() and DDERequest() wait for the period specified by DDESetOption() for a server application to respond unless you specify a user-defined function (UDF) to execute when the server application responds. Specifying a UDF to execute in these functions creates an asynchronous transaction.

If DDEAbortTrans() is called before the server has processed the request, the UDF won't be called for the transaction.

True (.T.) is returned if the asynchronous transaction is successfully aborted. False (.F.) is returned if the asynchronous transaction cannot be aborted. Use DDELastError() to determine why the transaction couldn't be aborted.

Clause **<expN>**

<expN> is the transaction number returned by DDEExecute(), DDEPoke() or DDERequest() when the transaction is sent to the server application.

DDEAdvise()

Purpose	Creates a notify link or an automatic link used in dynamic data exchange (DDE)
Syntax	DDEAdvise(<expN1>, <expC1>, <expC2>, <expN2>)
Parameters	<expN1> The channel number <expC1> The item name <expC2> The name of a user-defined function <expN2> A number indicating link type
Returns	Logical
Remarks	FoxPro for Windows only
See Also	DDEInitiate(), DDELastError(), DDESetOption(), DDETerminate()

Description DDEAdvise() is used to create a notify link or an automatic link to an item name in a server application. When a *notify link* is created with DDEAdvise(), the server application notifies FoxPro for Windows that the item name has been modified. If an *automatic link* is created, the server application notifies FoxPro for Windows that the item name has been modified and passes the new data to FoxPro.

Before you can create a link, you must establish a channel to the server application with DDEInitiate().

You can also use DDEAdvise() to turn off notification from the server.

DDEAdvise() returns a true value (.T.) if it executes successfully; otherwise, it returns false (.F.).

Parameters <expN1>

Specify the channel number returned by DDEInitiate() with <expN1>.

<expC1>

Use <expC1> to specify the item name. For example, Microsoft Excel uses row and column notation to refer to cells in a spreadsheet. The item name R1C1 designates the cell in the first row and first column of the spreadsheet.

<expC2>

Use <expC2> to specify the user-defined function (UDF) that is executed when a notify link or an automatic link is established and the item <expC1> is modified. When the UDF is executed, it is passed the following six parameters in the order given below:

Parameter	Contents
Channel Number	The channel number of the server application
Action	ADVISE or TERMINATE
Item	The item name; for example, R1C1 for a Microsoft Excel spreadsheet cell
Data	The new data (automatic link) or the null string (notify link)
Format	The data format; for example, CF_TEXT
Advise Status	The link type (0 = manual, 1 = notify, 2 = automatic)

The UDF should have six parameters in its PARAMETER statement to accept the values passed from the server application. If a notify link is established, the UDF is executed and the null string is passed in the Data parameter. You can issue DDERequest() later to retrieve the data. If an automatic link is established, the UDF is executed and the data is passed in the Data parameter.

The Action parameter contains ADVISE when the link is being updated by the server. The UDF is called, and the Action parameter contains TERMINATE when the link is closed by the client or server.

Any value returned by the UDF is ignored.

<expN2>

<expN2> is a numeric value that specifies the link type in the following way:

<expN2>	Link type
0	Manual
1	Notify
2	Automatic

You can turn off notification from the server application by specifying 0 for <expN2>. If the item changes, the UDF isn't executed.

Example The following short program demonstrates how you can establish a channel to a Microsoft Excel spreadsheet named Sheet1. DDEAdvise() is used to establish two links to data in two cells (R1C1 and R1C2) in the spreadsheet. The UDF NEWDATA is executed when data in either of the cells changes. The UDF tests the Item and Advise parameters to determine which item changed and what kind of link has been established.

```

PUBLIC mchannum
mchannum = DDEInitiate('Excel', 'Sheet1')

IF mchannum != -1
    = DDEAdvise(mchannum, 'R1C1', 'newdata', 1)  && Notify link
    = DDEAdvise(mchannum, 'R1C2', 'newdata', 2)  && Automatic link
    WAIT WINDOW 'Enter data in EXCEL's first two cells.'
ENDIF

PROCEDURE newdata
PARAMETERS channel, action, item, data, format, advise
IF action = 'ADVISE'
    DO CASE
        CASE item = 'R1C1' && Notify link
            newvalue = DDERequest(channel, item)
            ? 'R1C1 notify link: ' + newvalue
        CASE item = 'R1C2' && Automatic link
            newvalue = data
            ? 'R1C2 automatic link: ' + newvalue
    ENDCASE
ELSE
    = DDETerminate(mchannum)
ENDIF

```

DDEEnabled()

Purpose	Enables or disables dynamic data exchange (DDE) processing or returns the status of DDE processing
Syntax	DDEEnabled([<expL1> <expN1> [, <expL2>]])
Parameters	<expL1> Globally enable or disable DDE processing <expN1> The channel number <expL2> Enable or disable DDE processing for a single channel
Returns	Logical
Remarks	FoxPro for Windows only
See Also	DDEAbortTrans(), DDEAdvise(), DDEExecute(), DDEInitiate(), DDELastError(), DDEPoke(), DDERequest(), DDESetOption(), DDESetService(), DDESetTopic(), DDETerminate()

Description Using DDEEnabled(), you can globally enable or disable processing. You can also enable or disable DDE processing for specific channels.

DDEEnabled() can be used to protect critical code or disable links for short periods. When DDE processing is disabled, client requests are queued until DDE processing is enabled.

If DDEEnabled() is issued without any of its optional arguments, the global DDE processing status is returned. DDE processing has been globally enabled if true (.T.) is returned. DDE processing has been globally disabled if false (.F.) is returned.

Parameters <expL1>

DDE processing can be enabled or disabled globally by specifying true (.T.) or false (.F.), respectively, with <expL1>. If DDE processing is successfully enabled or disabled, true (.T.) is returned. False (.F.) is returned if DDE processing cannot be enabled or disabled.

<expN1>

To return the DDE processing status for a specific channel number, specify with the channel number <expN1>. DDE processing is enabled for the channel if true (.T.) is returned. False (.F.) is returned if processing is disabled.

<expL2>

To enable DDE processing for a specific channel, include the channel number (<expN1>) and specify true (.T.) for <expL2>. To disable DDE processing for a specific channel, include the channel number (<expN1>) and specify false (.F.) for <expL2>.

DDEExecute()

Purpose	Using dynamic data exchange (DDE), sends a command to another application
Syntax	DDEExecute(<expN>, <expC1> [, <expC2> [, <expC3>]])
Parameters	<expN> The channel number <expC1> A command <expC2> The format <expC3> The name of a user-defined function (UDF)
Returns	Logical
Remarks	FoxPro for Windows only
See Also	DDEAbortTrans(), DDEInitiate(), DDELastError(), DDESetOption(), DDETerminate()

Description The command sent with DDEExecute() must be understood by the application. Before you can execute the command, you must establish a channel to the server application with DDEInitiate().

For example, Microsoft Excel has an extensive set of macro commands, including DDE commands that let you request data from FoxPro for Windows from within Microsoft Excel. If you establish a channel to Microsoft Excel, you can then use DDEExecute() to send macro commands to Microsoft Excel from within FoxPro for Windows.

If the receiving application successfully executes the command, DDEExecute() returns true (.T.). If the receiving application cannot successfully execute the command or if the channel number you include is not valid, DDEExecute() returns false (.F.). If the optional asynchronous UDF <expC3> is included, a transaction number is returned; if an error occurs, -1 is returned.

Parameters <expN>

<expN> is the channel number.

<expC1>

<expC1> is the command you want to send to the other application. The format of the command is determined by the application you are sending it to. Consult the application's documentation for the correct syntax.

<expC2>

The character expression <expC2> specifies the format for sending the command. The default format is CF_TEXT.

<expC3>

Include the optional character expression <expC3> to permit asynchronous command execution requests. If <expC3> is omitted, a client waits for the period specified with DDESetOption(). If you specify the name of a UDF with <expC3>, client program execution continues immediately after the command execution request is made.

When the server application finishes executing the command, the UDF specified with <expC3> is executed. The UDF is passed five parameters in this order:

Parameter	Contents
Channel Number	The channel number of the server application
Action	XACTCOMPLETE (successful execution) XACTFAIL (failed command execution)
Item	The item name; for example, R1C1 for a Microsoft Excel spreadsheet cell
Data	The new data (REQUEST) or data passed (POKE or EXECUTED)
Transaction Number	The transaction number returned by DDEExecute()

Use DDEAbortTrans() to cancel an uncompleted transaction. If the transaction fails, you can use DDELastError() to determine why it failed.

When you include <expC3>, DDEExecute() doesn't return a logical value. Instead, a transaction number is returned; if an error occurs, -1 is returned.

Example The following example uses DDEInitiate() to establish a channel between FoxPro for Windows and the Microsoft Excel spreadsheet Sheet1. DDEExecute() is used to execute a Microsoft Excel command that maximizes the Microsoft Excel window.

```
mchannum = DDEInitiate('Excel', 'Sheet1')

IF mchannum != -1
    mexecute = DDEExecute(mchannum, '[App.Maximize]')
    IF mexecute != .F.
        WAIT WINDOW 'EXCEL window has been zoomed out.'
    ENDIF
    = DDETerminate(mchannum) && Close the channel
ENDIF
```

DDEInitiate()

Purpose	Establishes a dynamic data exchange (DDE) channel between FoxPro for Windows and another Microsoft Windows-based application
Syntax	DDEInitiate(<expC1>, <expC2>)
Parameters	<expC1> The service name of the server application <expC2> The topic name
Returns	Numeric
Remarks	FoxPro for Windows only
See Also	DDEAdvise(), DDEEnabled(), DDEExecute(), DDELastError(), DDEPoke(), DDERequest(), DDESetOption(), DDETerminate(), RUN

Description DDEInitiate() establishes a DDE channel between FoxPro for Windows and a DDE server application. Once a channel is established, FoxPro can request data from the server by referencing the channel in subsequent DDE functions. FoxPro for Windows acts as the client, requesting data from the server application through the channel.

If the channel is successfully established, DDEInitiate() returns the channel number. Channel numbers are non-negative, and the number of channels you can establish is limited only by your system resources.

DDEInitiate() returns -1 if the channel cannot be established. If the server application isn't open, FoxPro asks if you would like to open it. If you choose **Yes**, FoxPro attempts to open the application. (You can use DDELastError() to determine why a channel cannot be established.)

To avoid being asked whether you want to open the application, set the DDESetOption() SAFETY option. You can also use RUN with the /N option to start the application.

A channel can be closed with DDETerminate().

Parameters <expC1>

<expC1> is the service name of the server application which, in most cases, is the name of the executable file without its extension. The default service name for FoxPro for Windows is FoxPro. If you are establishing a channel to Microsoft Excel, <expC1> is Excel.

<expC2>

Use <expC2> to specify the topic name. The topic is application specific and must be understood by the application. For example, one topic supplied by most DDE servers is the System topic.

Example

The following example uses DDEInitiate() to establish a channel between FoxPro for Windows and a Microsoft Excel spreadsheet named Sheet1. Excel is the service name, and Sheet1 is the topic name. The channel number is stored to the memory variable MCHANNEL for use in subsequent DDE functions.

```
mchannel = DDEInitiate('Excel', 'Sheet1')

IF mchannel != -1
    * Process client actions
    = DDETerminate(mchannel)  && Close the channel
ENDIF
```

DDELastError()

Purpose	Returns an error number for the last dynamic data exchange (DDE) function
Syntax	DDELastError()
Returns	Numeric
Remarks	FoxPro for Windows only
See Also	DDEAbortTrans(), DDEAdvise(), DDEEnabled(), DDEExecute(), DDEInitiate(), DDEPoke(), DDERequest(), DDESetOption(), DDESetService(), DDESetTopic(), DDETerminate()

Description You can use DDELastError() to help determine the cause of an error when a DDE function doesn't execute successfully.

DDELastError() returns 0 if the last DDE function executed successfully. It returns a non-zero value if the last DDE function was unsuccessful. The following table lists the error numbers and their descriptions.

Error Number	Description	Error Number	Description
1	Service busy	14	Low memory
2	Topic busy	15	Memory error
3	Channel busy	16	Connect failure
4	No such service	17	Request failure
5	No such topic	18	Poke timeout
6	Bad channel	19	Could not display message
7	Insufficient memory	20	Multiple synchronous transactions
8	Acknowledge timeout	21	Server died
9	Request timeout	22	Internal DDE error
10	No DDEInitiate()	23	Advise timeout
11	Client attempted server transaction	24	Invalid transaction identifier
12	Execute timeout	25	Unknown
13	Bad parameter		

DDEPoke()

Purpose	Sends data between client and server applications in a dynamic data exchange (DDE) conversation
Syntax	DDEPoke(<expN>, <expC1>, <expC2> [, <expC3> [, <expC4>]])
Parameters	<expN> The channel number <expC1> The item name <expC2> Data <expC3> Data format <expC4> The name of a user-defined function (UDF)
Returns	Logical
Remarks	FoxPro for Windows only
See Also	DDEAbortTrans(), DDEInitiate(), DDELastError(), DDESetOption(), DDETerminate()

Description DDEPoke() sends data as a character string to the item name in the application specified by the channel number.

If the data is successfully sent, DDEPoke() returns true (.T.). If the data cannot be sent, false (.F.) is returned. If the asynchronous UDF <expC3> is included, a transaction number is returned; if an error occurs, -1 is returned.

Parameters **<expN>**

Use the application's channel number (<expN>) to specify the application to send the data to. If the channel number is a server channel, DDEPoke() sends the data in response to a request or a previously established notify or automatic link.

<expC1>

The data is sent to the item name specified with <expC1>. The item name is application-specific and must be understood by the application. For example, Microsoft Excel supports R1C1 as a valid item name that refers to the first cell in a spreadsheet.

<expC2>

The data sent to the item name is specified with <expC2>.

<expC3>

The character expression <expC3> specifies the format used to send the data. The default format is CF_TEXT. In this format, fields are delimited with tabs and records are delimited with a carriage return and a line feed.

<expC4>

Include the character expression <expC4> to permit asynchronous data transfer. If <expC4> is omitted, a client waits for the period specified with DDESetOption(). If you specify the name of a UDF with <expC4>, client program execution continues immediately after the request is made.

When the data is available from the server application, the UDF specified with <expC4> is executed. The UDF is passed six parameters in this order:

Parameter	Contents
Channel Number	The channel number of the server application
Action	XACTCOMPLETE (successful transaction) XACTFAIL (failed transaction)
Item	The item name; for example, R1C1 for a Microsoft Excel spreadsheet cell
Data	The new data (REQUEST) or data passed (POKE or EXECUTED)
Format	The data format; for example, CF_TEXT
Transaction Number	The transaction number returned by DDEPoke()

Use DDEAbortTrans() to cancel an uncompleted transaction. If the transaction fails, you can use DDELastError() to determine why it failed.

When you include <expC3>, DDEPoke() returns a transaction number if successful or -1 if an error occurs.

DDERequest()

Purpose	Requests data from a server application in a dynamic data exchange (DDE) conversation
Syntax	DDERequest(<expN>, <expC1> [, <expC2> [, <expC3>]])
Parameters	<expN> The channel number <expC1> The item name <expC2> The data format <expC3> The name of a user-defined function (UDF)
Returns	Character
Remarks	FoxPro for Windows only
See Also	DDEAbortTrans(), DDEInitiate(), DDELastError(), DDESetOption(), DDETerminate()

Description Before you can request data using DDERequest(), you must establish a channel to the server application with DDEInitiate().

If the request for data is successful, DDERequest() returns the data as a character string. If the request fails, DDERequest() returns the null string and DDELastError() returns a non-zero value. If you include the asynchronous UDF <expC3>, DDERequest() returns a transaction number if successful or -1 if an error occurs.

Parameters <expN>

<expN> is the channel number.

<expC1>

Use <expC1> to specify the item name. The item name is application-specific and must be understood by the application. For example, Microsoft Excel uses row and column notation to refer to cells in a spreadsheet. The item name R1C1 designates the cell in the first row and first column of the spreadsheet.

<expC2>

The character expression <expC2> specifies a format for the data requested. The default format is CF_TEXT. In this format, fields are delimited with tabs and records are delimited with a carriage return and a line feed.

<expC3>

Include the character expression <expC3> to permit asynchronous data transfer. If <expC3> is omitted, a client waits for the period specified with DDESetOption(). If you specify the name of a UDF with <expC3>, client program execution continues immediately after the request is made.

When the data is available from the server application, the UDF specified with <expC3> is executed. The UDF is passed six parameters in this order:

Parameter	Contents
Channel Number	The channel number of the server application
Action	XACTCOMPLETE (successful transaction) XACTFAIL (failed transaction)
Item	The item name; for example, R1C1 for a Microsoft Excel spreadsheet cell
Data	The new data (REQUEST) or data passed (POKE or EXECUTED)
Format	The data format; for example, CF_TEXT
Transaction Number	The transaction number returned by DDERequest()

Use DDEAbortTrans() to cancel an uncompleted transaction. If the transaction fails, you can use DDELastError() to determine why it failed.

When you include <expC3>, DDERequest() returns a transaction number equal to or greater than 0 if successful, or -1 if an error occurs.

Example

The following example uses DDEInitiate() to establish a channel between FoxPro for Windows and a Microsoft Excel spreadsheet named Sheet1. Excel is the service name and Sheet1 is the topic name. The channel number is stored to a memory variable named MCHANNUM for use in subsequent DDE functions.

DDERequest() requests the item name R1C1, the data in the first row and column of the Sheet1 spreadsheet.

```
mchannum = DDEInitiate('Excel', 'Sheet1')

IF mchannum != -1
    mrequest = DDERequest(mchannum, 'R1C1')
    IF !EMPTY(mrequest) AND DDELastError( ) = 0  && Successful
        WAIT WINDOW 'R1C1 contents: ' + mrequest
    ENDIF
    = DDETerminate(mchannum)  && Close the channel
ENDIF
```

DDESetOption()

Purpose	Changes or returns dynamic data exchange (DDE) settings
Syntax	DDESetOption(<expC> [, <expN> <expL>])
Parameters	<expC> The setting option <expN> The timeout value <expL> Enable or disable dialog display
Returns	Logical or numeric
Remarks	FoxPro for Windows only
See Also	DDEAdvise(), DDEEnabled(), DDEExecute(), DDEInitiate(), DDEPoke(), DDERequest(), DDESetService(), DDETerminate()

Description Use DDESetOption() to change or return DDE settings. Two options are available.

Option <expC>	Setting	Default	Description
TIMEOUT	<expN> (number of milliseconds)	2000	The number of milliseconds DDE functions wait for the server application to respond. The current TIMEOUT value is returned if <expN> is omitted.
SAFETY	<expL> (true (.T.) or false (.F.))	.T.	Specifies if a dialog is displayed when you use DDEInitiate() to establish a channel to a server application and the application doesn't respond. The current SAFETY setting is returned if <expL> is omitted.

DDESetService()

Purpose	Creates, releases or modifies service names and settings in FoxPro for Windows
Syntax	DDESetService(<expC1>, <expC2> [, <expC3> <expL>])
Parameters	<expC1> The service name <expC2> The service option <expC3> The data format <expL> Enable or disable options
Returns	Logical
Remarks	FoxPro for Windows only
See Also	DDEEnabled(), DDELastError(), DDEPoke(), DDESetTopic()

Description FoxPro for Windows can act as a dynamic data exchange (DDE) server to send data to client Microsoft Windows-based applications. DDESetService() is used to create, release or modify service names and settings in FoxPro for Windows. Each service name can have a set of topic names created with DDESetTopic(). Client applications request data from DDE topic names.

DDESetService() returns true (.T.) if the service name is successfully created, released or modified. If the service name cannot be created, released or modified, false (.F.) is returned.

DDESetService() can be used to return information about a service name.

FoxPro for Windows has the default service name FoxPro. The FoxPro service name has one topic name called System. The following table lists all the item names supported by the System topic.

Item Name	Item Description
Topics	A list of available topic names
Formats	A list of supported formats
Status	Busy or Ready
SysItems	A list of item names

You can use DDESetTopic() to modify the FoxPro service name or to release it. See the DDESetTopic() options below for information about manipulating the FoxPro service name.

Parameters <expC1>

<expC1> is the service name to create, release, modify or return information about.

<expC2>

Use <expC2> to create, release or modify a service name or to return information about a service name. The following table lists the options you can specify with <expC2>, the default values for the options and a description of each option.

Option	Default Value	Description
DEFINE	—	Creates a new service name
RELEASE	—	Releases an existing service name
ADVISE	.F.	Enables or disables client notification of changes to item names
EXECUTE	.F.	Enables or disables command execution
POKE	.F.	Enables or disables client pokes to the service
REQUEST	.T.	Enables or disables requests to the service name
FORMATS	CF_TEXT	Specifies supported data formats

DEFINE

DEFINE creates a new service name. For example, the following command creates the service name MYSERVICE:

```
mnewservice = DDESetService('myservice', 'DEFINE')
```

RELEASE

RELEASE releases an existing service name to free up system resources. When a service name is released, all of the service's topic names are also released.

The following command releases the service name created in the previous example:

```
mrelease = DDESetService('myservice', 'RELEASE')
```

To release the default FoxPro service, issue this command:

```
mrelfox = DDESetService('FoxPro', 'RELEASE')
```

ADVISE

Use the ADVISE option to specify whether a client is notified when data changes in an item name or to return the current advise status for a service name. Refer to DDEAdvise() for additional information about advising clients.

To enable client notification, specify true (.T.) for <expL>. Specifying false (.F.) for <expL> disables client notification.

To return the current client notification status for the service name, omit <expL>. DDESetService() returns true if client notification is enabled for the service name; it returns false if client notification is disabled.

EXECUTE

With the EXECUTE option, you can enable or disable command execution requests to a service name or determine the current execute status for a service name.

To enable client requests to execute a command, specify true (.T.) for <expL>. Specifying false (.F.) for <expL> disables client requests to execute a command. .F. is the default value.

To return the current command execution status for the service name, omit <expL>. DDESetService() returns true if client command execution requests are enabled for the service name; otherwise, it returns false.

The following commands enable command execution and disable data requests from client applications for the service name MYSERVICE. The current command execution status for MYSERVICE is then displayed:

```
mexecute = DDESetService('myservice', 'EXECUTE', .T.)
mrequest = DDESetService('myservice', 'REQUEST', .F.)
? DDESetService('myservice', 'EXECUTE')
```

POKE

With the POKE option, you can enable or disable poke requests to the service name. You can also determine the current poke status for a service name. Refer to DDEPoke() for additional information about poking data to a server or a client.

To enable client poke requests, specify true (.T.) for <expL>. Specifying false (.F.) for <expL> disables client poke requests. .F. is the default value.

To return the current poke status for the service name, omit <expL>. DDESetService() returns true if poke requests are enabled for a service name; it returns false if poke requests are disabled.

REQUEST

Use REQUEST to enable or disable client requests to a service name or to return the current request status for the service name.

To enable client requests to the service name, specify true (.T.) for <expL>. Specifying false (.F.) disables client requests to the service name. .T. is the default value.

To return the current request status for a service name, omit <expL>. DDESetService() returns true if client requests are enabled for the service name; it returns false if client requests are disabled.

The following commands disable requests from client applications to the service name MYSERVICE and displays the current request status for MYSERVICE:

```
mrequest = DDESetService('myservice', 'REQUEST', .F.)
? DDESetService('myservice', 'REQUEST')
```

FORMATS

Include FORMATS and <expC3> to specify the data formats supported by the service name. Server requests for formats not specified with <expC3> are rejected.

<expC3>

You can specify data formats supported by the service name by specifying the formats with <expC3>. Include a list of the supported formats separated by commas. For example:

```
=DDESetService('myservice', 'FORMATS', 'CF_TEXT, CF_SYLK')
```

If <expC3> is omitted, only the CF_TEXT format is supported.

<expL>

<expL> specifies the state of the REQUEST, EXECUTE, POKE or ADVISE options. Specify true (.T.) for <expL> to enable the option or false (.F.) to disable it.

DDESetTopic()

Purpose	In a dynamic data exchange (DDE) conversation, creates or releases a topic name from a service name
Syntax	DDESetTopic(<expC1>, <expC2> [, <expC3>])
Parameters	<expC1> The service name <expC2> The topic name <expC3> The name of a user-defined function (UDF)
Returns	Character
Remarks	FoxPro for Windows only
See Also	DDEEnabled(), DDELastError(), DDEPoke(), DDESetService()

Description After a topic name is created, any client requests to the topic name cause FoxPro to execute the UDF specified with <expC3>. The UDF is passed a set of parameters whose values are determined by the client request. The UDF return value is passed to the client with DDEPoke(). The return value is a logical value indicating whether the topic name can provide the service requested by the client.

DDESetTopic() returns true (.T.) if it successfully creates or releases the topic name. It returns false (.F.) if the topic name cannot be created or released. Use DDELastError() to determine why a topic name cannot be created or released.

Parameters <expC1>

Specify the service name with <expC1>. Additional service names can be created with DDESetService().

<expC2>

Specify the topic name to create or release with <expC2>. If you include <expC3>, DDESetService() creates the topic name <expC2>. If you omit <expC3>, the topic name <expC2> is released. If <expC2> is the null string, the UDF specified with <expC3> is executed for any topic name that isn't explicitly declared.

<expC3>

<expC3> is the name of the UDF executed when a client application makes a request to the topic name. If <expC3> is omitted, the topic name <expC2> is released from the service name.

When the UDF is executed, it is passed the following six parameters in the order given below:

Parameter	Contents
Channel Number	The client channel number
Action	ADVISE, EXECUTE, INITIATE, POKE, REQUEST, or TERMINATE
Item	The item name; for example, R1C1 for a Microsoft Excel spreadsheet cell
Data	Data from the client
Format	The data format; for example, CF_TEXT
Advise Status	The link type (0 = manual, 2 = notify or automatic)

The values of the Item, Data and Advise Status parameters depend on the Action parameter. The following table lists the Action parameter values and the values contained in the Item, Data and Advise Status parameters. A dash (—) indicates that the parameter value is the null string.

Action Value	Item Value	Data Value	Advise Status
INITIATE	—	Topic name	—
TERMINATE	—	—	—
POKE	Item name	New data	—
REQUEST	Item name	—	—
EXECUTE	—	New command	—
ADVISE	Item name	—	Link type

If the UDF successfully handles the client request, the UDF returns true (.T.). If the request can't be handled or an error occurs, the UDF returns false (.F.). If false is returned when the Action parameter value is INITIATE, the client topic name request is rejected. If false is returned when the value is POKE, REQUEST, or EXECUTE, the request is ignored. If false is returned when the value is ADVISE, the client request for a notify or automatic link is rejected.

Example The following example creates a basic sample server called MYSERVER that supports FoxPro for Windows command execution from a client application. The client application makes requests to MYSERVER through the DO topic, and macro substitution is used to execute the client's command.

```
*** Set FoxPro up as a DDE server ***

= DDESetService('myserver', 'DEFINE')
= DDESetService('myserver', 'EXECUTE', .T.)
= DDESetTopic('myserver', 'DO', 'DOTOPIC')
WAIT WINDOW 'Server portion service setup..' NOWAIT

*** Use FoxPro as a DDE client ***

dchannel = DDEInitiate('myserver', 'DO')
=DDEExecute(dchannel, 'WAIT WINDOW "Command Executed..."')
=DDETerminate(dchannel)

PROCEDURE dotopic
PARAMETERS channel, action, item, data, format, advise
mresult = .F.

*** It's necessary to return .T. from an      ***
*** INITIATE action or no connection is made ***

IF action = 'INITIATE'
    mresult = .T.
ENDIF

IF action = 'EXECUTE'
    &data
    mresult = .T.
ENDIF

IF action = 'TERMINATE'
    WAIT WINDOW 'Goodbye ... ' NOWAIT
    mresult = .T.
ENDIF
RETURN mresult
```

After running this example program, you have set up FoxPro service, which other applications can access. If you have Microsoft Excel, you can run the following macro:

```
mychan = INITIATE("myserver", "DO")
=EXECUTE(mychan, "WAIT WINDOW 'Hi, this is EXCEL speaking'")
=RETURN( )
```

DDETerminate()

Purpose	Closes a dynamic data exchange (DDE) channel established with DDEInitiate()
Syntax	DDETerminate(<expN> <expC>)
Parameters	<expN> The channel number to close <expC> The service name
Returns	Logical
Remarks	FoxPro for Windows only
See Also	DDEAbortTrans(), DDEAdvise(), DDEExecute(), DDEInitiate(), DDELastError(), DDEPoke(), DDERequest(), DDESetOption()

Description DDETerminate() closes a channel established with DDEInitiate(). If the channel is successfully closed, DDETerminate() returns true (.T.). If the channel cannot be closed, false (.F.) is returned.

Be sure to close channels as soon as they are no longer needed to conserve system resources.

All channels are automatically closed if you exit FoxPro for Windows by choosing **Exit** from the **File** menu or by issuing QUIT in the Command window or from within a program.

Parameters <expN>

Specify the channel number to close with <expN>.

<expC>

Include the service name <expC> to close all channels to the service.

DEACTIVATE MENU

Purpose	Deactivates a user-defined menu and removes it from the screen but doesn't remove the menu definition from memory
Syntax	DEACTIVATE MENU <menu name1> [, <menu name2> ...] ALL
See Also	ACTIVATE MENU, CLEAR ALL, CLEAR MENU, CREATE MENU, DEFINE MENU, HIDE MENU, RELEASE MENUS, SHOW MENU

Description DEACTIVATE MENU removes an active menu or set of menus from the desktop in FoxPro for MS-DOS, the main FoxPro window in FoxPro for Windows or a user-defined window without removing the popup definition from memory. A menu can be reactivated with ACTIVATE MENU and the menu name.



When you include the system menu bar (_MSYSMENU) in an application, you don't need to define, activate or deactivate the menu. Instead, issue SET SYSMENU AUTOMATIC. For more information about using the System menu, see the Menus chapter in the FoxPro *Developer's Guide*.

To release a specific menu or a set of menus from memory, use RELEASE MENUS. You can release all menus from memory with CLEAR MENUS or CLEAR ALL.

Program control is returned to the program line immediately following the line that activated the menu unless DEFINE MENU BAR is used to create the menu or ACTIVATE MENU NOWAIT is used to activate the menu.

Clauses **<menu name1> [, <menu name2> ...]**

These clauses specify the names of the menus to deactivate. You can deactivate a set of menus by including a list of menu names separated by commas.

ALL

Including ALL deactivates all active menus.

Example This example uses DEACTIVATE MENU to deactivate a menu and remove it from the screen.

The current system menu bar is saved to memory with SET SYSMENU SAVE, and all the system menu pads are removed with SET SYSMENU TO.

Two menu pads are created with DEFINE PAD, and DEFINE POPUP creates a popup for each menu pad. DEFINE BAR creates options on each of the popups. When a menu pad is chosen, ON PAD uses ACTIVATE POPUP to activate the corresponding popup. ACTIVATE MENU displays and activates the menu bar.

When an option is chosen from a popup, the CHOICE procedure is executed. CHOICE displays the name of the chosen option and the name of the popup containing the option. Program control continues on the line after ACTIVATE MENU.

Finally, the menu is deactivated and removed from the screen and is then released from memory with RELEASE MENUS EXTENDED.

```
*** Name this program DEACMENU.PRG ***
```

```
CLEAR
```

```
SET SYSMENU SAVE
SET SYSMENU TO
```

```
ON KEY LABEL ESC KEYBOARD CHR(13)
```

```
DEFINE MENU example BAR AT LINE 1
```

```
DEFINE PAD convpad OF example PROMPT '\<Conversions' COLOR SCHEME 3 ;
    KEY ALT+C, ''
DEFINE PAD cardpad OF example PROMPT 'Card \<Info' COLOR SCHEME 3 ;
    KEY ALT+I, ''
```

```
ON PAD convpad OF example ACTIVATE POPUP conversion
ON PAD cardpad OF example ACTIVATE POPUP cardinfo
```

```

DEFINE POPUP conversion MARGIN RELATIVE SHADOW COLOR SCHEME 4
DEFINE BAR 1 OF conversion PROMPT 'Ar\<ea' ;
    KEY CTRL+E, '^E'
DEFINE BAR 2 OF conversion PROMPT '\<Length' ;
    KEY CTRL+L, '^L'
DEFINE BAR 3 OF conversion PROMPT 'Ma\<ss' ;
    KEY CTRL+S, '^S'
DEFINE BAR 4 OF conversion PROMPT 'Spee\<d' ;
    KEY CTRL+D, '^D'
DEFINE BAR 5 OF conversion PROMPT '\<Temperature' ;
    KEY CTRL+T, '^T'
DEFINE BAR 6 OF conversion PROMPT 'T\<ime' ;
    KEY CTRL+I, '^I'
DEFINE BAR 7 OF conversion PROMPT 'Volu\<me' ;
    KEY CTRL+M, '^M'

```

```

ON SELECTION POPUP conversion;
    DO choice IN deacmenu WITH PROMPT( ), POPUP( )

```

```

DEFINE POPUP cardinfo MARGIN RELATIVE SHADOW COLOR SCHEME 4
DEFINE BAR 1 OF cardinfo PROMPT '\<View Charges' ;
    KEY ALT+V, ''
DEFINE BAR 2 OF cardinfo PROMPT 'View \<Payments' ;
    KEY ALT+P, ''
DEFINE BAR 3 OF cardinfo PROMPT 'Vie\<w Users' ;
    KEY ALT+W, ''
DEFINE BAR 4 OF cardinfo PROMPT '\<- '
DEFINE BAR 5 OF cardinfo PROMPT '\<Charges '

```

```

ON SELECTION POPUP cardinfo;
    DO choice IN deacmenu WITH PROMPT( ), POPUP( )

```

ACTIVATE MENU example

```

DEACTIVATE MENU example
RELEASE MENU example EXTENDED
SET SYSMENU NOSAVE
SET SYSMENU TO DEFAULT
ON KEY LABEL ESC

```

```

PROCEDURE choice
PARAMETERS mprompt, mpopup

```

```

WAIT WINDOW 'You chose ' + mprompt + ;
    ' from popup ' + mpopup NOWAIT

```

DEACTIVATE POPUP

Purpose	Deactivates a popup defined with DEFINE POPUP and removes it from the screen but doesn't remove the popup definition from memory
Syntax	DEACTIVATE POPUP <popup name1> [, <popup name2> ...] ALL
See Also	ACTIVATE POPUP, CLEAR ALL, CLEAR POPUPS, CREATE MENU, DEFINE POPUP, HIDE POPUP, RELEASE POPUPS, SHOW POPUP

Description DEACTIVATE POPUP removes an active menu popup or set of menu popups from the desktop in FoxPro for MS-DOS, the main FoxPro window FoxPro for Windows or a user-defined window without removing the popup definition from memory. A popup can be reactivated by using ACTIVATE POPUP and the menu popup name.

Use RELEASE POPUPS with the menu popup name to release a specific menu popup or a set of menu popups from memory. You can release all menu popups from memory with CLEAR POPUPS or CLEAR ALL.

Program control is returned to the line immediately following the line that activated the menu popup unless ACTIVATE POPUP NOWAIT is used to activate the menu.

Clauses <popup name1> [, <popup name2> ...]

These clauses specify the names of the menu popups to deactivate. You can deactivate a set of menu popups by including a list of menu popup names separated by commas.

ALL

Including ALL deactivates all active menu popups.

Example In this example, DEACTIVATE POPUP is used to deactivate a popup when you choose a special menu pad.

The current system menu bar is saved to memory with SET SYSMENU SAVE, and all the system menu pads are removed with SET SYSMENU TO.

Two new system menu pads are created with DEFINE PAD, and DEFINE POPUP creates a popup for each menu pad. DEFINE BAR creates options in each of the popups. When a menu pad is chosen, ON PAD uses ACTIVATE POPUP to activate the corresponding popup.

When an option is chosen from a popup, the CHOICE procedure is executed. CHOICE displays the name of the chosen option and the name of the popup containing the option.

If you choose the **Disable Conversions** option on the **Card Info** popup, the **Conversions** popup is deactivated with DEACTIVATE POPUP and released from memory with RELEASE POPUPS. The **Conversion** menu pad is removed from the menu bar with RELEASE PAD and the **Disable Conversions** option in the **Card Info** popup is removed with RELEASE BAR.

If you choose the **Exit** option on the **Card Info** popup, the FoxPro system menu is restored.

```
*** Name this program DEACTPOP.PRG ***
```

```
CLEAR
```

```
SET SYSMENU SAVE
```

```
SET SYSMENU TO
```

```
DEFINE PAD convpad OF _MSYSMENU PROMPT '\<Conversions' COLOR SCHEME 3 ;  
KEY ALT+C, ''
```

```
DEFINE PAD cardpad OF _MSYSMENU PROMPT 'Card \<Info' COLOR SCHEME 3 ;  
KEY ALT+I, ''
```

```
ON PAD convpad OF _MSYSMENU ACTIVATE POPUP conversion
```

```
ON PAD cardpad OF _MSYSMENU ACTIVATE POPUP cardinfo
```

```
DEFINE POPUP conversion MARGIN RELATIVE SHADOW COLOR SCHEME 4
```

```
DEFINE BAR 1 OF conversion PROMPT 'Ar\<ea' ;
```

```
KEY CTRL+E, '^E'
```

```
DEFINE BAR 2 OF conversion PROMPT '\<Length' ;
```

```
KEY CTRL+L, '^L'
```

```
DEFINE BAR 3 OF conversion PROMPT 'Ma\<ss' ;
```

```
KEY CTRL+S, '^S'
```

```
DEFINE BAR 4 OF conversion PROMPT 'Spee\<d' ;
```

```
KEY CTRL+D, '^D'
```

```
DEFINE BAR 5 OF conversion PROMPT '\<Temperature' ;
```

```
KEY CTRL+T, '^T'
```

```
DEFINE BAR 6 OF conversion PROMPT 'T\<ime' ;
```

```
KEY CTRL+I, '^I'
```

```
DEFINE BAR 7 OF conversion PROMPT 'Volu\<me' ;
```

```
KEY CTRL+M, '^M'
```

```

ON SELECTION POPUP conversion;
    DO choice IN deactpop WITH PROMPT( ), POPUP( )

DEFINE POPUP cardinfo MARGIN RELATIVE SHADOW COLOR SCHEME 4
DEFINE BAR 1 OF cardinfo PROMPT '\<View Charges' ;
    KEY ALT+V, ''
DEFINE BAR 2 OF cardinfo PROMPT 'View \<Payments' ;
    KEY ALT+P, ''
DEFINE BAR 3 OF cardinfo PROMPT 'Vie\<w Users' ;
    KEY ALT+W, ''
DEFINE BAR 4 OF cardinfo PROMPT '\-'
DEFINE BAR 5 OF cardinfo PROMPT '\<Charges '
DEFINE BAR 6 OF cardinfo PROMPT '\-'
DEFINE BAR 7 OF cardinfo PROMPT '\<Disable Conversions '
DEFINE BAR 8 OF cardinfo PROMPT 'E\<xit '

ON SELECTION POPUP cardinfo;
    DO choice IN deactpop WITH PROMPT( ), POPUP( )

PROCEDURE choice
PARAMETERS mprompt, mpopup

WAIT WINDOW 'You chose ' + mprompt + ;
    ' from popup ' + mpopup NOWAIT

IF mprompt = 'Disable Conversions'
    DEACTIVATE POPUP conversion
    RELEASE POPUP conversion
    RELEASE PAD convpad OF _MSYSMENU
    RELEASE BAR 7 OF cardinfo
ENDIF

IF mprompt = 'Exit'
    SET SYSMENU TO DEFAULT
ENDIF

```


DEACTIVATE WINDOW

Purpose	Deactivates specific user-defined windows or FoxPro system windows and removes them from the screen but not from memory
Syntax	DEACTIVATE WINDOW <window name1> [, <window name2> ...] ALL
See Also	ACTIVATE WINDOW, CLEAR WINDOWS, DEFINE WINDOW, HIDE WINDOW, RELEASE WINDOWS, SHOW WINDOW
Description	More than one window can be placed on the desktop in FoxPro for MS-DOS or the main FoxPro window in FoxPro for Windows at the same time, but output is directed only to the most recently activated window. When more than one window is present, deactivating the current output window clears the contents of the window, removes the window from the screen and sends subsequent output to the previously activated window. If there is no output window, output is directed to the desktop or main FoxPro window. Use CLEAR WINDOWS or RELEASE WINDOWS to remove windows from both the screen and memory.
Clauses	<window name1> [, <window name2> ...] These clauses specify one or more windows to deactivate. ALL Including ALL deactivates all active windows.
Example	In this example, a window named Output is defined and activated. After a record from INVOICES.DBF is displayed, the program waits for you to press a key and then deactivates the window. <pre> CLOSE DATABASES CLEAR DEFINE WINDOW output FROM 2,1 TO 13,75 TITLE 'Output' ; CLOSE FLOAT GROW SHADOW ZOOM ACTIVATE WINDOW output USE invoices DISPLAY WAIT RELEASE WINDOW output </pre>

DECLARE

Purpose	Creates one or more arrays
Syntax	DECLARE <array1> (<expN1> [, <expN2>]) [, <array2> (<expN3> [, <expN4>])] ...
See Also	APPEND FROM ARRAY, COPY TO ARRAY, DIMENSION, GATHER, PRIVATE, PUBLIC, SCATTER, SET COMPATIBLE, STORE

Description Use DECLARE to create one- or two-dimensional arrays. DECLARE is identical in operation and syntax to DIMENSION. See DIMENSION in this chapter for more information.

#DEFINE ... #UNDEF

Purpose	Creates and releases compile-time constants
Syntax	#DEFINE <constant name> <expr> #UNDEF <constant name>
See Also	COMPILE, #IF ... #ENDIF

Description The #DEFINE and #UNDEF preprocessor directives are used to create compile-time constants in programs. By creating constants with #DEFINE instead of using system memory variables, you can reduce memory consumption, increase performance and simplify programs.

To create a constant with #DEFINE, specify the constant's name with <constant name> and its value with <expr>. When the program is compiled, text substitution is performed and the constant value expression is substituted for the constant name wherever it appears in the program. You can stop the substitution for the constant by issuing #UNDEF.

Substitution occurs only in program lines that follow the #DEFINE statement that creates the constant and that precede the #UNDEF statement for that constant. The constant is available only to the program that creates the constant.

Clauses **<constant name>**

To create a compile-time constant with #DEFINE, specify its name with <constant name>. The constant name must be a legitimate FoxPro name that begins with a letter or an underscore and consists of up to 10 letters, digits or underscores. To improve program readability and simplify debugging, use a standard naming convention for your constant names.



Don't use FoxPro key words for constant names.

To stop text substitution for a constant created with #DEFINE, issue #UNDEF <constant name>.

#DEFINE ... #UNDEF

<expr>

Specify the value of the compile-time constant with <expr>. <expr> can be a name or an expression that evaluates to a character, numeric, date or logical value.

Example The following program creates a compile-time constant named MAXITEMS. This constant is used in a FOR ... NEXT loop to display the numbers 1 through 10.

```
#DEFINE maxitems 10

FOR i = 1 TO maxitems
    ? i
NEXT
```

DEFINE BAR

Purpose	Creates a menu option on a popup defined with DEFINE POPUP or a FoxPro menu system popup
Syntax	<pre> DEFINE BAR <expN1> <system option name> OF <popup name> PROMPT <expC1> [BEFORE <expN2> AFTER <expN3>] [KEY <key label> [, <expC2>]] [MARK <expC3>] [MESSAGE <expC4>] [SKIP [FOR <expL>]] [COLOR SCHEME <expN4> COLOR <color pair list>] </pre>
Remarks	RGB color pairs — FoxPro for Windows only
See Also	ACTIVATE POPUP, DEACTIVATE POPUP, DEFINE POPUP, HIDE POPUP, RELEASE BAR, SET MESSAGE, SHOW POPUP

Description DEFINE BAR is used with DEFINE POPUP to create popups. A popup is created and assigned a name with DEFINE POPUP. Menu options are placed on the popup with a series of DEFINE BARs.

When a menu option is chosen, another popup can be displayed or a routine can be executed. A popup that displays another popup or a menu bar is called a *cascading menu*. For more information on creating cascading menus, see ON BAR.

You can define more menu options than will fit on a popup. This creates a scrollable popup. Press Home to move to the first option or End to move to the last option. In FoxPro for MS-DOS, click on the top or bottom border of the popup to scroll through the options. In FoxPro for Windows, click the up or down arrow on the popup to scroll through the options.

Using the FoxPro system menu bar and its popups is the preferred method of including popups in your applications. If you use system menu bar popups, you don't have to define the popups. For more information on using the system menu bar and its popups, see SET SYSMENU in this chapter.

You can also create a popup that contains records or fields from a table/.DBF or a list of files available on disk. See the PROMPT FIELD, PROMPT STRUCTURE and PROMPT FILES clauses of DEFINE POPUP for more information.

Clauses

<expN1>

<expN1> is the bar number of the popup option you create with DEFINE BAR.

<system option name>

Options that are available in FoxPro system menu popups can also be placed on popups you create. You can place an option from a system menu popup on your popup by including <system option name>. For example, to provide access to the FoxPro Calendar/Diary from a popup named **Other**, you could create the option with the following command:

Not all system menu popups options are available for user-defined popups. Options on conditional menu pads (**Browse**, **Screen**, **Project**, etc.) are not available.

```
DEFINE BAR _MST_DIARY OF OTHER PROMPT 'Enter Appointments'
```

Whenever this option is chosen from the popup, the Calendar/Diary is opened.

Not all system menu popups are available for user-defined menus. Options on conditional menu pads (Browse, Screen, Project, etc.) are not available.

For a list of system menu names, use SYS(2013).

OF <popup name>

Specify the name of the popup on which the options are placed with <popup name>.

PROMPT <expC1>

PROMPT <expC1> specifies the text that appears in the option.

A separating bar can be created and placed between options by specifying a backslash and a dash (\-) for <expC1>. A separating bar is used to group options together. For example, including the following command in a popup definition creates a separating bar between the third and fifth options:

```
DEFINE BAR 4 OF <popup name> PROMPT '\-'
```

You can create a hot key for a menu option by placing a backslash and a less-than sign (\<) before the character you would like to be the hot key. A *hot key* is a key that you press to choose the corresponding option. For example:

```
DEFINE POPUP receive
DEFINE BAR 1 OF receive PROMPT '\<Invoices'
DEFINE BAR 2 OF receive PROMPT 'In\<quiry'
ACTIVATE POPUP receive
```

The first popup option, BAR 1, is chosen by pressing I; the second option, BAR 2, is chosen by pressing Q.

BEFORE <expN2> | AFTER <expN3>

An option can be placed on a popup relative to another option on the popup by including the BEFORE or AFTER clause. The popup must be created with DEFINE POPUP RELATIVE to include BEFORE or AFTER.

If BEFORE <expN2> is included, the option is placed to the left of the option specified with <expN2>. If AFTER <expN3> is included, the option is placed to the right of the option specified with <expN3>.

You can include _MFIRST and _MLAST in the BEFORE and AFTER clauses to reference the first and last items on a popup, respectively. If _MFIRST is included in the BEFORE clause, the popup option is the first option on the popup. If _MFIRST is included in the AFTER clause, the popup option is the second item on the popup. If _MLAST is included in the AFTER clause, the popup option is the last option on the popup. If _MLAST is included in the BEFORE clause, the option is the next-to-last option on the popup.

Popups created with DEFINE POPUP RELATIVE don't include space for undefined menu bar options. For example, if you define bars 1, 2, 4 and 5, a line is not reserved for option 3. You can later insert option 3 into the popup, which will expand to accommodate the third option.

Examples

Run the following program examples and note the differences in the order and placement of the options on each popup:

*** RELATIVE Example ***

```
DEFINE POPUP relatyес RELATIVE FROM 1,1

DEFINE BAR 4 OF relatyес PROMPT '4444'
DEFINE BAR 3 OF relatyес PROMPT '3333'
DEFINE BAR 2 OF relatyес PROMPT '2222'
DEFINE BAR 1 OF relatyес PROMPT '1111'
DEFINE BAR 6 OF relatyес PROMPT '6666' BEFORE 4

ACTIVATE POPUP relatyес
```

*** NON-RELATIVE Example ***

```
DEFINE POPUP notrelat FROM 1,10
DEFINE BAR 4 OF notrelat PROMPT '4444'
DEFINE BAR 3 OF notrelat PROMPT '3333'
DEFINE BAR 2 OF notrelat PROMPT '2222'
DEFINE BAR 1 OF notrelat PROMPT '1111'
DEFINE BAR 6 OF notrelat PROMPT '6666'
ACTIVATE POPUP notrelat
```

KEY <key label> [, <expC2>]

In addition to pressing a hot key, you can choose an option by pressing another key or a key combination. <key label> designates the key or key combination used to choose the option. See ON KEY LABEL in this chapter for a list of available keys and key combinations and their key label names.



If a keyboard macro is already defined with the same key label, the keyboard macro takes precedence, and the option cannot be chosen with the specified key or key combination.

On the popup, the key label is displayed to the right of an option name. Include <expC2> to replace the key label with your own text. For example, including KEY Ctrl+B places the text “Ctrl+B” on the popup to the right of the option name. If you specify KEY Ctrl+B, “^B”, “^B” will appear on the popup as a reminder of the key combination used to choose the option. The key label can be suppressed by including the null string for <expC2>.

MARK <expC3> — FoxPro for MS-DOS only

A mark character can be placed to the left of an option name on a popup with SET MARK OF. The default mark character is a diamond (◆) in FoxPro for MS-DOS and a check (✓) in FoxPro for Windows.

In FoxPro for MS-DOS, the MARK clause can be included to change the default mark character to <expC3>. (Specifying a mark character doesn't mark an option. Use SET MARK OF to mark an option.) If <expC3> includes more than one character, only the first character is used as the mark character.

The MARK clause is ignored in FoxPro for Windows.

Mark characters specified with a DEFINE BAR take precedence over mark characters specified with MARK in DEFINE POPUP. SET MARK OF is used to toggle mark characters on or off, and can also be used to specify a mark character for an individual popup option or for all popup options in FoxPro for MS-DOS.

MESSAGE <expC4>

A message can be displayed when an option is selected by including MESSAGE <expC4>. In FoxPro for MS-DOS, the message <expC4> is centered on the last line of the desktop by default; the message location can be changed with SET MESSAGE.

In FoxPro for Windows, the message is placed in the Windows-style status bar. If the Windows-style status bar has been turned off with SET STATUS BAR OFF, the message is centered on the last line of the main FoxPro window.

SKIP [FOR <expL>]

A popup option can be disabled based on a logical condition. If you include the SKIP FOR clause, the logical expression <expL> is evaluated, and based on the result (true or false), the option is disabled or enabled. If <expL> evaluates to true (.T.), the option is disabled and cannot be selected or chosen. If <expL> evaluates to false (.F.), the option is enabled.

You can also disable a popup option by placing a backslash (\) before the text of the prompt. For example:

```
DEFINE BAR 1 OF receive PROMPT '\Invoices'
```

The first menu option, BAR 1, appears in disabled colors and cannot be selected or chosen. In FoxPro for MS-DOS, a bar disabled with SKIP or the \ cannot be selected. In FoxPro for Windows, a disabled bar can be selected but it cannot be chosen — a message for a disabled bar is still displayed when it is selected.

**COLOR SCHEME <expN4>
| COLOR <color pair list>**

Use the COLOR or COLOR SCHEME clauses to specify the colors for an *individual* bar, overriding the default colors or the colors specified with DEFINE POPUP. The COLOR and COLOR SCHEME clauses of DEFINE POPUP let you specify the colors of *all* popup options, mark characters and messages.

A color scheme is a set of 10 predefined color pairs. The color pairs in a color scheme can be changed with SET COLOR OF SCHEME. In FoxPro for MS-DOS the color pairs in a color scheme can also be changed in the Color Picker.

A color pair is a set of two letters separated by a forward slash. The first color letter specifies the foreground color and the second letter specifies the background color.

For example, this color pair specifies a red foreground on a white background:

R/W

For a list of colors and their corresponding color letters, see the SET COLOR commands section in this chapter or the Color Table by Color Pair topic in the online help facility.

In FoxPro for Windows, a color pair can also be specified with a set of 6 RGB (Red Green Blue) color values separated by commas. The first 3 color values specify the foreground color and the second 3 color values specify the background color. The color values can range from 0 through 255.

In FoxPro for Windows, the R/W color pair in the example above can also be specified with this RGB color pair:

RGB(255,0,0,255,255,255)

By default, the colors of popup options are determined by color scheme 2 of the current color set.

Example This example uses DEFINE BAR to create options on menu popups.

The current system menu bar is first saved to memory with SET SYSMENU SAVE, and then all system menu pads are removed with SET SYSMENU TO.

Two new system menu pads are created with DEFINE PAD, and DEFINE POPUP is used to create a popup for each menu pad. DEFINE BAR is used to create options on each of the popups. When a menu pad is chosen, ON PAD uses ACTIVATE POPUP to activate the corresponding popup.

When an option is chosen from a popup, ON SELECTION POPUP uses PROMPT() and POPUP() to pass the option number and popup name to the CHOICE procedure. CHOICE displays the prompt of the chosen option and the name of the popup containing the option. If the **Exit** option is chosen from the **Card Info** popup, the original FoxPro system menu is restored.

```
*** Name this program DEFINBAR.PRG ***
```

```
CLEAR
```

```
SET SYSMENU SAVE
SET SYSMENU TO
```

```
DEFINE PAD convpad OF _MSYMENU PROMPT '\<Conversions' COLOR SCHEME 3 ;
    KEY ALT+C, ''
DEFINE PAD cardpad OF _MSYMENU PROMPT 'Card \<Info' COLOR SCHEME 3 ;
    KEY ALT+I, ''
```

```
ON PAD convpad OF _MSYMENU ACTIVATE POPUP conversion
ON PAD cardpad OF _MSYMENU ACTIVATE POPUP cardinfo
```

```
DEFINE POPUP conversion MARGIN RELATIVE SHADOW COLOR SCHEME 4
DEFINE BAR 1 OF conversion PROMPT 'Ar\<ea' ;
    KEY CTRL+E, '^E'
DEFINE BAR 2 OF conversion PROMPT '\<Length' ;
    KEY CTRL+L, '^L'
DEFINE BAR 3 OF conversion PROMPT 'Ma\<ss' ;
    KEY CTRL+S, '^S'
DEFINE BAR 4 OF conversion PROMPT 'Spee\<d' ;
    KEY CTRL+D, '^D'
DEFINE BAR 5 OF conversion PROMPT '\<Temperature' ;
    KEY CTRL+T, '^T'
DEFINE BAR 6 OF conversion PROMPT 'T\<ime' ;
    KEY CTRL+I, '^I'
```

DEFINE BAR

```
DEFINE BAR 7 OF conversion PROMPT 'Volu\<me' ;
    KEY CTRL+M, '^M'

ON SELECTION POPUP conversion;
    DO choice IN definbar WITH PROMPT( ), POPUP( )

DEFINE POPUP cardinfo MARGIN RELATIVE SHADOW COLOR SCHEME 4
DEFINE BAR 1 OF cardinfo PROMPT '\<View Charges' ;
    KEY ALT+V, ''
DEFINE BAR 2 OF cardinfo PROMPT 'View \<Payments' ;
    KEY ALT+P, ''
DEFINE BAR 3 OF cardinfo PROMPT 'Vie\<w Users' ;
    KEY ALT+W, ''
DEFINE BAR 4 OF cardinfo PROMPT '\-'
DEFINE BAR 5 OF cardinfo PROMPT '\<Charges '
DEFINE BAR 6 OF cardinfo PROMPT '\-'
DEFINE BAR 7 OF cardinfo PROMPT 'E\<xit '

ON SELECTION POPUP cardinfo;
    DO choice IN definbar WITH PROMPT( ), POPUP( )

PROCEDURE choice
PARAMETERS mprompt, mpopup

WAIT WINDOW 'You chose ' + mprompt + ;
    ' from popup ' + mpopup NOWAIT

IF mprompt = 'Exit'
    SET SYSMENU TO DEFAULT
ENDIF
```

DEFINE BOX

Purpose	Draws a box around printed text
Syntax	DEFINE BOX FROM <column1> TO <column2> HEIGHT <expN1> [AT LINE <expN2>] [SINGLE DOUBLE <border string>]
Remarks	Included for backward compatibility — use the Screen Builder instead
See Also	@ ... BOX, _BOX, PCOL(), PROW(), _PSPACING, SET BORDER, SET PRINTER

Description This command draws a box around text only in printed reports and only if the BOX system memory variable is set to true (.T.). Use @ ... BOX or @ ... TO to draw boxes on the desktop in FoxPro for MS-DOS, the main FoxPro window in FoxPro for Windows or a user-defined window.

A box can be printed only if SET PRINTER is ON. Boxes can be drawn only around output created with the ? or ?? command. Boxes cannot be drawn around output created with @ ... SAY.

Two functions that are useful with DEFINE BOX are PCOL() and PROW(). These functions return the current print column and print row positions.

Clauses **FROM <column1> TO <column2>**
The upper-left corner of the box appears in print column <column1>. The upper-right corner appears in column <column2>.

HEIGHT <expN1>
The height of the box is specified with <expN1>. Box height is affected by the value of the system memory variable _PSPACING. For example, if <expN1> is set to 5 and _PSPACING is 3, the height of the box is 15 lines.

AT LINE <expN2>
The top of the box is drawn beginning at the current print line unless you include AT LINE <expN2> in DEFINE BOX. <expN2> is the number of the line at which you want the top of the box to be drawn.

SINGLE | DOUBLE | <border string>

The default border of the box is a single line. You can specify a different border style with the SINGLE, DOUBLE or <border string> clause. Include SINGLE to draw a single-line border; include DOUBLE to draw a double-line border. You can define your own border with <border string>. The syntax for the border string is identical to that in SET BORDER. See SET BORDER in this chapter for more information.

DEFINE MENU

Purpose	Creates a menu bar
Syntax	DEFINE MENU <menu name> [BAR [AT LINE <expN1>]] [IN [WINDOW] <window name> IN SCREEN] [KEY <key label>] [MARK <expC1>] [MESSAGE <expC2>] [NOMARGIN] [COLOR SCHEME <expN2> COLOR <color pair list>]
Remarks	RGB color pairs — FoxPro for Windows only
See Also	ACTIVATE MENU, CNTPAD(), CREATE MENU, DEACTIVATE MENU, DEFINE PAD, GETPAD(), HIDE MENU, MRKPAD(), ON PAD, ON SELECTION PAD, PRMPAD(), RELEASE PAD, SET MARK OF, SET SYSMENU, SHOW MENU

Description A menu system usually contains a horizontal menu bar with popups under each of the menu bar pads. As you move from pad to pad on the menu bar, a popup appears under each pad. You can then choose an option from a popup.

You use DEFINE MENU to create the menu bar of the menu system you're building. You define each of the pads on the menu bar using a series of DEFINE PADS, and you specify which popup is displayed under each menu pad with ON PAD ... ACTIVATE POPUP. Then, you create a popup for each of the menu pads using DEFINE POPUP and display the entire menu system with ACTIVATE MENU.

Choosing a menu bar pad doesn't have to activate a popup. When a pad is chosen, another menu bar can be activated with ON PAD ... ACTIVATE MENU, or a routine can be executed with ON SELECTION PAD ... DO.



Using the FoxPro system menu bar is the preferred method of placing menus in your applications. If you use the FoxPro system menu bar, you don't need to issue DEFINE MENU.

Clauses **<menu name>**
 <menu name> is the name of the menu bar you want to create.

BAR [AT LINE <expN1>]

Including the BAR clause creates a bar menu that behaves like the FoxPro System menu bar. The menu bar has these characteristics:

- Once a selection is made, the menu bar is no longer active.
- A horizontal menu bar one line high is drawn across the width of the desktop in FoxPro for MS-DOS, the main FoxPro window in FoxPro for Windows or the user-defined window it is placed in.
- The placement of the pads is handled automatically.
- If the size or number of pads you define exceeds the size of the screen or a window in which the menu bar is placed, the menu bar scrolls.

You can specify the row in which the menu is placed with the AT LINE clause. The row number is specified with <expN1>.

IN [WINDOW] <window name> | IN SCREEN

By default, menu bars are placed on the desktop or main FoxPro window unless there is an active user-defined window. If there is an active user-defined window, the menu bar is placed in the active window.

You can place a menu bar in a specific user-defined window by including the IN or IN WINDOW clause in DEFINE MENU. Specify the name of the window in which you want to place the menu bar with <window name>.

Include IN SCREEN to explicitly place the menu bar on the desktop or the main FoxPro window.

KEY <key label>

A menu bar can be activated by pressing a key or a combination of keys. <key label> specifies the key or key combination used to activate the menu bar. See ON KEY LABEL in this chapter for a list of available keys and key combinations and their key label names. Including the KEY clause is equivalent to issuing the following command:

```
ON KEY LABEL <key label> ACTIVATE MENU <menu name>
```



If a keyboard macro is already defined with the same key label, the keyboard macro takes precedence, and the menu bar cannot be activated with the specified key or key combination.

MARK <expC1> — FoxPro for MS-DOS only

A mark character can be placed to the left of each pad in a menu bar with SET MARK OF. The default mark character is a diamond (♦) in FoxPro for MS-DOS and a check (✓) in FoxPro for Windows. Include the MARK clause to change the default mark character to <expN3>. (Specifying a mark character doesn't mark a pad. Use SET MARK OF to mark a pad.) If <expC1> includes more than one character, only the first character is used as the mark character.

You can use the mark character <expC1> to indicate that a certain condition exists — a popup is available for the pad, an option controlled by the pad is on or off, and so forth.

The MARK clause is ignored in FoxPro for Windows.

The MARK clause sets the mark character for *all* pads in the menu bar. Mark characters specified with DEFINE PAD take precedence over mark characters specified with the MARK clause in DEFINE MENU. SET MARK OF is used to toggle marks on or off and can also be used to specify a mark character for an individual menu pad or for all menu pads.

MESSAGE <expC2>

A message can be displayed when a pad is selected by including MESSAGE <expC2>. In FoxPro for MS-DOS, the message <expC2> is centered on the last line of the desktop by default; the message location can be changed with SET MESSAGE.

In FoxPro for Windows, the message is placed in the Windows-style status bar. If the Windows-style status bar has been turned off with SET STATUS BAR OFF, the message is centered on the last line of the main FoxPro window.

NOMARGIN

By default, one space is placed to the left and right of each menu pad. Include NOMARGIN to remove the spaces.

**COLOR SCHEME <expN2>
| COLOR <color pair list>**

You can specify the color of all elements of the menu bar by including the number of an existing color scheme in the COLOR SCHEME clause or a set of color pairs in the COLOR clause.

A color scheme is a set of 10 predefined color pairs. The color pairs in a color scheme can be changed with SET COLOR OF SCHEME. In FoxPro for MS-DOS the color pairs in a color scheme can also be changed in the Color Picker.

A color pair is a set of two letters separated by a forward slash. The first color letter specifies the foreground color and the second letter specifies the background color.

For example, this color pair specifies a red foreground on a white background:

R/W

For a list of colors and their corresponding color letters, see the SET COLOR commands section in this chapter or the Color Table by Color Pair topic in the online help facility.

In FoxPro for Windows, a color pair can also be specified with a set of 6 RGB (Red Green Blue) color values separated by commas. The first 3 color values specify the foreground color and the second 3 color values specify the background color. The color values can range from 0 through 255.

In FoxPro for Windows, the R/W color pair in the example above can also be specified with this RGB color pair:

RGB(255,0,0,255,255,255)

By default, the colors of menu bars created with DEFINE MENU are controlled by color scheme 2.

Example This example uses DEFINE MENU to create a user-defined menu system.

The current system menu bar is first saved to memory with SET SYSMENU SAVE, and then the system menu pads are cleared with SET SYSMENU TO.

DEFINE MENU creates the menu bar, and two menu pads are created with DEFINE PAD. DEFINE POPUP creates a popup for each menu pad. DEFINE BAR creates options on each of the popups. When a menu pad is chosen, ON PAD uses ACTIVATE POPUP to activate the corresponding popup. ACTIVATE MENU displays and activates the menu bar.

When an option is chosen from a popup, the CHOICE procedure is executed. CHOICE displays the name of the chosen option and the name of the popup containing the option.

```
*** Name this program DEFIMENU.PRG ***
```

```
CLEAR
```

```
SET SYSMENU SAVE
```

```
SET SYSMENU TO
```

```
ON KEY LABEL ESC KEYBOARD CHR(13)
```

```
DEFINE MENU example BAR AT LINE 1
```

```
DEFINE PAD convpad OF example PROMPT '\<Conversions' COLOR SCHEME 3 ;  
KEY ALT+C, ''
```

```
DEFINE PAD cardpad OF example PROMPT 'Card \<Info' COLOR SCHEME 3 ;  
KEY ALT+I, ''
```

```
ON PAD convpad OF example ACTIVATE POPUP conversion
```

```
ON PAD cardpad OF example ACTIVATE POPUP cardinfo
```

```
DEFINE POPUP conversion MARGIN RELATIVE SHADOW COLOR SCHEME 4
```

```
DEFINE BAR 1 OF conversion PROMPT 'Ar\<ea' ;
```

```
KEY CTRL+E, '^E'
```

```
DEFINE BAR 2 OF conversion PROMPT '\<Length' ;
```

```
KEY CTRL+L, '^L'
```

```
DEFINE BAR 3 OF conversion PROMPT 'Ma\<ss' ;
```

```
KEY CTRL+S, '^S'
```

```
DEFINE BAR 4 OF conversion PROMPT 'Spee\<d' ;
```

```
KEY CTRL+D, '^D'
```

```
DEFINE BAR 5 OF conversion PROMPT '\<Temperature' ;
```

```
KEY CTRL+T, '^T'
```

DEFINE MENU

```
DEFINE BAR 6 OF conversion PROMPT 'T\<ime' ;
    KEY CTRL+I, '^I'
DEFINE BAR 7 OF conversion PROMPT 'Volu\<me' ;
    KEY CTRL+M, '^M'

ON SELECTION POPUP conversion;
    DO choice IN defimenu WITH PROMPT( ), POPUP( )

DEFINE POPUP cardinfo MARGIN RELATIVE SHADOW COLOR SCHEME 4
DEFINE BAR 1 OF cardinfo PROMPT '\<View Charges' ;
    KEY ALT+V, ''
DEFINE BAR 2 OF cardinfo PROMPT 'View \<Payments' ;
    KEY ALT+P, ''
DEFINE BAR 3 OF cardinfo PROMPT 'Vie\<w Users' ;
    KEY ALT+W, ''
DEFINE BAR 4 OF cardinfo PROMPT '\-'
DEFINE BAR 5 OF cardinfo PROMPT '\<Charges '

ON SELECTION POPUP cardinfo;
    DO choice IN defimenu WITH PROMPT( ), POPUP( )

ACTIVATE MENU example

DEACTIVATE MENU example
RELEASE MENU example EXTENDED
SET SYSMENU TO DEFAULT
ON KEY LABEL ESC

PROCEDURE choice
PARAMETERS mprompt, mpopup

WAIT WINDOW 'You chose ' + mprompt + ;
    ' from popup ' + mpopup NOWAIT
```

DEFINE PAD

Purpose	Creates a pad on a user-defined menu bar or a FoxPro system menu bar
Syntax	DEFINE PAD <pad name> OF <menu name> PROMPT <expC1> [AT <row, column>] [BEFORE <pad name> AFTER <pad name>] [KEY <key label> [, <expC2>]] [MARK <expC3>] [SKIP [FOR <expL>]] [MESSAGE <expC4>] [COLOR SCHEME <expN> COLOR <color pair list>]
Remarks	RGB color pairs — FoxPro for Windows only
See Also	ACTIVATE MENU, CREATE MENU, DEACTIVATE MENU, DEFINE MENU, GETPAD(), HIDE MENU, MRKPAD(), ON PAD, ON SELECTION PAD, PRMPAD(), SET MARK OF, SET MESSAGE, SET SYSMENU, SHOW MENU

Description DEFINE PAD is used with DEFINE MENU to create a menu system. DEFINE PAD creates a *menu pad*, which is an option on a menu bar that can be selected (highlighted) and chosen. When a menu pad is chosen, a popup can be displayed with more options, another menu bar can be displayed, or a routine can be executed.

Every pad you place on a menu bar requires its own DEFINE PAD command. A menu bar must be defined before you can place pads on it, and you must include the menu bar name in DEFINE PAD.

Using the FoxPro system menu bar is the preferred method of including menus in your applications. If you use the system menu bar (_MSYSMENU), you don't have to first define the menu. For more information on using the system menu bar, see SET SYSMENU in this chapter.

Clauses **<pad name>**
 <pad name> is the name of the menu pad you want to create.

OF <menu name>
 <menu name> is the name of the menu bar in which the pad is to appear.

PROMPT <expC1>

PROMPT <expC1> specifies the text that appears in the menu pad.

You can create a hot key for a menu pad by placing a backslash and a less-than sign (\<) before the character you would like to be the hot key. A *hot key* is key that can be pressed to choose the corresponding menu pad. For example:

```
DEFINE MENU receive
DEFINE PAD invoice OF receive PROMPT "\<Invoices"
DEFINE PAD inquire OF receive PROMPT "In\<quiry"
ACTIVATE MENU receive
```

The first pad, INVOICE, is chosen by pressing I; the second pad, INQUIRE, is chosen by pressing Q.

AT <row, column>

You can designate where a pad is displayed by including the AT clause in DEFINE PAD. <row, column> are the coordinates of the upper-left corner of the pad on the desktop in FoxPro for MS-DOS, in the main FoxPro window in FoxPro for Windows, or in a user-defined window. To place a menu bar in a user-defined window, the menu bar must be created with DEFINE MENU IN WINDOW.

If you omit the AT clause, the upper-left corner of the first menu pad is placed on row 0 of the desktop, main FoxPro window or user-defined window. The next menu pad is placed on the right of the first pad on row 0, and so on.



You cannot include the AT clause to specify a location for pads in menu bars created with the BAR clause in DEFINE MENU.

Pads are activated in the order in which they are created.

BEFORE <pad name> | AFTER <pad name>

Include the BEFORE or AFTER clause to specify the location and activation order of a pad in relation to other pads in a menu bar. The menu bar pad you specify in a BEFORE or AFTER clause should be defined before it is referenced. If the pad isn't defined, the placement of the pad is determined by the order in which it is defined or by a location specified with the AT clause.

For a menu bar pad created without the AT clause, a BEFORE or AFTER clause determines the pad's physical placement and activation order. If BEFORE is included, the pad is inserted into the menu bar to the left of the pad specified with <pad name>. If AFTER is included, the pad is inserted into the menu bar to the right of the pad specified with <pad name>. The activation order corresponds to the physical order of the pads in the menu bar.

For menu bars created without BAR, a BEFORE or AFTER clause determines the activation order of the pad. The physical placement of a pad is determined by the location specified with the AT clause.

Examples

Run the following two program examples and note the differences in pad placement and activation order when menu bar pads are defined with and without the AT clause:

*** Program Example 1 without ATs ***

```
DEFINE MENU beforafter
```

```
  DEFINE PAD one    OF beforafter PROMPT '1111'
```

```
  DEFINE PAD two    OF beforafter PROMPT '2222'
```

```
  DEFINE PAD three  OF beforafter PROMPT '3333'
```

```
  DEFINE PAD four   OF beforafter PROMPT '4444' BEFORE two
```

```
ACTIVATE MENU beforafter
```

*** Program Example 2 with ATs ***

```
DEFINE MENU beforafter
```

```
  DEFINE PAD one    OF beforafter PROMPT '1111' AT 1,5
```

```
  DEFINE PAD two    OF beforafter PROMPT '2222' AT 1,15
```

```
  DEFINE PAD three  OF beforafter PROMPT '3333' AT 1,25
```

```
  DEFINE PAD four   OF beforafter PROMPT '4444' BEFORE two AT 1,35
```

```
WAIT WINDOW 'Press Esc to erase menu' NOWAIT
```

```
ACTIVATE MENU beforafter
```

KEY <key label> [, <expC2>]

In addition to pressing a hot key, you can choose a pad by pressing another key or a key combination. <key label> specifies the key or key combination used to choose the pad. See ON KEY LABEL in this chapter for a list of available keys and key combinations and their key label names.



If a keyboard macro is already defined with the same key label, the keyboard macro takes precedence, and the pad cannot be chosen with the specified key or key combination.

The key label is placed to the right of pads in menu bars created *without* the BAR clause. The key label isn't displayed in menu bars created with the BAR clause or for pads in the system menu bar. Include <expC2> to replace the key label with your own text. For example, including KEY Ctrl+B places the text "Ctrl+B" to the right of the pad. If you instead specify KEY Ctrl+B, "^B", "^B" will be displayed as a reminder of the key combination used to choose the pad. The key label can be suppressed by including the null string for <expC2>.

MARK <expC3> — FoxPro for MS-DOS only

A mark character can be placed to the left of a pad in a menu bar with SET MARK OF. The default mark character is a diamond (♦) in FoxPro for MS-DOS and a check (✓) in FoxPro for Windows.

In FoxPro for MS-DOS, the MARK clause can be included to change the default mark character to <expC3>. (Specifying a mark character doesn't mark a pad. Use SET MARK OF to mark a pad).

The MARK clause is ignored in FoxPro for Windows.

You can use a mark character to indicate that a certain condition exists — a popup is available for the pad, an option controlled by the pad is on or off, and so forth.

If <expC3> includes more than one character, only the first character is used as the mark character.

Mark characters specified with DEFINE PAD take precedence over mark characters specified with the MARK clause in DEFINE MENU. SET MARK OF is used to toggle marks on or off and can also be used to specify a mark character for an individual menu pad or for all menu pads.

SKIP [FOR <expL>]

A pad can be disabled based on a logical condition. If you include SKIP FOR <expL>, the logical expression <expL> is evaluated, and based on the result (true or false), the pad is disabled or enabled. If <expL> evaluates to true (.T.), the pad is disabled and cannot be selected or chosen. If <expL> evaluates to false (.F.), the pad is enabled.

Another method for disabling a menu pad is to place a backslash (\) before the text of the prompt. For example:

```
DEFINE PAD pageaccts OF receive PROMPT '\Age Accounts'
```

The pad **Pageaccts** is displayed in the menu bar in disabled colors and cannot be selected or chosen.

In FoxPro for MS-DOS, a menu pad disabled with SKIP or the \ cannot be selected. In FoxPro for Windows, a disabled menu pad can be selected but it cannot be chosen — a message for a disabled menu pad is still displayed when it is selected.

MESSAGE <expC4>

A message can be displayed when a pad is selected by including MESSAGE <expC4>. In FoxPro for MS-DOS, the message <expC4> is centered on the last line of the desktop by default; the message location can be changed with SET MESSAGE.

In FoxPro for Windows, the message is placed in the Windows-style status bar. If the Windows-style status bar has been turned off with SET STATUS BAR OFF, the message is centered on the last line of the main FoxPro window.

COLOR SCHEME <expN>**| COLOR <color pair list>**

Include the number of an existing color scheme in the COLOR SCHEME clause or a set of color pairs in the COLOR clause to specify the colors of *all* pads, mark characters and messages in a menu bar. You can also specify the colors of an *individual* pad, overriding the default colors or the colors specified with DEFINE MENU.

A color scheme is a set of 10 predefined color pairs. The color pairs in a color scheme can be changed with SET COLOR OF SCHEME. In FoxPro for MS-DOS the color pairs in a color scheme can also be changed in the Color Picker.

A color pair is a set of two letters separated by a forward slash. The first color letter specifies the foreground color and the second letter specifies the background color.

For example, this color pair specifies a red foreground on a white background:

```
R/W
```

For a list of colors and their corresponding color letters, see the SET COLOR commands section in this chapter or the Color Table by Color Pair topic in the online help facility.

In FoxPro for Windows, a color pair can also be specified with a set of 6 RGB (Red Green Blue) color values separated by commas. The first 3 color values specify the foreground color and the second 3 color values specify the background color. The color values can range from 0 through 255.

In FoxPro for Windows, the R/W color pair in the example above can also be specified with this RGB color pair:

```
RGB(255,0,0,255,255,255)
```

By default, the colors of pads in menu bars are determined by color scheme 2 of the current color set.

Example

This example uses DEFINE PAD to place menu pads in the FoxPro system menu bar.

The current system menu bar is first saved to memory with SET SYSMENU SAVE, and then all system menu pads are removed with SET SYSMENU TO.

Several system menu pads are created with DEFINE PAD. When a menu pad is chosen, the CHOICE procedure is executed. CHOICE displays the name of the chosen menu pad and the name of the menu bar, and toggles the menu pad's mark character on and off. If the **Exit** menu pad is chosen, the original FoxPro system menu is restored.

```
*** Name this program DEFINPAD.PRG ***
```

```
CLEAR
SET TALK OFF
SET SYSMENU SAVE
SET SYSMENU TO
```

```
PUBLIC markpad
markpad = .T.
```

```

DEFINE PAD syspad    OF _MSYSMENU PROMPT '\<System'  COLOR SCHEME 3 ;
    KEY ALT+S, ''
DEFINE PAD editpad   OF _MSYSMENU PROMPT '\<Edit'    COLOR SCHEME 3 ;
    KEY ALT+E, ''
DEFINE PAD recordpad OF _MSYSMENU PROMPT '\<Record'  COLOR SCHEME 3 ;
    KEY ALT+R, ''
DEFINE PAD windowpad OF _MSYSMENU PROMPT '\<window'  COLOR SCHEME 3 ;
    KEY ALT+W, ''
DEFINE PAD reportpad OF _MSYSMENU PROMPT 'Re\<ports' COLOR SCHEME 3 ;
    KEY ALT+P, ''
DEFINE PAD exitpad   OF _MSYSMENU PROMPT 'E\<xit'   COLOR SCHEME 3 ;
    KEY ALT+X, ''

ON SELECTION MENU _MSYSMENU ;
    DO choice IN definpad WITH PAD( ), MENU( )

PROCEDURE choice
    PARAMETER mpad, mmenu

    WAIT WINDOW 'You chose ' + mpad + ;
        ' from menu ' + mmenu NOWAIT
    SET MARK OF PAD (mpad) OF _MSYSMENU TO ;
        ! MRKPAD('_MSYSMENU', mpad)
    markpad = ! markpad

    IF mpad = 'EXITPAD'
        SET SYSMENU TO DEFAULT
    ENDIF

```

DEFINE POPUP

Purpose	Creates a popup
Syntax	<pre> DEFINE POPUP <popup name> [FROM <row1, column1>] [TO <row2, column2>] [IN [WINDOW] <window name> IN SCREEN] [FOOTER <expC1>] [KEY <key label>] [MARGIN] [MARK <expC2>] [MESSAGE <expC3>] [MOVER] [MULTISELECT] [PROMPT FIELD <expr> PROMPT FILES [LIKE <skel>] PROMPT STRUCTURE] [RELATIVE] [SCROLL] [SHADOW] [TITLE <expC4>] [COLOR SCHEME <expN> COLOR <color pair list>] </pre>
Remarks	RGB color pairs — FoxPro for Windows only
See Also	ACTIVATE POPUP, CNTBAR(), CREATE MENU, DEFINE BAR, GETBAR(), HIDE POPUP, MOVE POPUP, MRKBAR(), ON BAR, ON SELECTION BAR, PRMBAR(), SET MARK OF, SET MESSAGE, SIZE POPUP, SHOW POPUP

Description DEFINE POPUP creates a popup, which can contain a list of options that you define, the records from a field, a list of files, or a list of fields from a table/.DBF. To place a set of options that you define on a popup, use a series of DEFINE BARS. To place records, files or fields on a popup, use the PROMPT FIELD, PROMPT FILES or PROMPT STRUCTURE clause of DEFINE POPUP.

When the popup is activated and displayed with ACTIVATE POPUP, you can choose one of the options. Depending on the option chosen, a routine can be executed or another popup can be displayed and activated. A popup that displays another popup or a menu bar when an option is chosen is called a *cascading menu*. For more information on creating cascading menus, see ON BAR in this chapter.

Clauses **<popup name>**

<popup name> is the name of the popup you want to create.

FROM <row1, column1>
TO <row2, column2>

You can specify where the popup is placed by including the FROM clause. The upper-left corner of the popup is placed at the coordinates specified by <row1, column1>. If the FROM clause is omitted, FoxPro places the upper-left corner of the popup in the first row and first column of the desktop, main FoxPro window or user-defined window.

You can also include <row2, column2> to specify the location of the lower-right corner of the popup to create a popup with a specific size.

If you include FROM <row1, column1> and omit TO <row2, column2>, FoxPro automatically sizes the popup. The popup is as wide as the longest menu option (if the options are created with DEFINE BAR) and as long as needed to display all of the menu options. The popup length is limited by the size of the desktop, main FoxPro window or user-defined window. If a popup isn't large enough to contain all of the menu options, the menu options scroll within the popup.

If you include both FROM and TO clauses to create a popup with a specific size, you can then use ACTIVATE POPUP NOWAIT to display the popup without having first defined bars for the popup. This lets you display a popup and then place options on it.

IN [WINDOW] <window name> | IN SCREEN

By default, popups are placed on the desktop or main FoxPro window unless there is an active user-defined window. If there is an active user-defined window, the popup is placed in the active window.

You can place a popup in a specific user-defined window by including the IN or IN WINDOW clause. Specify the name of the window in which you want to place the popup with <window name>.

You can explicitly place the popup on the desktop or main FoxPro window by including IN SCREEN.

FOOTER <expC1>

You can create a footer that is centered in the bottom border of the popup. Include the FOOTER key word followed by <expC1>, which is the text you want to display in the bottom border of the popup. If the footer is wider than the popup, the footer is truncated.

The FOOTER clause is ignored in FoxPro for Windows.

KEY <key label>

A popup can be activated by pressing a key or a combination of keys. <key label> specifies the key or key combination used to activate the popup. See ON KEY LABEL in this chapter for a list of available keys and key combinations and their key label names. Including KEY is equivalent to issuing the following command:

```
ON KEY LABEL <key label> ACTIVATE MENU <menu name>
```



If a keyboard macro is already defined with the same key label, the keyboard macro takes precedence, and the popup cannot be activated with the specified key or key combination.

MARGIN

Including MARGIN places one extra space to the left and right of each popup option. A mark character is displayed in the space to the left of an option, and an arrow indicating an additional cascading menu is available is displayed to the right of an option. If MARGIN isn't included, the mark character overwrites the first character of the option name; a hierarchical arrow overwrites the last character of the option name.

MARK <expC2> — FoxPro for MS-DOS only

A mark character can be placed to the left of each option name on a popup with SET MARK OF. The default mark character is a diamond (◆) in FoxPro for MS-DOS and a check (✓) in FoxPro for Windows.

In FoxPro for MS-DOS, MARK <expC2> can be included to change the default mark character to <expC2>. (Specifying a mark character doesn't mark an option. Use SET MARK OF to mark an option.) If <expC2> includes more than one character, only the first character is used as the mark character.

The MARK clause is ignored in FoxPro for Windows.

The MARK clause sets the mark character for *all* options on the popup. Mark characters specified with DEFINE BARS take precedence over mark characters specified with the MARK clause in DEFINE POPUP. SET MARK OF is used to toggle mark characters on or off and can also be used to specify a mark character for an individual popup option or for all popup options.

MESSAGE <expC3>

A message can be displayed when the popup is active by including MESSAGE <expC3>. In FoxPro for MS-DOS, the message <expC3> is centered on the last line of the desktop by default; the message location can be changed with SET MESSAGE.

In FoxPro for Windows, the message is placed in the Windows-style status bar. If the Windows-style status bar has been turned off with SET STATUS BAR OFF, the message is centered on the last line of the main FoxPro window.

MOVER

Options on a popup can be rearranged within the popup if MOVER is included. When MOVER is included, a double-headed arrow (↕) is placed to the left of each option on the popup.

To move an option with the mouse, click on the double-headed arrow and drag the option up or down. With the keyboard, first select an option, and then press Ctrl+↑ or Ctrl+↓ to move the option up or down.

You cannot rearrange options on a popup created with a PROMPT clause.

GETBAR() returns the bar number of a position on a popup and can be used to determine where each option is positioned on the popup.

MULTISELECT

Multiple options can be chosen from a popup when MULTISELECT is included in DEFINE POPUP. When an option is chosen from a popup, the mark character is placed to the left of the option.

In FoxPro for MS-DOS, you can select multiple options from a popup with the keyboard by holding down the Shift key and selecting the options by pressing Enter or the Spacebar. To select consecutive options, select one option, then hold down the Shift key and press the Up or Down Arrow key.

To select consecutive options, hold down the Shift key and drag the cursor through the options with the mouse. To select nonconsecutive options with a mouse, hold down the Shift key and click each option.

In FoxPro for Windows, the KEYCOMP setting determines how you can select multiple options. If KEYCOMP is set to DOS, multiple options can be selected in the manner described for FoxPro for MS-DOS.

If KEYCOMP is set to WINDOWS, you can select consecutive options from a popup with the keyboard. To select consecutive options with the keyboard, select one option, then hold down the Shift key and press the Up or Down Arrow key. To select consecutive options with a mouse, click on the first option you want to select and drag the cursor to the last option you want to select. To select nonconsecutive options with a mouse, hold down the Ctrl key and click on each option.

You cannot make multiple selections from a popup created with a PROMPT clause.

MRKBAR() is used to determine which options are chosen from the popup. Choosing an option from a popup created with MULTISELECT marks the option. MRKBAR() returns a true (.T.) value if an option is marked.

If you include MULTISELECT in DEFINE POPUP, you may also want to include MARGIN to reserve space in each option for the mark character.

Example In the following example, a popup named **Fruits** is created. MULTISELECT is included to create a popup that allows multiple options to be chosen.

In FoxPro for MS-DOS, each of the four options has a different mark character. When options are chosen from the popup, the options are marked and a routine named YOURCHOICE displays the chosen options.

```
CLEAR
DEFINE POPUP fruits FROM 5,5 ;
    MULTISELECT MARGIN && Create multi-choice popup
DEFINE BAR 1 OF fruits ;
    PROMPT '\<Apples' MARK CHR(3) && First option
DEFINE BAR 2 OF fruits ;
    PROMPT '\<Bananas' MARK CHR(4)&& Second option
DEFINE BAR 3 OF fruits ;
    PROMPT '\<Grapes' MARK CHR(5) && Third option
DEFINE BAR 4 OF fruits ;
    PROMPT '\<Lemons' MARK CHR(6) && Fourth option

@ 12,5 SAY 'Your choices:'

ON SELECTION POPUP fruits DO yourchoice && Choice routine
ACTIVATE POPUP fruits

PROCEDURE yourchoice && Executed when choice is made
@ 13,5 CLEAR
FOR count = 1 TO CNTBAR('fruits') && Loop for number of options
    IF MRKBAR('fruits', count) = .T. && If option is marked,
        ? PRMBAR('fruits', count) AT 5 && display option prompt
    ENDIF
NEXT
```

PROMPT FIELD <expr>

If you include PROMPT FIELD and specify a field name from an open table/.DBF with <expr>, the records in that field become the options on the popup. The popup will contain an option for each record in the table/.DBF. When the popup is activated, the work area containing the table/.DBF is selected.



You can take advantage of Rushmore optimization if you set a filter on the field specified with PROMPT FIELD used in the popup. For further details on Rushmore optimization, see the *Optimizing Your Application* chapter in the *FoxPro Developer's Guide*.

<expr> can also contain multiple field names and expressions concatenated with the addition operator (+). <expr> can also be the name of the field in a table/.DBF open in another work area or a user-defined function (UDF).

The maximum number of options that may appear on a popup created with PROMPT FIELD in the standard version of FoxPro for MS-DOS is 32,767. If the table/.DBF has more records, the message "Popup too big, first <nnnnn> entries shown" is displayed. In the 32-bit Extended version of FoxPro for MS-DOS and in FoxPro for Windows, there is no limit to the number of entries that can appear in a popup created with PROMPT FIELD.

PROMPT FILES [LIKE <skel>]

You can easily create a popup that displays the names of files available on disk. If you include PROMPT FILES, a popup is created that displays the names of all files in the current directory.

An additional LIKE <skel> clause can be included to selectively place the names of files on the popup. The file specification skeleton <skel> supports wildcard characters. For example, to create a popup that displays the names of tables/.DBFs in the default drive and directory, include the following command:

```
PROMPT FILES LIKE *.DBF
```

You can create a popup that displays the names of files on other drives and in other directories by including a drive specification, a directory specification or both. For example, to create a popup that displays the names of program files in a directory called PROGRAMS on drive A, include this command:

```
PROMPT FILES LIKE A:\PROGRAMS\*.PRG
```

PROMPT STRUCTURE

PROMPT STRUCTURE displays the names of the fields in the current table/.DBF on the popup according to the table/.DBF's field structure. When the popup is activated, the work area containing the table/.DBF is selected.

RELATIVE

The order that options are inserted onto a popup can be controlled with **RELATIVE**. If a popup is created *without* the **RELATIVE** clause, an option is positioned on a popup in an order dictated by the option's bar number. Space on the popup is reserved for undefined options. For example, if the first and third options are defined and the popup is activated, a blank line reserved for the second option is placed on the popup.

If a popup is created *with* **RELATIVE**, the options appear on the popup in the order in which they are defined. Space in the popup isn't reserved for undefined options.

Defining a popup with **RELATIVE** also lets you make use of **BEFORE** and **AFTER** clauses in **DEFINE BAR** to position options on a popup relative to other options. If a popup is created without **RELATIVE**, including **BEFORE** or **AFTER** in **DEFINE BAR** generates a syntax error.

Examples Run the following two program examples and compare the placement of the options on each popup.

*** **RELATIVE** Example ***

```
DEFINE POPUP relatyес RELATIVE FROM 1,1
```

```
DEFINE BAR 4 OF relatyес PROMPT '4444'
DEFINE BAR 3 OF relatyес PROMPT '3333'
DEFINE BAR 2 OF relatyес PROMPT '2222'
DEFINE BAR 1 OF relatyес PROMPT '1111'
DEFINE BAR 6 OF relatyес PROMPT '6666' BEFORE 4
```

```
ACTIVATE POPUP relatyес
```

*** **NON-RELATIVE** Example ***

```
DEFINE POPUP notrelat FROM 1,1
```

```
DEFINE BAR 4 OF notrelat PROMPT '4444'
DEFINE BAR 3 OF notrelat PROMPT '3333'
DEFINE BAR 2 OF notrelat PROMPT '2222'
DEFINE BAR 1 OF notrelat PROMPT '1111'
DEFINE BAR 6 OF notrelat PROMPT '6666'
```

```
ACTIVATE POPUP notrelat
```

SCROLL

If SCROLL is included in DEFINE POPUP, a scroll bar is placed to the right of the popup you create. The scroll bar lets you move quickly through the popup with the mouse and provides a visual indication of your position on the popup. The scroll bar is displayed only when there are more options than can fit on the popup or if the popup is too long to fit on the desktop, the main FoxPro window or the active user-defined window.

SHADOW — FoxPro for MS-DOS only

In FoxPro for MS-DOS, a shadow can be placed behind a popup if SHADOW is included when the popup is defined. SET SHADOWS can be used to display or remove the popup's shadow. For further information, see SET SHADOWS in this chapter.

SHADOW is ignored in FoxPro for Windows.

TITLE <expC4>

You can specify a title that is displayed in the center of the top border of the popup. <expC4> is the title you want to display in the top border of the popup. If the title is wider than the popup, the title is truncated.

TITLE is ignored in FoxPro for Windows.

COLOR SCHEME <expN>**| COLOR <color pair list>**

You can specify the color of all elements of the popup by including the number of an existing color scheme in the COLOR SCHEME clause or a set of color pairs in the COLOR clause.

A color scheme is a set of 10 predefined color pairs. The color pairs in a color scheme can be changed with SET COLOR OF SCHEME. In FoxPro for MS-DOS the color pairs in a color scheme can also be changed in the Color Picker.

A color pair is a set of two letters separated by a forward slash. The first color letter specifies the foreground color and the second letter specifies the background color.

For example, this color pair specifies a red foreground on a white background:

R/W

For a list of colors and their corresponding color letters, see the SET COLOR commands section in this chapter or the Color Table by Color Pair topic in the online help facility.

In FoxPro for Windows, a color pair can also be specified with a set of 6 RGB (Red Green Blue) color values separated by commas. The first 3 color values specify the foreground color and the second 3 color values specify the background color. The color values can range from 0 through 255.

In FoxPro for Windows, the R/W color pair in the example above can also be specified with this RGB color pair:

```
RGB(255,0,0,255,255,255)
```

By default, the colors of popups created with DEFINE POPUP are controlled by color scheme 2.

Example

This example uses DEFINE POPUP to create popups that are activated when a menu pad in the menu bar is chosen.

The current system menu bar is first saved to memory with SET SYSMENU SAVE, and then all system menu pads are removed with SET SYSMENU TO.

Two new system menu pads are created with DEFINE PAD, and DEFINE POPUP creates a popup for each menu pad. DEFINE BAR creates options on each of the popups. When a menu pad is chosen, ON PAD uses ACTIVATE POPUP to activate the corresponding popup.

When an option is chosen from a popup, ON SELECTION POPUP uses PROMPT() and POPUP() to pass the option number and popup name to the CHOICE procedure. CHOICE displays the prompt of the chosen option and the name of the popup containing the option. If the **Exit** option is chosen from the **Card Info** popup, the original FoxPro system menu is restored.

```
*** Name this program DEFINPOP.PRG ***

CLEAR

SET SYSMENU SAVE
SET SYSMENU TO

DEFINE PAD convpad OF _MSYMENU PROMPT '\<Conversions' COLOR SCHEME 3 ;
    KEY ALT+C, ''
DEFINE PAD cardpad OF _MSYMENU PROMPT 'Card \<Info' COLOR SCHEME 3 ;
    KEY ALT+I, ''

ON PAD convpad OF _MSYMENU ACTIVATE POPUP conversion
ON PAD cardpad OF _MSYMENU ACTIVATE POPUP cardinfo
```

```

DEFINE POPUP conversion MARGIN RELATIVE SHADOW COLOR SCHEME 4
DEFINE BAR 1 OF conversion PROMPT 'Ar\<ea' ;
    KEY CTRL+E, '^E'
DEFINE BAR 2 OF conversion PROMPT '\<Length' ;
    KEY CTRL+L, '^L'
DEFINE BAR 3 OF conversion PROMPT 'Ma\<ss' ;
    KEY CTRL+S, '^S'
DEFINE BAR 4 OF conversion PROMPT 'Spee\<d' ;
    KEY CTRL+D, '^D'
DEFINE BAR 5 OF conversion PROMPT '\<Temperature' ;
    KEY CTRL+T, '^T'
DEFINE BAR 6 OF conversion PROMPT 'T\<ime' ;
    KEY CTRL+I, '^I'
DEFINE BAR 7 OF conversion PROMPT 'Volu\<me' ;
    KEY CTRL+M, '^M'

ON SELECTION POPUP conversion;
    DO choice IN definpop WITH PROMPT( ), POPUP( )

DEFINE POPUP cardinfo MARGIN RELATIVE SHADOW COLOR SCHEME 4
DEFINE BAR 1 OF cardinfo PROMPT '\<View Charges' ;
    KEY ALT+V, ''
DEFINE BAR 2 OF cardinfo PROMPT 'View \<Payments' ;
    KEY ALT+P, ''
DEFINE BAR 3 OF cardinfo PROMPT 'Vie\<w Users' ;
    KEY ALT+W, ''
DEFINE BAR 4 OF cardinfo PROMPT '\<- '
DEFINE BAR 5 OF cardinfo PROMPT '\<Charges '
DEFINE BAR 6 OF cardinfo PROMPT '\<- '
DEFINE BAR 7 OF cardinfo PROMPT 'E\<xit '

ON SELECTION POPUP cardinfo;
    DO choice IN definpop WITH PROMPT( ), POPUP( )

PROCEDURE choice
PARAMETERS mprompt, mpopup

WAIT WINDOW 'You chose ' + mprompt + '
    ' from popup ' + mpopup NOWAIT

IF mprompt = 'Exit'
    SET SYSMENU TO DEFAULT
ENDIF

```

DEFINE WINDOW

Purpose	Creates a window and specifies its attributes
Syntax	<pre> DEFINE WINDOW <window name1> FROM <row1, column1> TO <row2, column2> AT <row3, column3> SIZE <row4, column4> [IN [WINDOW] <window name2> IN SCREEN IN DESKTOP] [FONT <expC1> [, <expN1>]] [STYLE <expC2>] [FOOTER <expC3>] [TITLE <expC4>] [HALFHEIGHT] [DOUBLE PANEL NONE SYSTEM <border string>] [CLOSE NOCLOSE] [FLOAT NOFLOAT] [GROW NOGROW] [MDI NOMDI] [MINIMIZE] [SHADOW] [ZOOM NOZOOM] [ICON FILE <expC5>] [FILL <expC6> FILL FILE <bmp file>] [COLOR SCHEME <expN2>] COLOR <color pair list> </pre>
Remarks	AT, FILL FILE, FONT, HALFHEIGHT, ICON FILE, IN DESKTOP, MDI, NOMDI, SIZE, STYLE clauses and RGB color pairs — FoxPro for Windows only FOOTER and SHADOW — FoxPro for MS-DOS only
See Also	ACTIVATE WINDOW, DEACTIVATE WINDOW, HIDE WINDOW, MODIFY WINDOW, MOVE WINDOW, RELEASE WINDOWS, SET BORDER, SHOW WINDOW

Description DEFINE WINDOW creates a user-defined window and specifies its attributes. After user-defined windows are created, they can be displayed on the desktop in FoxPro for MS-DOS or the main FoxPro window in FoxPro for Windows with ACTIVATE WINDOW or SHOW WINDOW. The number of user-defined windows you can create is limited only by the amount of available memory and system resources.

Window definitions are placed in memory, and the user-defined window names are listed in the bottom portion of the **Window** menu popup. The name of the user-defined window created first appears at the top of the list, followed by the name of each additional user-defined window. The name of the user-defined window that output is currently directed to is marked with a diamond (♦) in FoxPro for MS-DOS and a check (✓) in FoxPro for Windows. If there is no output window, output is directed to the desktop or the main FoxPro window.

Activated windows remain on the desktop or main FoxPro window until `DEACTIVATE WINDOW` or `HIDE WINDOW` is issued. `DEACTIVATE WINDOW` and `HIDE WINDOW` remove windows from the desktop or main FoxPro window but don't remove the window definitions from memory. Windows can be placed back on the desktop or main FoxPro window with `ACTIVATE WINDOW` or `SHOW WINDOW`.

Use `CLEAR WINDOWS` or `RELEASE WINDOWS` to remove windows from the desktop or main FoxPro window and window definitions from memory. Windows whose definitions have been removed from memory must be recreated with `DEFINE WINDOW` to be redisplayed.

Clauses

<window name1>

Specify the name of the window to create with <window name1>. Window names can be up to 10 characters long. They must begin with a letter or underscore, and cannot begin with a number. They can contain any combination of letters, numbers and underscores.

FROM <row1, column1> TO <row2, column2>

The position of the upper-left corner of the window on the desktop or main FoxPro window is determined by the coordinates specified with <row1, column1>. The position of the lower-right corner is specified with <row2, column2>. These two sets of coordinates determine the window size. A window can be defined with coordinates that lie outside the desktop or main FoxPro window border and can be larger than the desktop or main FoxPro window. User-defined windows can also be placed inside other user-defined windows.

In FoxPro for Windows, the window's location and size are determined by the font of the window's parent. The window's parent can be another user-defined window or the main FoxPro window.

AT <row3, column3> — FoxPro for Windows only

SIZE <row4, column4> — FoxPro for Windows only

The AT and SIZE clauses are useful for creating a user-defined window in FoxPro for Windows. When a window is created with the AT and SIZE clauses, the size of the window is determined by the window's font.

You can specify a font and font style for a user-defined window by including the FONT and STYLE clauses. If you specify a font for the window and include the SIZE clause, its size is determined by the window's font height and width. If you don't specify a font for a window, the window's font is the default system font — 10 point FoxFont.

The location of the upper-left corner of the window you create is specified with AT <row3, column3>. This location is determined by the current font of the window's parent. Because the AT clause is identical to the FROM clause in all respects, the two clauses can be used interchangeably.

The SIZE clause can be used instead of the TO clause to ensure that text displayed in a specific font will fit in the window you create. The size of the user-defined window in rows and columns is specified with <row4, column4>.

IN [WINDOW] <window name2>

Include the IN clause to place a user-defined window in a parent window. The user-defined window (the child window) becomes integrated with the parent window it is placed in. A child window placed in a parent window cannot be moved outside the parent window. If the parent window is moved, the child window moves with it.

When a child window is placed in a parent window, the child window coordinates specified with the FROM and TO clauses or the AT and SIZE clauses are relative to the parent window, not the desktop or main FoxPro window.

Example In the following example, a parent window, TEST1, is created. A child window, TEST2, is placed in the parent window.

```
CLEAR

DEFINE WINDOW test1 ;
    FROM 1, 1 TO 20, 30 ;
    TITLE "Test1"                && Parent window.
ACTIVATE WINDOW test1

DEFINE WINDOW test2 ;
    FROM 1, 1 TO 20, 20 ;
    TITLE "Test2" ;
    IN WINDOW test1              && Child window.
ACTIVATE WINDOW test2

ACTIVATE SCREEN

WAIT WINDOW 'Press a key to clear the windows'

RELEASE WINDOW test1, test2
CLEAR
```

IN SCREEN

Include IN SCREEN to explicitly place a window on the desktop or the main FoxPro window. Windows are placed on the desktop or the main FoxPro window by default.

IN DESKTOP — FoxPro for Windows only

Include IN DESKTOP to place a user-defined window in the Microsoft Windows desktop, outside the main FoxPro window. The position of the window is relative to the Windows desktop, and size of the window is determined by the current font of the main FoxPro window.

FONT <expC1> [, <expN1>] — FoxPro for Windows only

Use the FONT clause to specify a font for output directed to the window. <expC1> is the name of the font and <expN1> is the font size. If <expN1> is omitted, a 9 point font is used.

For example, the command creates a window that displays output directed to the window in a 16 point Roman font:

```
DEFINE WINDOW romanfont FROM 2, 2 TO 12, 22 FONT 'ROMAN', 16
```

If the FONT clause is omitted, 10 point FoxFont is used. If the font you specify is not available, Windows substitutes a font with similar font characteristics.

STYLE <expC2> — FoxPro for Windows only

In FoxPro for Windows, include the STYLE clause to specify a font style for output directed to the window. The styles that are available for a font are determined by Windows. If the font style you specify is not available, Windows substitutes a font style with similar characteristics.

The font style is specified with <expC2>. If the STYLE clause is omitted, the standard font style is used.

Character	Font Style
B	Bold
I	Italic
N	Normal
O	Outline
Q	Opaque
S	Shadow
—	Strikeout
T	Transparent
U	Underline

You can include more than one character to specify a combination of font styles. For example:

```
DEFINE WINDOW bolditalic FROM 2, 2 TO 12, 22 STYLE 'BI'
```

FOOTER <expC3> — FoxPro for MS-DOS only

In FoxPro for MS-DOS, a window can be assigned a footer with the FOOTER clause. <expC3> is the text you want to display as the footer. The footer is centered in the bottom border of the window. If the footer is wider than the window, the footer is truncated.

In FoxPro for Windows, the FOOTER clause is ignored.

TITLE <expC4>

In FoxPro for MS-DOS, a window can be assigned a title with the TITLE clause. The title <expC4> is centered in the top border of the window. If the title is wider than the window, the title is truncated.

HALFHEIGHT — FoxPro for Windows only

Use HALFHEIGHT to create windows with half-height title bars, which provide compatibility for windows created in FoxPro version 2.0 and imported into FoxPro for Windows.

When you use DEFINE WINDOW to create a window in FoxPro for Windows, a half-height title bar is used unless you include the SYSTEM key word or you include a FONT clause.

If you include the HALFHEIGHT key word, a half-height title bar is used regardless of whether a SYSTEM or a FONT clause is included.

DOUBLE | PANEL | NONE | SYSTEM | <border string>

You can specify a border for a user-defined window by including the DOUBLE, PANEL, NONE, SYSTEM or <border string> clause. The default border is a single line.

The DOUBLE option places a double-line border around the window. The PANEL option places a wide border around the window. The NONE option suppresses the border entirely.

Include the SYSTEM clause to emulate the look of system windows. When you include certain other clauses (GROW, ZOOM, and so on), the appropriate window controls are placed in the top border of the window.

You can create your own custom border for a window with <border string>. For more information on defining a custom border, see SET BORDER in this chapter for the border string syntax.

In FoxPro for Windows, including DOUBLE or a custom border string creates a window with the PANEL border. Including the CLOSE, FLOAT, GROW, ZOOM, or MINIMIZE clauses places the appropriate controls on the window even if the SYSTEM window type definition clause is not included.

CLOSE | NOCLOSE

If the CLOSE clause is included, you can close a user-defined window from the **File** menu popup or with the keyboard or mouse. Closing a window removes it from the desktop, the main FoxPro window or a parent user-defined window and removes its definition from memory.

In FoxPro for MS-DOS, you can close a window created with CLOSE by choosing the **Close** option from the **File** menu.

If the window was created with the **SYSTEM** clause, click on the close box in the upper-left corner of the window's border to close the window. If the window wasn't created with the **SYSTEM** clause, a "hidden" close box is available in the upper-left corner of the window's border. Clicking on this corner closes the window.

In FoxPro for Windows, you can close a window created with **CLOSE** by pressing Ctrl+F4 or by choosing the **Close** option from the **File** menu or the window's **Control** menu.

For more information on closing windows, see the Interface Basics chapter in the FoxPro *User's Guide*.

Include **NOCLOSE** or omit **CLOSE** to prevent a window from being closed.

FLOAT | NOFLOAT

A window can be moved if the **FLOAT** option is included in **DEFINE WINDOW**.

To move a window in FoxPro for MS-DOS, choose **Move** from the **Window** menu or press Ctrl+F7. You can also click on the top border of the window and drag the window to a new location.

To move a window in FoxPro for Windows, choose **Move** from the window's **Control** menu or press Ctrl+F7. Press the arrow keys to move the window. You can also click on the top border of the window and drag the window to a new location.

For more information on moving windows, see the Interface Basics chapter in the FoxPro *User's Guide*.

Include **NOFLOAT** or omit **FLOAT** to prevent a user-defined window from being moved.

GROW | NOGROW

The size of a user-defined window can be changed if you include **GROW** in **DEFINE WINDOW**.

To change the size of a window in FoxPro for MS-DOS, choose **Size** from the **Window** menu or press Ctrl+F8. To change the size of a window with the mouse, click on the lower-right corner of the window and drag it to change the window's size.

To change the size of a window in FoxPro for Windows, press Ctrl+F8 and then press the arrow keys. If the window is created with the **SYSTEM** clause, you can choose **Size** from the window's **Control** menu and then press the arrow keys. To change the size of a window with the mouse, position the pointer over any side or corner of the window and drag.

For more information on sizing windows, see the Interface Basics chapter in the FoxPro *User's Guide*.

Include **NOGROW** or omit **GROW** to prevent a window's size from being changed.

MDI | NOMDI — FoxPro for Windows only

In FoxPro for Windows, you can create a user-defined window that is MDI (Multiple Document Interface)–compliant by including the **MDI** clause. MDI is a specification that allows multiple document windows and determines their structure and behavior.

Include **NOMDI** to create a window that is not MDI-compliant. If you don't include either clause, the window is not MDI-compliant.

MINIMIZE

A user-defined window can be minimized if you include **MINIMIZE** in **DEFINE WINDOW**.

To minimize a window in FoxPro for MS-DOS, double-click on the top border of the window, press Ctrl+F9 or choose **Zoom** ↓ from the **Window** menu. To return the window to its original size, repeat the same steps. You can minimize a window and “dock” it in the lower-right corner of the desktop by pressing the Shift key when you double-click on the top border of the window.

To minimize a window in FoxPro for Windows, choose **Minimize** from the window's **Control** menu, press Ctrl+F9 or click on the window's minimize button. To return the window to its original size, double-click on the window's icon; or click on the window's icon and then choose **Restore** from the **Control** menu or press Ctrl+F5.

SHADOW — FoxPro for MS-DOS only

A user-defined window created with **SHADOW** has a darkened area behind it resembling a shadow. Any text or objects covered by the shadow are still visible. By default, windows don't have shadows. You can change the default setting for shadows with **SET SHADOWS**.

The **SHADOW** clause is ignored in FoxPro for Windows.

ZOOM | NOZOOM

A user-defined window can be enlarged to fill the entire desktop or main FoxPro window if ZOOM is included. If a window is enlarged to full size, it can also be reduced to its original size.

To enlarge a window to fill the entire desktop in FoxPro for MS-DOS, choose **Zoom** ↑ from the **Window** menu or press Ctrl+F10. If the window was created with the SYSTEM clause, you can click on the Zoom control in the upper-right corner of the window's border.

If the window wasn't created with the SYSTEM option, a "hidden" Zoom control is available in the upper-right corner of the window's border. Clicking on this corner enlarges the window to fill the entire desktop.

Repeat these steps to return the window to its original size.

To enlarge a window to fill the entire main FoxPro window in FoxPro for Windows, choose **Maximize** from the window's **Control** menu, press Ctrl+F10, click on the window's Maximize button or double-click on the window's top border. To return the window to its original size, choose **Restore** from the **Control** menu, press Ctrl+F5 or click on the Restore button.

Windows defined with NOZOOM or without ZOOM cannot be enlarged.

See the Interface Basics chapter in the FoxPro *User's Guide* for more information about zooming windows.

ICON FILE <expC5> — FoxPro for Windows only

If you include the MINIMIZE key word in DEFINE WINDOW, you can also include ICON FILE <expC5> to specify the icon displayed when the window is minimized. You can specify an icon (.ICO) file only; you cannot specify a bitmap (.BMP) file.

FILL <expC6> — FoxPro for MS-DOS**| FILL FILE <bmp file> — FoxPro for Windows only**

In FoxPro for MS-DOS, the background of a window can be filled with a character by including FILL <expC6>. If <expC6> contains more than one character, only the first character is used as the fill character. You can also specify a fill character with CHR().

In FoxPro for Windows, the FILL clause is ignored.

FILL FILE <bmp file> is a FoxPro for Windows clause that specifies a wallpaper (background) for a user-defined window. The window is tiled with the specified bitmap (.BMP) file.

COLOR SCHEME <expN2>

| COLOR <color pair list>

The color of user-defined windows can be specified by including the number of an existing color scheme in the COLOR SCHEME clause or a set of color pairs in the COLOR clause.

A color scheme is a set of 10 predefined color pairs. The color pairs in a color scheme can be changed with SET COLOR OF SCHEME. In FoxPro for MS-DOS the color pairs in a color scheme can also be changed in the Color Picker.

A color pair is a set of two letters separated by a forward slash. The first color letter specifies the foreground color and the second letter specifies the background color.

For example, this color pair specifies a red foreground on a white background:

R/W

For a list of colors and their corresponding color letters, see the SET COLOR commands section in this chapter or the Color Table by Color Pair topic in the online help facility.

In FoxPro for Windows, a color pair can also be specified with a set of 6 RGB (Red Green Blue) color values separated by commas. The first 3 color values specify the foreground color and the second 3 color values specify the background color. The color values can range from 0 through 255.

In FoxPro for Windows, the R/W color pair in the example above can also be specified with this RGB color pair:

RGB(255,0,0,255,255,255)

By default, the colors of windows created with DEFINE WINDOW are controlled by color scheme 1.

Example In this example, a window named Output is created and activated. The program waits for you to press a key and then hides the window. The program waits for you to press a key again and then redisplay the window.

```
CLEAR DEFINE WINDOW output FROM 2,1 TO 13,75 TITLE 'Output' ;
      CLOSE FLOAT GROW SHADOW ZOOM
```

```
ACTIVATE WINDOW output
WAIT WINDOW 'press any key to hide window output'
HIDE WINDOW output
WAIT WINDOW 'press any key to show window output'
SHOW WINDOW output
WAIT WINDOW 'press any key to release window output'
RELEASE WINDOW output
```

DELETE

Purpose	Marks records for deletion
Syntax	DELETE [<scope> [FOR <expL1> [WHILE <expL2> [NOOPTIMIZE]
See Also	DELETED(), PACK, RECALL, SET DELETED

Description DELETE marks records for deletion in the current table/.DBF. Records marked for deletion aren't physically removed from the table/.DBF until PACK is issued. Records marked for deletion can be recalled (unmarked) with RECALL.

Records can also be marked for deletion from within a Browse, Change or Edit window or from the **Record** menu. See the *FoxPro User's Guide* for more information.

Clauses <scope>
The scope clauses are: ALL, NEXT <expN>, RECORD <expN>, and REST. These are explained in the Overview of the FoxPro Language chapter of this manual. Commands which include <scope> operate only on the table/.DBF in the active work area.

You can specify a scope of records to mark for deletion. Only the records that fall within the range of records specified by the scope are marked for deletion.

The default scope for DELETE is the current record (NEXT 1).

FOR <expL1>
Including the FOR clause lets you conditionally delete records. If the FOR clause is included, only the records that satisfy the logical condition <expL1> are marked for deletion.

Rushmore optimizes a query created with DELETE ... FOR if <expL1> is an optimizable expression and the table/.DBF is indexed on DELETED(). For best performance, use an optimizable expression in the FOR clause. A discussion of Rushmore optimizable expressions appears in the Optimizing Your Application chapter in the *FoxPro Developer's Guide*.

WHILE <expL2>

If the WHILE clause is included, records are marked for deletion for as long as <expL2> evaluates to true (.T.).

NOOPTIMIZE

To disable Rushmore optimization of DELETE, include NOOPTIMIZE. For further information on NOOPTIMIZE, see SET OPTIMIZE in this chapter and the discussion of Rushmore optimization in the Optimizing Your Application chapter in the *FoxPro Developer's Guide*.

Example

In this example, all customer records with Brookline in the CITY field are marked for deletion. The marked records are then unmarked with RECALL.

```
USE DATABASES
USE customer
DELETE FOR city = 'Brookline'
GOTO TOP
WAIT WINDOW 'First record marked deleted ... press Esc' NOWAIT
BROWSE FIELDS city
RECALL FOR city = 'Brookline'
GOTO TOP
WAIT WINDOW 'First record now unmarked ... press Esc' NOWAIT
BROWSE FIELDS city
USE
```

DELETE FILE

Purpose	Deletes a file from disk
Syntax	DELETE FILE [<file> ?]
See Also	ERASE, FILER, SET SAFETY

Description Use DELETE FILE to erase a file from disk.



Caution! Any file deleted with this command cannot be retrieved. Even if SET SAFETY is ON, you are not warned before the file is deleted.

The file you want to delete cannot be open when DELETE FILE is issued. If you delete a table/.DBF that has an associated .FPT memo file, be sure to delete the memo file. The file name must include a path if it is on a different drive or in a different directory than the default, and the file name extension must be included. The file name cannot contain MS-DOS wildcard characters.

Example In this example, the structure of CUSTOMER.DBF and all records in which the state is Ohio are copied to a table named BACKUP. The data in BACKUP is then copied to a text file, TEMP, which is opened and then deleted when the user closes it.

```
CLOSE DATABASES
USE customer
COPY STRUCTURE TO backup
USE backup
APPEND FROM customer FOR state = 'OH'
COPY TO temp TYPE DELIMITED
WAIT WINDOW 'Press Esc to close and erase temp.txt' NOWAIT
MODIFY FILE temp.txt
DELETE FILE temp.txt
?IIF(FILE('temp.txt'),'file not deleted','file deleted')
USE
DELETE FILE backup.dbf
```

DELETE TAG

Purpose	Removes a tag or tags from a .CDX compound index file
Syntax	DELETE TAG <tag name1> [OF <cdx file1>] [, <tag name2> [OF <cdx file2>]] ... or DELETE TAG ALL [OF <cdx file>]
See Also	COPY INDEX, COPY TAG, INDEX, TAG()

Description Compound index files, created with INDEX, contain tags corresponding to index entries. DELETE TAG is used to remove a tag or tags from open compound index files. You can delete only tags from compound index files open in the current work area. If you remove all the tags from a compound index file, the file is deleted from the disk.

FoxPro looks first for a tag in the structural compound index file (if one is open). If the tag isn't in the structural compound index file, FoxPro then looks for the tag in the other open compound index files.

Clauses **<tag name1> OF <cdx file1> [,<tag name2> [OF <cdx file2>]] ...**

To remove a tag in a compound index file, specify its name with <tag name1>. You can delete multiple tags with one DELETE TAG by including a list of tag names separated by commas.

If two or more tags with the same name exist in the open index files, you can remove a tag from a specific index file by including OF <cdx file>.

ALL [OF <cdx file>]

You can remove every tag from a compound index file by including the ALL key word. If the current table/.DBF has a structural compound index file, all tags are removed from the index file, the index file is deleted from the disk, and the flag in the table/.DBF's header indicating the presence of an associated structural compound index file is removed. Use ALL with OF <cdx file> to remove all tags from an open compound index file other than the structural compound index file.

DELETED()

Purpose	Returns true (.T.) if the current record is marked for deletion
Syntax	DELETED([<expC> <expN>])
Parameters	<expC> A table/.DBF alias <expN> A work area number
Returns	Logical
See Also	DELETE, PACK, RECALL, SET DELETED

Description DELETED() returns a logical value that indicates whether the current record is marked for deletion. If the record is marked for deletion, a true value (.T.) is returned; otherwise, a false value (.F.) is returned.

Records can be marked for deletion with DELETE, and they can be unmarked with RECALL. Records can also be deleted or recalled from within the Browse, Change and Edit windows.

Queries that test the deleted status of records can be optimized by Rushmore if the table/.DBF is indexed on DELETED(). See the *Optimizing Your Application* chapter in the *FoxPro Developer's Guide* for more information on using Rushmore to optimize queries.

Parameter <expC> | <expN>

DELETED supports optional arguments that let you return the deleted status of the current record in a non-current work area.

You can check the status of the current record in a table/.DBF open in another work area by specifying the work area number with <expN> or the table/.DBF alias with <expC>. If a table/.DBF isn't open in the work area you specify, DELETED() returns false.

If you omit <expC> and <expN>, the deleted status is returned for the current record in the current work area.

Example This example marks all but the first two records in INVOICES for deletion and uses DELETED() to scan the table/.DBF for all records that aren't marked for deletion and then adds their total inventory value to the value of the I_COST variable.

```

CLOSE DATABASES SET TALK OFF
CLEAR
USE invoices
GOTO 3
DELETE REST
i_cost = 0
?'INO  ITOTAL'
?
SCAN FOR NOT DELETED( )
    ? ino, itotal
    i_cost = i_cost + itotal
ENDSCAN
?' _____'
? i_cost, ' Total cost of current inventory'
GOTO TOP
RECALL ALL

```

DIFFERENCE()

Purpose	Returns a number representing the relative orthographic difference between two character expressions
Syntax	DIFFERENCE(<expC1>, <expC2>)
Parameters	<expC1> A character expression <expC2> A character expression
Returns	Numeric
See Also	SOUNDEX()

Description This function returns an integer from 0 through 4 that represents the relative orthographic difference between two character expressions. DIFFERENCE() is useful for searching tables/.DBFs when the exact spelling of an entry isn't known.

The more similarly the two expressions are spelled, the higher the number returned by DIFFERENCE(). If the character expressions are spelled very similarly, DIFFERENCE() returns 4. For two character expressions with little in common orthographically, DIFFERENCE() returns 0.

Parameters <expC1>, <expC2>

The character expressions that are compared by DIFFERENCE() are specified with <expC1> and <expC2>.

Example

```
STORE 'Smith' TO name1
STORE 'Smythe' TO name2
STORE 'Smittie' TO name3
STORE '' TO name4
? DIFFERENCE(name1, name2)
4
? DIFFERENCE(name1, name3)
4
? DIFFERENCE(name1, name4)
1
```


DIMENSION

Purpose	Creates one or more arrays
Syntax	DIMENSION <array1> (<expN1> [, <expN2>]) [, <array2> (<expN3> [, <expN4>])] ...
See Also	ACOPY(), ADEL(), ADIR(), AELEMENT(), AFIELDS(), AFONT(), AINS(), ALEN(), APPEND FROM ARRAY, ASCAN(), ASORT(), ASUBSCRIPT(), COPY TO ARRAY, DECLARE, EXTERNAL, GATHER, PRIVATE, PUBLIC, SCATTER, SET COMPATIBLE, STORE

Description Use DIMENSION to create one- or two-dimensional arrays of memory variables. DIMENSION is identical in operation and syntax to DECLARE.

Clauses **<array1>**
The name of the array you create is specified with <array1>. Multiple arrays can be created with a single DIMENSION by including additional array names (<array2>, <array3> and so on).

<expN1> [, <expN2>]

After you specify the name of the array to create, you must specify the size of the array with <expN1> and <expN2>.

If you include just <expN1>, a one-dimensional array is created. One-dimensional arrays have one column and <expN1> rows. For example, the following command creates a one-dimensional array named ARRAYONE that contains one column and ten rows.

```
DIMENSION arrayone(10)
```

To create a two-dimensional array, include both <expN1> and <expN2>. <expN1> specifies the number of rows in the array, and <expN2> specifies the number of columns. The following example creates a two-dimensional array named ARRAYTWO containing two rows and four columns:

```
DIMENSION arraytwo(2,4)
```

You must specify a size for each array you create with DIMENSION. In this example, three arrays are created: ARRAYONE and ARRAYTWO from the previous examples and a third array called ARRAYTHREE:

```
DIMENSION arrayone(10), arraytwo(2,4), arraythree(3,3)
```

You can use either square brackets or parentheses to enclose the expressions in DIMENSION or DECLARE. For example, the following two commands create identical arrays:

```
DIMENSION arrayone(10), arraytwo[2,4], arraythree(3,3)
DIMENSION arrayone[10], arraytwo(2,4), arraythree[3,3]
```

Array Elements

The size of an array determines how many elements it can contain. Each element in an array can store a single piece of information. To determine how many elements an array contains and how much information it can store, multiply the number of rows (<expN1>) in the array by the number of array columns (<expN2>).

Array elements can contain any type of data and are initialized to false (.F.) when the array is first created. You can initialize all the elements in an array to the same value with STORE if SET COMPATIBLE is FOXPLUS or OFF (the default setting). For example:

```
DIMENSION y(10,3)
STORE 'initial' TO y
```

Subscripts

Elements in an array are referenced by their subscripts. Each array element has a unique numeric subscript that identifies it. If the array is one-dimensional, an element's subscript is the same as its row number. For example, the subscript for the element in the third row of a one-dimensional array is 3. Elements in two-dimensional arrays are referenced by two subscripts: The first subscript indicates the row location of the element, and the second subscript indicates the column location. For example, the subscripts for the element in the third row and fourth column of a two-dimensional array are 3,4. For a further discussion of array element subscripts, see ASUBSCRIPT() in this chapter.

The subscript or subscripts for the first element in an array always start with 1. If an array is two-dimensional, it can also be referenced by a single subscript. Use AELEMENT() to return the single subscript from a pair of array row and column subscripts. Use ASUBSCRIPT() to return the row and column subscripts from a single subscript.

Redimensioning Arrays

You can change the size and dimensions of an array by issuing DIMENSION again. The size of an array can be increased or decreased, one-dimensional arrays can be converted to two dimensions, and two-dimensional arrays can be reduced to one dimension.

If the number of elements in an array is increased, the contents of all the elements in the original array are copied to the newly redimensioned array. The additional array elements are initialized to false (.F.).

Example

The following example demonstrates the result of increasing the size of a one-dimensional array. (Note that if you type these commands in the Command window the array will be PUBLIC, but it will be PRIVATE if you copy them into a program and run it.)

```
DIMENSION marray(2)
STORE 'A' TO marray(1)
STORE 'B' TO marray(2)
DISPLAY MEMORY LIKE marray
```

*** The following is displayed ***

```
MARRAY      Pub    A
              (  1)   C  "A"
              (  2)   C  "B"
```

```
DIMENSION marray(4)
DISPLAY MEMORY LIKE marray
```

*** The redimensioned array is displayed ***

```
MARRAY      Pub    A
              (  1)   C  "A"
              (  2)   C  "B"
              (  3)   L  .F.
              (  4)   L  .F.
```

If the number of elements in an array is decreased, the elements and any data they contain are deleted.

When a one-dimensional array is redimensioned to two dimensions, the contents of the original one-dimensional array are copied to the new array in an element-to-row order.

Example

In the following example, a one-dimensional array is converted to a two-dimensional array. The contents of the elements of the one-dimensional array are copied to the first row of the new array, followed by the second row and so on. The additional elements are initialized to false (.F.).

```
DIMENSION marrayone(4)
STORE 'E' TO marrayone(1)
STORE 'F' TO marrayone(2)
STORE 'G' TO marrayone(3)
STORE 'H' TO marrayone(4)
DISPLAY MEMORY LIKE marrayone
```

MARRAYONE	Pub	A
(1)	C	"E"
(2)	C	"F"
(3)	C	"G"
(4)	C	"H"

```
DIMENSION marrayone(2,3)
DISPLAY MEMORY LIKE marrayone
```

MARRAYONE	Pub	A
(1, 1)	C	"E"
(1, 2)	C	"F"
(1, 3)	C	"G"
(2, 1)	C	"H"
(2, 2)	L	.F.
(2, 3)	L	.F.

When a two-dimensional array is converted to one dimension, the contents of the original two-dimensional array are copied to the new array in a row-to-element order: the first element in the first row becomes the first element in the one-dimensional array, the second element in the first row becomes the second element, and so on.

Use ADEL() or AINS() to delete or insert array elements, rows and columns. Use APPEND FROM ARRAY, COPY TO ARRAY, SCATTER and GATHER to transfer data between table/.DBF records and arrays.

Example

In this example, a two-dimensional array is created and loaded with data. The array elements and the data they contain are displayed.

```
DIMENSION sample(2,3)
STORE 'Goodbye' TO sample(1,2)
STORE 'Hello' TO sample(2,2)
STORE 99 TO sample(6)
STORE .T. TO sample(1)
DISPLAY MEMORY LIKE sample
```

DIR or DIRECTORY

Purpose	Displays the contents of a disk directory
Syntax	DIR DIRECTORY [ON <drive>] [[LIKE] [<path>] [<skel>]] [TO PRINTER [PROMPT] TO FILE <file>]
Remarks	PROMPT clause — FoxPro for Windows only
See Also	DISPLAY FILES, FILER

Description Use DIR to display information about files. DIR without the LIKE clause displays:

- The names of all tables/.DBFs in the directory.
- The number of records in each table/.DBF.
- The date each table/.DBF was last updated.
- The size of each table/.DBF in bytes (tables/.DBFs in the original FoxBASE format are noted as such).
- The number of tables/.DBFs displayed.
- The total number of bytes remaining on the disk.

The table/.DBF information for the current drive and directory is displayed unless otherwise specified with <drive> and/or <path>.

Clauses **ON <drive>**

<drive> is the name of the drive on which the directory is located.

[LIKE] [<path>] [<skel>]

<path> is the MS-DOS path of the directory. The path can include the drive name if ON <drive> is omitted.

Include <skel> to display information about file types other than tables/.DBFs. <skel> is a file specification skeleton that supports wildcard characters. For example, to list all program files in the current directory, use the following command:

```
DIR *.PRG
```

TO PRINTER [PROMPT] | TO FILE <file>

Include TO PRINTER to send the directory information to the printer.

In FoxPro for Windows, you can include PROMPT to display a dialog before printing starts. Place the PROMPT key word immediately after TO PRINTER. PROMPT is ignored in FoxPro for MS-DOS.

In this dialog, you can adjust printer settings. You can specify the number of copies and page numbers to print and direct print output to a file. The Windows Printer Setup dialog can be opened to adjust additional printer settings. The printer settings you can adjust depend on the currently installed printer driver.

Include TO FILE <file> to send the information to the file specified with <file>.

Examples

```
DIR          && Display tables/.DBFS in the current directory
DIR *.CDX    && Display index files in the current directory
DIR V*.DBF   && Display tables/.DBFS that begin with V
```

DISKSPACE()

Purpose	Returns the number of bytes available on the default disk drive
Syntax	DISKSPACE()
Returns	Numeric
See Also	HEADER(), RECSIZE()

Description This function is useful for determining whether sufficient space is available to back up files or to execute commands such as SORT that require additional disk space for temporary work files.

On some networks, the value returned by DISKSPACE() may not be accurate for large network drives.

Example This example uses DISKSPACE() to determine whether sufficient disk space is available to perform a sort.

```
*** Check DISKSPACE before sort ***

CLOSE DATABASES
USE customer

*** Get size of table/.DBF header ***

dbf_top = HEADER( )

*** Calculate size of table/.DBF ***

file_size = dbf_top + (RECSIZE( ) * RECCOUNT( ) + 1)

IF DISKSPACE( ) > (file_size * 3)
    @ 5,0 SAY 'Sufficient diskspace to Sort.'
ELSE
    @ 5,0 SAY 'Insufficient diskspace. Sort cannot be done.'
ENDIF
```

DISPLAY

Purpose	Displays the contents of records in the current table/.DBF and the results of expressions
Syntax	DISPLAY [[FIELDS] <field list>] [<scope>] [FOR <expL1>] [WHILE <expL2>] [OFF] [TO PRINTER [PROMPT] TO FILE <file>] [NOCONSOLE] [NOOPTIMIZE]
Remarks	PROMPT clause — FoxPro for Windows only
See Also	LIST, SET HEADING, SET MEMOWIDTH

Description DISPLAY displays the contents of the current table/.DBF. DISPLAY fills the entire desktop in FoxPro for MS-DOS, main FoxPro window in FoxPro for Windows or user-defined window with information and then pauses. Press any key or click anywhere to see the next set of information. DISPLAY is similar to LIST, except that LIST displays the same information in a continuous stream without pausing.

DISPLAY can also be used to display the results of expressions, which can consist of literals, memory variables and arrays, fields, and memo fields. Field names and expressions are displayed unless SET HEADING is OFF.

Clauses **FIELDS <field list>**

All fields from the table/.DBF are displayed unless you include FIELDS <field list>. Then, only the fields included in the field list are displayed.

Memo field contents aren't displayed unless the memo field name is explicitly included in the field list. The displayed width of memo fields is determined by SET MEMOWIDTH.

<scope>

The scope clauses are: ALL, NEXT <expN>, RECORD <expN>, and REST. These are explained in the Overview of the FoxPro Language chapter of this manual. Commands which include <scope> operate only on the table/.DBF in the active work area.

You can specify a scope of records to display. Only the records that fall within the range of records specified by the scope are displayed.

The default scope for DISPLAY is the current record (NEXT 1).

FOR <expL1>

Including the FOR clause lets you conditionally display records, filtering out undesired records. If the FOR clause is included, only the records that satisfy the logical condition <expL1> are displayed.

Rushmore optimizes a query created with DISPLAY ... FOR if <expL1> is an optimizable expression. For best performance, use an optimizable expression in the FOR clause. A discussion of Rushmore optimizable expressions appears in the Optimizing Your Application chapter in the FoxPro *Developer's Guide*.

WHILE <expL2>

If the WHILE clause is included, records are displayed for as long as the logical expression <expL2> evaluates to true (.T.).

OFF

Include OFF to suppress record number display. If OFF is omitted, the record number is displayed before each record.

TO PRINTER [PROMPT] | TO FILE <file>

Include TO PRINTER to direct output from DISPLAY to a printer.

In FoxPro for Windows, you can include the optional PROMPT clause to display a dialog before printing starts. Place the PROMPT key word immediately after TO PRINTER. PROMPT is ignored in FoxPro for MS-DOS.

In this dialog you can adjust printer settings. You can specify the number of copies and page numbers to print and direct print output to a file. The Windows Printer Setup dialog can be opened to adjust additional printer settings. The printer settings you can adjust depend on the currently installed printer driver.

Include `TO FILE <file>` to direct output of the display to the disk file specified with `<file>`.

In FoxPro for MS-DOS, the output from `DISPLAY` directed to a file or a printer uses the settings from the printer driver setup if a printer driver setup is loaded. Use a printer driver to print output from `DISPLAY` in different font styles and sizes.

NOCONSOLE

Include `NOCONSOLE` to suppress output to the desktop, main FoxPro window or the active user-defined window.

NOOPTIMIZE

To disable Rushmore optimization of `DISPLAY`, include `NOOPTIMIZE`. For further information on `NOOPTIMIZE`, see `SET OPTIMIZE` in this chapter and the discussion of Rushmore optimization in the *Optimizing Your Application* chapter in the FoxPro *Developer's Guide*.

DISPLAY FILES

Purpose	Displays information about specified files
Syntax	DISPLAY FILES [ON <drive>] [LIKE <skel>] [TO PRINTER [PROMPT] TO FILE <file>]
Remarks	PROMPT clause — FoxPro for Windows only
See Also	DIR or DIRECTORY, FILER, LIST

Description Use DISPLAY FILES to display information about files residing on a disk. You can display information about all files on a specified disk drive or directory or only files that match a skeleton pattern containing wildcard characters.

Issuing DISPLAY FILES without any arguments displays information about tables/.DBFs in the current directory. The information displayed includes:

- The table/.DBF name.
- The number of records in the table/.DBF.
- The date and time of the last update to the table/.DBF.

Clauses **ON <drive>**

<drive> is the drive on which the files reside.

LIKE <skel>

If you include LIKE <skel>, FoxPro displays information only about files that match the skeleton pattern <skel>. The skeleton pattern can contain wildcard characters.

TO PRINTER [PROMPT] | TO FILE <file>

Include TO PRINTER to direct output from DISPLAY FILES to a printer.

In FoxPro for Windows, you can include PROMPT to display a dialog before printing starts. Place the PROMPT key word immediately after TO PRINTER. PROMPT is ignored in FoxPro for MS-DOS.

In this dialog, you can adjust printer settings. You can specify the number of copies and page numbers to print and direct print output to a file. The Windows Printer Setup dialog can be opened to adjust additional printer settings. The printer settings you can adjust depend on the currently installed printer driver.

Include TO FILE <file> to direct output from DISPLAY FILES to the disk file specified with <file>.

In FoxPro for MS-DOS, output from DISPLAY FILES directed to a file or a printer uses the settings from the printer driver setup if a printer driver setup is loaded. Use a printer driver to print output from DISPLAY FILES in different font styles and sizes.

DISPLAY MEMORY

Purpose	Displays the current contents of memory variables and arrays
Syntax	DISPLAY MEMORY [LIKE <skel>] [TO PRINTER [PROMPT] TO FILE <file>] [NOCONSOLE]
Remarks	PROMPT clause — FoxPro for Windows only
See Also	DECLARE, DIMENSION, LIST, STORE

Description DISPLAY MEMORY shows the name, type, contents and status of all currently defined memory variables and memory variable arrays. It also displays the number of memory variables defined, the number of bytes used and the number of additional memory variables available.

Information about system memory variables, menu bars and pads, popups and windows is also displayed.

The maximum number of memory variables you can create are listed in the capacities table of the FoxPro *Developer's Guide*. These limits can be increased in your FoxPro configuration file. For more information on configuring FoxPro, refer to the FoxPro *Installation and Configuration* manual.

Clauses **LIKE <skel>**

You can selectively display information about memory variables and arrays by including the LIKE clause. If you include LIKE<skel>, FoxPro displays the contents only of memory variables and arrays the match <skel>. The file skeleton <skel> supports wildcard characters. For example, to display all memory variables that begin with the letter A, use:

```
DISPLAY MEMORY LIKE A*
```

TO PRINTER [PROMPT] | TO FILE <file>

Include TO PRINTER to direct output from DISPLAY MEMORY to a printer.

In FoxPro for Windows, you can include PROMPT to display a dialog before printing starts. Place the PROMPT key word immediately after TO PRINTER. PROMPT is ignored in FoxPro for MS-DOS.

In this dialog you can adjust printer settings. You can specify the number of copies and page numbers to print and direct print output to a file. The Windows Printer Setup dialog can be opened to adjust additional printer settings. The printer settings you can adjust depend on the currently installed printer driver.

Include TO FILE <file> to direct output to the disk file specified with <file>.

In FoxPro for MS-DOS, output from DISPLAY MEMORY directed to a file or a printer uses the settings from the printer driver setup if a printer driver setup is loaded. Use a printer driver to print output from DISPLAY MEMORY in different font styles and sizes.

NOCONSOLE

Include NOCONSOLE to suppress output to the desktop, the main FoxPro window or the active user-defined window.

Example

In this example, several memory variables are created and assigned values. DISPLAY MEMORY first displays all memory variables that begin with "sam" and then displays all memory variables that contain five letters and end with "exit."

```
STORE 'Goodbye' TO sample1
STORE 'Hello' TO sample2
STORE .T. TO textit
STORE .F. TO mexit
DISPLAY MEMORY LIKE sam*
DISPLAY MEMORY LIKE ?exit
```

DISPLAY STATUS

Purpose Displays the status of the FoxPro environment

Syntax **DISPLAY STATUS**
[TO PRINTER [PROMPT] | TO FILE <file>]
[NOCONSOLE]

Remarks PROMPT clause — FoxPro for Windows only

See Also LIST

Description This form of DISPLAY lists information about the current FoxPro environment. This information includes the following:

- Table/.DBF and index file information:
 - Active tables/.DBFs
 - Active memo files
 - Table/.DBF aliases
 - Table/.DBF relations
 - Active indexes
 - Index file keys
 - The controlling index file
 - Structural compound files
 - Compound index tags
 - The shared attribute status of each open table/.DBF
 - The currently locked records in each table/.DBF
 - The EXCLUSIVE use setting
 - The LOCK setting
 - The MULTLOCKS setting
 - The SET REFRESH value
 - The SET REPROCESS value
- Open low-level file information:
 - Open low-level files
 - The file handle number for each low-level file
 - The file pointer position for each low-level file
 - Read/write attributes for each low-level file

- Additional FoxPro environment information:
 - The procedure file in use
 - The processor type
 - The file search path
 - The default disk drive
 - The print destination
 - The margin setting
 - The current work area
 - SET command switch settings
 - Binary modules currently loaded
 - DDE information in FoxPro for Windows

Clauses TO PRINTER [PROMPT] | TO FILE <file>

Include TO PRINTER to direct output from DISPLAY STATUS to a printer.

In FoxPro for Windows, you can include PROMPT to display a dialog before printing starts. Place the PROMPT key word immediately after TO PRINTER. PROMPT is ignored in FoxPro for MS-DOS.

In this dialog you can adjust printer settings. You can specify the number of copies and page numbers to print and direct print output to a file. The Windows Printer Setup dialog can be opened to adjust additional printer settings. The printer settings you can adjust depend on the currently installed printer driver.

Include TO FILE <file> to direct output to the disk file specified with <file>.

In FoxPro for MS-DOS, the output from DISPLAY STATUS directed to a file or a printer uses the settings from the printer driver setup if a printer driver setup is loaded. Use a printer driver to print output from DISPLAY STATUS in different font styles and sizes.

NOCONSOLE

Include NOCONSOLE to suppress output to the desktop, the main FoxPro window or the active user-defined window.

DISPLAY STRUCTURE

Purpose	Displays the structure of a table/.DBF file
Syntax	DISPLAY STRUCTURE [IN <expN> <expC>] [TO PRINTER [PROMPT] TO FILE <file>] [NOCONSOLE]
Remarks	PROMPT clause — FoxPro for Windows only
See Also	LIST

Description Use DISPLAY STRUCTURE to display the field structure of a table/.DBF. The name of each field in the table/.DBF is displayed with its type and width. If a field is numeric, the number of decimal places in the field is displayed.

DISPLAY STRUCTURE also displays the current number of records in the table/.DBF and the date it was last updated. If the table/.DBF has an associated memo field, the memo field blocksize is shown. The total width of all fields is also displayed.

The table/.DBF can have a structural compound index that is opened with the table/.DBF. If a tag in the structural compound index has the same name as a field in the table/.DBF and is indexed on that field, the order of the tag (ascending or descending) is displayed next to the field name.

If SET FIELDS is issued to limit access to fields in the table/.DBF, an angle bracket (>) appears beside the names of fields that can be accessed.

Clauses IN <expN> | <expC>

DISPLAY STRUCTURE displays the structure of the table/.DBF in the current work area. You can display the structure of a table/.DBF open in a different work area by specifying the work area number with <expN> or table/.DBF alias with <expC>.

TO PRINTER [PROMPT] | TO FILE <file>

Include TO PRINTER to direct the output from DISPLAY STRUCTURE to a printer.

In FoxPro for Windows, you can include PROMPT to display a dialog before printing starts. Place the PROMPT key word immediately after TO PRINTER. PROMPT is ignored in FoxPro for MS-DOS.

In this dialog, you can adjust printer settings. You can specify the number of copies and page numbers to print and direct print output to a file. The Windows Printer Setup dialog can be opened to adjust additional printer settings. The printer settings you can adjust depend on the currently installed printer driver.

Include TO FILE <file> to direct the output to the disk file specified with <file>.

In FoxPro for MS-DOS, the output from DISPLAY STRUCTURE directed to a file or a printer uses the settings from the printer driver setup if a printer driver setup is loaded. Use a printer driver to print output from DISPLAY STRUCTURE in different font styles and sizes.

NOCONSOLE

Include NOCONSOLE to suppress output to the desktop, the main FoxPro window or the active user-defined window.

DMY()

Purpose	Returns a date expression in Day Month Year format
Syntax	DMY(<expD>)
Returns	Character
See Also	MDY(), SET CENTURY, SET DATE

Description DMY() returns a character string in a Day Month Year format (for example, 31 May 1965) from a date expression. The month name isn't abbreviated.

If SET CENTURY is OFF, DMY() returns a character string in a DD Month YY format (for example, 31 May 65). If SET CENTURY is ON, the format is DD Month YYYY (for example, 31 May 1965).

Clause **<expD>**
<expD> is the date expression you want to return in Day Month Year format.

Examples ? DMY(DATE())

SET CENTURY ON
? DMY(DATE())

DO

Purpose	Executes a command or procedure file
Syntax	DO <file> [WITH <parm list>] [IN <file>]
See Also	CLEAR, PARAMETERS, PARAMETERS(), PRIVATE, PROCEDURE, PUBLIC, SET DEVELOPMENT, SET PATH, SET PROCEDURE

Description DO executes a FoxPro program. A program file can itself contain additional DO commands. This nesting of DO commands is limited to a depth of 32.

When you use DO to run a program, the commands contained in the program file are executed until one of the following occurs:

- RETURN is encountered.
- CANCEL is executed.
- Another DO is issued.
- The end of the file is reached.
- QUIT is executed.

When program execution is complete, control is returned to the calling program, the Command window, MS-DOS in FoxPro for MS-DOS, or Microsoft Windows in FoxPro for Windows.

If you choose **Do...** from the **Program** menu and execute a program in a directory or on a drive other than the current directory or drive, FoxPro changes the default directory and drive to the directory and drive containing the program.

Clauses <file>

<file> is the command or the name of the file to execute.

If you don't include an extension with the program you execute, FoxPro looks for and executes these versions of the program in the following order:

- .EXE – The executable version
- .APP – An application
- .FXP – The compiled version
- .PRG – The program

Multiple procedures within an executable version (.EXE) or an application (.APP) can have the same name. When you issue DO in a procedure within an executable version or an application, only the main program of the executable version or application is searched for the specified procedure.

To use DO to execute a specific menu program, screen program or query, you must include its extension (.MPR, .SPR or .QPR).

WITH <parm list>

Use the WITH clause to pass parameters to a program. The parameters listed in <parm list> can be expressions, memory variables, literals, fields or user-defined functions (UDFs). By default, parameters are passed to programs by reference. You can pass a parameter by value by placing it in parentheses. See SET UDFPARMS for a discussion of passing parameters by value or reference. See PARAMETERS for additional information on parameter passing. The maximum number of parameters you can pass to a program is 24.

IN <file>

Include the IN clause to execute a procedure in the program file specified with <file>.

When the file is located, the procedure is executed. If the program file cannot be located, the message "File does not exist" is displayed. If the program file is located but the specified procedure isn't in the program file, the message "Procedure not found" is displayed.

Example

In the following example, CUSTOMER.DBF is opened and the first record is displayed. Choose a button to move forward or backward through the table/.DBF or to the top or bottom of the table/.DBF. When you make a choice, FoxPro moves the record pointer by using DO to execute the appropriate procedure. If you choose **Quit**, the program is exited.

```
CLOSE DATABASES
SET TALK OFF
DEFINE WINDOW customer FROM 3,3 TO 13,57 ;
    FLOAT SHADOW DOUBLE
DEFINE WINDOW panel2 FROM 2, 61 TO 14,74 ;
    FLOAT SHADOW DOUBLE COLOR SCHEME 5
USE customer
```

```

ACTIVATE WINDOW customer NOSHOW
@ 3,3 SAY 'Company'
@ 3,14 GET customer.company SIZE 1,35
@ 5,3 SAY 'Contact'
@ 5,14 GET customer.contact SIZE 1,35
@ 7,3 SAY 'City/State'
@ 7,14 GET customer.city SIZE 1,21
@ 7,36 GET customer.state
@ 7,39 GET customer.zip
@ 0,3 SAY 'Customer #'
@ 0,15 SAY cno SIZE 1,6
@ 0,36 SAY 'Record #'
@ 0,46 SAY RECNO( ) SIZE 1,3
ACTIVATE WINDOW panel2
@ 1,2 GET act PICTURE;
    '@*VN Top;Prior;Next;Bottom;\?Quit' ;
    SIZE 1,8,1 DEFAULT 0 VALID actvalid( )
READ CYCLE SHOW readshow( ) COLOR ,R/BG
RELEASE WINDOWS customer, panel2

```

```

FUNCTION actvalid

```

```

DO CASE
CASE act = 1
    GO TOP
CASE ACT = 2
    DO go_prev
CASE ACT = 3
    DO go_next
CASE act = 4
    GO BOTTOM
CASE act = 5
    CLEAR READ
ENDCASE
SHOW GETS
RETURN 0

```

```

FUNCTION readshow
STORE WOUTPUT( ) TO currwind
IF SYS(2016) = 'customer' OR SYS(2016) = ''
    ACTIVATE WINDOW customer SAME
    @ 0,15 SAY cno SIZE 1,6
    @ 0,46 SAY RECNO( ) SIZE 1,3
ENDIF
IF NOT EMPTY(currwind)
    ACTIVATE WINDOW (currwind) SAME
ENDIF
RETURN .T.

```

```
PROCEDURE go_prev  
SKIP -1  
IF BOF( )  
    GO TOP  
ENDIF  
RETURN
```

```
PROCEDURE go_next  
SKIP 1  
IF EOF( )  
    GO BOTTOM  
ENDIF  
RETURN
```

DO CASE ... ENDCASE

Purpose	Executes the first statement block after DO CASE and before END CASE whose associated conditional statement evaluates to true (.T.)
Syntax	<pre> DO CASE CASE <expL1> <statements> [CASE <expL2> <statements> ... CASE <expLN> <statements>] [OTHERWISE <statements>] ENDCASE </pre>
See Also	DO WHILE ... ENDDO, FOR ... ENDFOR, IF ... ENDIF, IIF(), SCAN

Description DO CASE is used to execute a set of FoxPro commands based on the result of a logical condition. When DO CASE is executed, successive logical cases are evaluated; the results of the evaluations determine which set of commands is executed.

When the first logically true (.T.) CASE clause is encountered, the statements following it are executed. Execution of the statements continues until the next CASE or ENDCASE is reached. Execution then resumes with the first command following ENDCASE.

If a CASE clause is logically false (.F.), the statements following it up to the next CASE clause are ignored.

One and only one CASE clause is executed — the first true CASE clause. Any succeeding true CASE clauses are ignored.

If all of the CASE clauses evaluate to false (.F.), OTHERWISE determines if any additional statements are executed:

- If OTHERWISE is included, the statements following OTHERWISE are executed and execution skips to the first command following ENDCASE.
- If OTHERWISE is omitted, execution skips to the first command following ENDCASE.

Comments can be placed after DO CASE and ENDCASE on the same line. The comments are ignored during program compilation and execution.

Example In this example, FoxPro evaluates each CASE clause until the MONTH variable is found in one of the lists. The appropriate string is stored to the memory variable RPT_TITLE and the DO CASE structure is exited.

```
STORE CMONTH( DATE( ) ) TO month  && The month today

DO CASE  You can place comments here!

    CASE INLIST(month, 'January', 'February', 'March')
        STORE 'First Quarter Earnings' TO rpt_title

    CASE INLIST(month, 'April', 'May', 'June')
        STORE 'Second Quarter Earnings' TO rpt_title

    CASE INLIST(month, 'July', 'August', 'September')
        STORE 'Third Quarter Earnings' TO rpt_title

    OTHERWISE
        STORE 'Fourth Quarter Earnings' TO rpt_title
ENDCASE  Or you can place comments here!
WAIT WINDOW rpt_title NOWAIT
```

DO WHILE ... ENDDO

Purpose	Executes a block of statements within a conditional loop
Syntax	DO WHILE <expL> <statements> [LOOP] [EXIT] ENDDO
See Also	DO CASE ... ENDCASE, EXIT, FOR ... ENDFOR, IF ... ENDIF, IIF(), SCAN

Description A set of commands placed between DO WHILE and ENDDO is executed for as long as the logical expression <expL> remains true (.T.). A DO WHILE statement must have a corresponding ENDDO statement.

Comments can be placed after DO WHILE and ENDDO on the same line. The comments are ignored during program compilation and execution.

Clauses **<expL>**
 <expL> is a logical expression. The statements placed between DO WHILE and ENDDO are executed while <expL> evaluates to true (.T.).

<statements>

<statements> is the list of FoxPro commands to be executed while <expL> evaluates to true (.T.).

LOOP

LOOP can be placed anywhere between DO WHILE and ENDDO. LOOP returns program control directly back to DO WHILE.

EXIT

EXIT transfers program control from within the DO WHILE loop to the first command following ENDDO. EXIT can be placed anywhere between DO WHILE and ENDDO.

Example In this example, the payments for the month of June are totaled in the DO WHILE loop until the end of the file (EOF) is encountered. The DO WHILE loop is exited and the total is displayed.

```

CLOSE DATABASES
SET TALK OFF
USE invoices
jUNETOT = 0
DO WHILE .T.  You can place comments here!
    IF EOF( )
        EXIT
    ENDIF
    IF CMONTH(idate) = 'June'
        SKIP
        LOOP
    ENDIF
    jUNETOT = jUNETOT + itotal
    SKIP
ENDDO  Or you can place comments here!
? 'Total of invoice totals for June is '
?? jUNETOT

```

DOW()

Purpose	Returns the day of the week from a given date expression
Syntax	DOW(<expD>)
Returns	Numeric
See Also	CDOW(), DAY(), SYS()

Description The value returned by DOW() ranges from 1 (Sunday) through 7 (Saturday).

Parameter **<expD>**
DOW() returns the day number for the date expression <expD>.

Example

```
STORE {06/06/92} TO daynum
? DOW(daynum)
7

? CDOW(daynum)
Saturday
```

DTOC()

Purpose	Returns a character type date from a date expression
Syntax	DTOC(<expD> [, 1])
Parameters	<expD> A date expression 1 An option for indexing
Returns	Character
See Also	CTOD(), SET CENTURY, SET DATE, SYS()

Description DTOC() returns a character string corresponding to the date expression <expD>. The date format is determined by SET CENTURY and SET DATE.

Parameters <expD>
<expD> can be a date type memory variable, an array element, the name of a field, or a function that returns a date.

1
If the optional argument 1 is included, the date is returned in a format suitable for indexing. This is particularly useful in combination with TIME() for maintaining the table/.DBF records in chronological sequence.

For example, to order table/.DBF records in entry sequence, you could use this command:

```
INDEX ON DTOC(invdate, 1) + invtime TAG Timeindx  
  
INVDATA and INVTIME are fields containing the date and time when  
the data was entered in the record.
```

Examples STORE CTOD('10/31/92') TO thisdate
 ? DTOC(thisdate)
 10/31/92

STORE DTOC({10/31/92}+90) TO expiredate
 ? 'Your 90-day warranty expires', expiredate
 Your 90-day warranty expires 01/29/93

 ? DTOC({10/31/92},1)
 19921031

DTOR()

Purpose	Converts degrees to radians
Syntax	DTOR(<expN>)
Returns	Numeric
See Also	COS(), RTOD(), SIN(), TAN()

Description DTOR() converts the value of a numeric expression given in degrees to an equivalent value in radians. DTOR() is useful for working with the FoxPro trigonometric functions (ACOS(), COS(), RTOD(), SIN(), TAN()).

Use RTOD() to convert radians to degrees.

Parameter <expN>

<expN> is the numeric expression whose value you want to convert to radians. An angle expressed in a degree:minute:second format should be converted to its decimal equivalent.

Examples

```
? DTOR(0)
0.00
? DTOR(45)
0.79
? DTOR(90)
1.57
? DTOR(180)
3.14
? COS(DTOR(90))
0.00
```

DTOS()

Purpose	Returns a character-string date in the format YYYYMMDD from a specified date expression
Syntax	DTOS(<expD>)
Returns	Character
See Also	CTOD(), DTOC()

Description This function is useful for indexing tables/.DBFs on a date field. It is equivalent to DTOC() when its optional 1 argument is included.

The character string returned by DTOS() isn't affected by SET DATE or SET CENTURY.

Parameter **<expD>**
<expD> is the date expression you want to covert to an eight-digit character string.

EDIT

Purpose	Displays fields for editing
Syntax	EDIT [FIELDS <field list>] [<scope>] [FOR <expL1>] [WHILE <expL2>] [FONT <expC1> [, <expN1>]] [STYLE <expC2>] [FREEZE <field>] [KEY <expr1> [, <expr2>]] [LAST] [LEDIT] [REDIT] [LPARTITION] [NOAPPEND] [NOCLEAR] [NODELETE] [NOEDIT NOMODIFY] [NOLINK] [NOMENU] [NOOPTIMIZE] [NORMAL] [NOWAIT] [PARTITION <expN2>] [PREFERENCE <expC3>] [REST] [SAVE] [TIMEOUT <expN3>] [TITLE <expC4>] [VALID [:F] <expL3> [ERROR <expC5>]] [WHEN <expL4>] [WIDTH <expN4>] [[WINDOW <window name1>] [IN [WINDOW] <window name2> IN SCREEN]] [COLOR SCHEME <expN5> COLOR <color pair list>]
See Also	BROWSE, CHANGE, WTITLE()

Description EDIT works just like CHANGE. See CHANGE for complete information.

EJECT

Purpose	Sends a form feed to the printer
Syntax	EJECT
See Also	EJECT PAGE, ON PAGE, SET DEVICE, SET PRINTER, PCOL(), PROW(), System Memory Variables

Description EJECT advances the printer to the top of the next page. EJECT sends a form feed to the printer if the _PADVANCE system memory variable is set to FORMFEED. If _PADVANCE is set to LINEFEEDS, EJECT issues line feeds to advance to the top of the next page.

EJECT resets PCOL() and PROW() values to the current column and row position of the printer's print head, but doesn't affect the value of the _PAGENO or _PLINENO system variables.

Example In the following program example, the CUSTOMER fields COMPANY and ADDRESS1 are printed. (Make sure a printer is attached and turned on for this example.) When the number of rows printed is greater than 62, the page is ejected.

```
USE customer
SET DEVICE TO PRINTER
DO WHILE NOT EOF( )
    @ PROW( )+1,10 SAY 'Company: ' + company
    @ PROW( )+1,10 SAY 'Address: ' + address1
    @ PROW( )+1,1 SAY ''
    IF PROW( ) > 62
        EJECT
    ENDIF
    SKIP
ENDDO
SET DEVICE TO SCREEN
```

EJECT PAGE

Purpose	Sends a conditional page advance to the printer
Syntax	EJECT PAGE
See Also	EJECT, ON PAGE, SET ALTERNATE, SET DEVICE, SET PRINTER, PCOL(), PROW(), System Memory Variables

Description Use EJECT PAGE to advance streaming output. The advance depends upon the value in the _PADVANCE system memory variable and whether or not an ON PAGE is in effect.

If the system memory variable _PADVANCE is set to FORMFEED and an ON PAGE isn't in effect, EJECT PAGE performs the following actions:

- Sends a formfeed to the printer if the printer is on line.
- Sends linefeeds, as determined by the system memory variables _PLENGTH and _PLINENO to the screen and/or to an alternate file.
- Increments the system memory variable _PAGENO by one.
- Sets the system memory variable _PLINENO to 0.

If the system memory variable _PADVANCE is set to LINEFEEDS, and if an ON PAGE routine is in effect and _PLINENO is less than the page line number specified with the ON PAGE command, EJECT PAGE performs the following action:

- Sends to the printer, the desktop (FoxPro for MS-DOS), the main FoxPro window (FoxPro for Windows) and/or to an alternate file as many linefeeds as are necessary to advance to the top of the next page.

If an ON PAGE is not in effect, or if _PADVANCE is set to LINEFEEDS and _PLINENO is greater than the page line number specified with the ON PAGE, EJECT PAGE performs the following actions:

- Sends linefeeds, as determined by the system memory variables _PLENGTH and _PLINENO to the printer, the desktop or main FoxPro window and/or to the alternate file if an alternate file is open.
- Increments the system memory variable _PAGENO by one.
- Sets the system memory variable _PLINENO to 0.

EMPTY()

Purpose	Determines whether or not an expression is empty
Syntax	EMPTY(<expr>)
Returns	Logical
See Also	LEN()

Description EMPTY() returns true (.T.) if the expression <expr> is empty. If the expression isn't empty, EMPTY() returns false (.F.).

Parameter <expr>

The expression you include can be a character, numeric, date or logical expression, or the name of a memo or general field in an open table/DBF. EMPTY() returns true (.T.) when expressions of the following data types contain the indicated data:

Data Type Evaluation	
Data Type	Contains
Character	Nulls, spaces, tabs, carriage returns or line feeds or any combination of these
Numeric	0
Date	Null (i.e. CTOD(' '))
Logical	False (.F.)
Memo	Empty (no contents)
General	Empty (no OLE object)

Example In this example, a filter is set to browse only records whose memo field NOTES contains data.

```
USE salesman
SET FILTER TO NOT EMPTY(notes) && Non-empty memos
BROWSE LAST
SET FILTER TO
```

EOF()

Purpose	Determines whether or not the record pointer is positioned past the last record in the current or specified table/.DBF
Syntax	EOF([<expN> <expC>])
Returns	Logical
See Also	BOF()

Description EOF() returns a logical true value (.T.) if the record pointer reaches the end of the file (EOF). The end of the file is reached when the record pointer passes the last record in the table/.DBF or index file. For example, when a FIND, LOCATE or SEEK is unsuccessful, FoxPro moves the record pointer past the last record, and EOF() returns true (.T.). EOF() returns false (.F.) if the record pointer isn't at the end of the file.

Parameter <expN> | <expC>

You can test for the end of file for a table/.DBF open in another work area by specifying the work area number <expN> or the table/.DBF or work area alias <expC>.

EOF() returns false (.F.) if a table/.DBF isn't open in the work area you specify.

If you do not specify a work area or alias, the table/.DBF open in the selected work area is tested for the end of the file.

Example In this example, the CUSTOMER table/.DBF is opened and push buttons are displayed that enable you to move the record pointer forward or backward in the table/.DBF. When you choose a push button, FoxPro moves the record pointer. Depending on the direction moved, FoxPro checks to see if the record pointer is at the end of the file (EOF) or at the beginning of the file (BOF). If either condition is true, FoxPro displays a message and repositions the record pointer to the last or first record.

This example moves the record pointer to the last record in the table/.DBF so selecting the **Forward** push button takes you immediately to the end of the file.

```

CLOSE DATABASES
SET TALK OFF
CLEAR
USE customer
GO BOTTOM
DEFINE WINDOW mskip FROM 9,10 TO 13,65 DOUBLE COLOR SCHEME 5
ACTIVATE WINDOW mskip
@ 1,5 GET mchoice FUNCTION '*HN \<Forward;\<Backward;\<Quit';
    DEFAULT 0 SIZE 1,14 VALID CHOICE( )
READ CYCLE
RELEASE WINDOW mskip
PROCEDURE choice
DO CASE
    CASE mchoice = 1
        SKIP
        IF EOF('CUSTOMER')
            WAIT WINDOW 'End of file encountered' NOWAIT
            SKIP -1
        ENDIF
    CASE mchoice = 2
        SKIP -1
        IF BOF('CUSTOMER')
            WAIT WINDOW 'Beginning of file encountered' NOWAIT
            GO TOP
        ENDIF
    CASE mchoice = 3
        CLEAR READ
ENDCASE
ACTIVATE SCREEN
@ 2,1 SAY 'Company: ' + company
RETURN .T.

```

ERASE

Purpose	Erases a file from disk
Syntax	ERASE <file> ?
See Also	DELETE FILE, FILER

Description ERASE deletes a file from disk.



Exercise caution when using ERASE. Any file erased with this command cannot be retrieved. You aren't warned before the file is erased, even if SET SAFETY is ON.

Clauses **<file>**

Specify the file to be erased. The file extension must be included. Include the path with the file name if the file is on a different drive or directory than the current drive or directory.

ERASE doesn't support wildcard characters in <file>.

?

ERASE ? brings forward the Open dialog in which you can choose a file to delete.

Example

In this example, the structure of PARTS.DBF is first copied to a new table/.DBF named TEMP. Second, the table/.DBF structure is modified and a new table/.DBF named BACKUP is created. Third, all the records in PARTS are appended to BACKUP.DBF. Last, the AVGPRIce field is replaced with the average sales per unit and the temporary table/.DBF TEMP is erased.

```
CLOSE DATABASES
USE items
COPY STRUCTURE EXTENDED TO temp
USE temp
APPEND BLANK
REPLACE field_name WITH 'avgprice'
REPLACE field_type WITH 'N'
REPLACE field_len WITH 10
REPLACE Field_dec WITH 2
CREATE backup FROM temp
USE backup
APPEND FROM items
REPLACE ALL avgprice with ytdsales / ytdunits FOR ytdsales # 0
BROWSE
USE
ERASE temp.dbf
```

ERROR()

Purpose	Returns the error number for the error that triggered an ON ERROR
Syntax	ERROR()
Returns	Numeric
See Also	MESSAGE(), ON ERROR, RETRY, RETURN

Description ERROR() returns the number of the most recent error. An ON ERROR must be active for ERROR() to return a value other than 0.

When an error is trapped during program execution, the error type can be identified with ERROR() in an ON ERROR routine. The corresponding error message can be returned with MESSAGE().

The value the ERROR() function returns is reset by RETURN or RETRY.

See the Error Messages appendix in the FoxPro *Developer's Guide* for a numerical listing of each error number and the corresponding error message.

Example This example demonstrates a simple error-handling routine that displays a message when an error is occurs.

```
CLEAR
ON ERROR DO errhand WITH ERROR( ), MESSAGE( )

*** The next line generates an error - there is no BWSE command

BWSE
ON ERROR
RETURN

*** Errhand ***

PROCEDURE errhand
PARAMETER errnum,message
@ 23,0 SAY Message+' ERROR: '+STR(Ernum)
RETURN
```


EVALUATE()

Purpose	Evaluates a character expression and returns the result
Syntax	EVALUATE(<expC>)
Returns	Character, numeric, date, logical or memo
See Also	TYPE()

Description EVALUATE() is similar to TYPE() but returns the result of an expression instead of the expression type. An expression containing EVALUATE() cannot be optimized by Rushmore.

Parameter **<expC>**
 The <expC> parameter specifies the expression to evaluate. <expC> can be a literal character string, or a valid FoxPro expression, memory variable, array element or field of any data type, enclosed in character string delimiters.

Examples Here are some command window examples and their results:

```
? EVALUATE('5 * 4')
20
```

```
USE customer
? EVALUATE('customer.contact')
N. Baker
```

```
? EVALUATE('DATE( )')
10/31/92
```

```
STORE 'DATE( ) = {10/31/93}' TO halloween
? EVALUATE(halloween)
.T.
```

```
? TYPE(halloween)
L
```

Whenever possible, use EVALUATE() or a name expression to replace macro substitution using the & operator. EVALUATE and name expressions execute faster than macro substitution.

EXIT

Purpose	Exits a DO WHILE, FOR or SCAN loop
Syntax	EXIT
See Also	DO WHILE, FOR ... ENDFOR, SCAN ... ENDSCAN

Description EXIT transfers control from within a DO WHILE ... ENDDO, FOR ... ENDFOR or SCAN ... ENDSCAN loop to the command immediately following ENDDO, ENDFOR or ENDSCAN.

Example In this example, the payments for the month of May are totaled in a DO WHILE loop until the end of the file is encountered. The DO WHILE loop is then exited and the total is displayed.

```
CLOSE DATABASES
SET TALK OFF
USE invoices
maytot = 0

DO WHILE .T.
  IF EOF( )
    EXIT
  ENDIF
  IF CMONTH(ideate) != 'May'
    SKIP
    LOOP
  ENDIF
  maytot = maytot + itotal
  SKIP
ENDDO
? 'Total of invoices for May is '
?? maytot
```

EXP()

Purpose	Returns the value of e^x where x is a specified numeric expression
Syntax	EXP(<expN>)
Returns	Numeric
See Also	LOG(), SET DECIMALS

Description The mathematical function EXP() returns the value of e^x where x is a numeric expression. The value of e , the base of natural logarithms, is approximately 2.71828. The number of decimal places displayed by EXP() is specified with SET DECIMALS.

Parameter **<expN>**
The numeric expression <expN> is the exponent, x , in the exponential equation e^x .

Example

```
? EXP(0)
1.00

? EXP(1)
2.72
```

EXPORT

Purpose	Copies data from a FoxPro table/.DBF to a file in a different format
Syntax	EXPORT TO <file> [FIELDS <field list> <scope> [FOR <expL1> [WHILE <expL2> [NOOPTIMIZE] [TYPE] DIF MOD SYLK WK1 WKS WR1 WRK XLS
See Also	APPEND FROM, COPY TO, IMPORT

Description EXPORT copies records from the current table/.DBF to a new file. Use EXPORT to use FoxPro data in other software packages.

If the table/.DBF you are exporting from is indexed, the new file created is created in the indexed order.

Clauses <file>

Include the name of the file to which FoxPro exports data. If you do not include an extension with the file name, the default extension for the specified file type is assigned.

FIELDS <field list>

Include FIELDS <field list> to specify which fields are copied to the new file. If the FIELDS clause is omitted, all fields are copied to the new file. Memo and general fields are not copied to the new file even if their names are included in the field list.

<scope>

The scope clauses are: ALL, NEXT <expN>, RECORD <expN>, and REST. These are explained in the Overview of the FoxPro Language chapter of this manual. Commands which include <scope> operate only on the table/.DBF in the active work area.

You can specify a scope of records to copy to the new file. Only the records that fall within the range of records specified by the scope are copied to the new file.

The default scope for EXPORT is ALL records.

FOR <expL1>

If the FOR clause is included, only records that satisfy the logical condition <expL1> are copied to the new file. Including the FOR clause lets you conditionally copy records, filtering out undesired records.

Rushmore optimizes an EXPORT ... FOR <expL1> command if <expL1> is an optimizable expression. For best performance, use an optimizable expression in the FOR clause. A discussion of Rushmore optimizable expressions appears in the Optimizing Your Application chapter in the *FoxPro Developer's Guide*.

WHILE <expL2>

If the WHILE clause is included, records are copied to the new file for as long as the logical expression <expL2> evaluates to true (.T.).

NOOPTIMIZE

To cause Rushmore not to optimize EXPORT, include NOOPTIMIZE. For further information on NOOPTIMIZE, see SET OPTIMIZE in this chapter, and the Rushmore discussion in the Optimizing Your Application chapter of the *FoxPro Developer's Guide*.

TYPE

You must specify the type of file to be created. The TYPE key word is optional, but you must specify one of the following file types.

DIF

When you specify DIF, each field from a FoxPro table/.DBF becomes a vector (column) and each record becomes a tuple (row) in a DIF (Data Interchange Format) file, used by VisiCalc. The new file name is assigned a .DIF extension if an extension isn't included in <file>.

MOD

Use the MOD clause to export to a file in Microsoft Multiplan version 4.01 MOD format. The new file name is assigned an .MOD extension if you don't include an extension in <file>.

SYLK

A SYLK file is a Symbolic Link interchange format (used by Microsoft Multiplan) in which each field from a FoxPro table/.DBF becomes a column in the spreadsheet and each record becomes a row. By default, SYLK file names have no extension.

WK1

Include this option to create a Lotus 1-2-3 spreadsheet from a FoxPro table/.DBF. A .WK1 extension is assigned to the spreadsheet file name for use with Lotus 1-2-3 revision 2.x. Each field from the table/.DBF becomes a column in the new spreadsheet, and each record in the table/.DBF becomes a spreadsheet row.

WKS

A Lotus 1-2-3 spreadsheet can be created from a FoxPro table/.DBF with this option. A .WKS extension is assigned to the spreadsheet file name for use with Lotus 1-2-3 revision 1-A. Each field from the table/.DBF becomes a column in the new spreadsheet, and each record becomes a row in the spreadsheet.

WR1

Include this option to create a Lotus Symphony spreadsheet from a FoxPro table/.DBF. A .WR1 extension is assigned to the spreadsheet for use with Symphony version 1.01. Each field from the table/.DBF becomes a column in the new spreadsheet, and each record in the table/.DBF becomes a row in the spreadsheet.

WRK

A Lotus Symphony spreadsheet can be created from a FoxPro table/.DBF with this option. A .WRK extension is assigned to the spreadsheet file name for use with Symphony version 1.10. Each field from the table/.DBF becomes a column in the new spreadsheet, and each record in the table/.DBF becomes a row in the spreadsheet.

XLS

Include XLS to create a spreadsheet you can use in Microsoft Excel. Each field in the selected table/.DBF becomes a column in the spreadsheet; each table/.DBF record becomes a row in the spreadsheet. An .XLS file name extension is assigned to the newly created spreadsheet file unless you specify a different extension.

EXTERNAL

Purpose	Alerts the Project Manager to an undefined reference
Syntax	EXTERNAL ARRAY LABEL LIBRARY MENU PROCEDURE REPORT SCREEN <file list> <array list>
See Also	BUILD APP, BUILD PROJECT

Description Use EXTERNAL to include files and to resolve undefined references in a project created by the FoxPro Project Manager. EXTERNAL is used only by the Project Manager and is ignored during program execution.

For additional information on creating projects with the Project Manager, see the Project — The Main Organizing Tool chapter in the FoxPro *Developer’s Guide*.

Files whose names you specify with EXTERNAL are included in a project by the Project Manager. You must include LABEL, LIBRARY, MENU, PROCEDURE, REPORT or SCREEN before the file names to tell the Project Manager the type of files to include in the project.

The Project Manager must also be alerted to file names contained in a name expression or macro substitution. This assures that all necessary files are included in a project when the project is built. It must also be alerted to arrays that are created in another procedure or UDF (user-defined function).

For more information on name expressions and macro substitution, see the first chapter and the & topic in this manual. Whenever possible, use a name expression instead of macro substitution to improve performance.

Several key words in EXTERNAL identify an array or the different file types. Follow the key word with the corresponding array or file names. You can include multiple array or file names after the key word if you separate the names with commas.

Clauses ARRAY

When an array is created in a program and then used in a lower-level program, include EXTERNAL ARRAY with the array name in the lower-level program.

Example In the following example, the first program creates an array named INVOICE. The array is initialized and a lower-level program named DISPINVO is called. DISPINVO displays the contents of the array created in the higher-level program. The command EXTERNAL ARRAY INVOICE is included to alert the Project Manager.

```
DIMENSION invoice(4)
STORE 'Paid' TO invoice
DO dispinvo

*** Program dispinvo ***

PROCEDURE dispinvo
EXTERNAL ARRAY invoice
? invoice(1)
? invoice(2)
? invoice(3)
? invoice(4)
RETURN

*** End of dispinvo program ***
```

When an array is passed to a UDF or procedure, the corresponding array in the UDF or procedure must be identified to the project manager. Include the ARRAY key word with name of the array included in the PARAMETER statement.

Example

```
DIMENSION firstarray(2)  && Create an array
EXTERNAL ARRAY arraytwo  && Name of the array used in the UDF
SET TALK OFF
STORE 10 TO firstarray(1)
STORE 2 TO firstarray(2)
= ADDTWO(@firstarray)    && Pass the array by reference to a UDF

FUNCTION ADDTWO
PARAMETER arraytwo
CLEAR
arraytwo(1) = arraytwo(1) + 2
arraytwo(2) = arraytwo(2) + 2
? arraytwo(1)
? arraytwo(2)
```


LABEL

LABEL tells the Project Manager the file you include in an indirect file reference or macro substitution is a label definition file created with CREATE LABEL.

Example

```
EXTERNAL LABEL dunlabel  && LABEL FORM dunlabel must exist
STORE 'dunlabel' TO labelfile
LABEL FORM (labelfile) PREVIEW
```

LIBRARY

Include LIBRARY when a library file is referenced by indirect file referencing or macro substitution in SET LIBRARY.

Example

```
EXTERNAL LIBRARY regress  && LIBRARY regress must exist
STORE 'regress' TO statfunc
SET LIBRARY TO (statfunc)
```

MENU

If a menu definition file is included in an indirect file reference or macro substitution, include MENU and the menu file name. Menus can be created with the FoxPro Menu Builder. For more information about the menu builder, see the Menu Builder chapter in the *FoxPro User's Guide*.

Example

```
EXTERNAL MENU pickfile  && MENU pickfile must exist
STORE 'pickfile' TO sysmenpad1
MODIFY MENU (sysmenpad1)
```

PROCEDURE

PROCEDURE is used to identify an external procedure or UDF.

Example

```
EXTERNAL PROCEDURE delblank  && PROCEDURE delblank must exist
STORE 'delblank' TO trimblanks
DO (trimblanks) WITH 'A B C D E'
```

REPORT

REPORT alerts the Project Manager that the file you include in an indirect file reference or macro substitution is a report definition file created with CREATE REPORT.

Example

```
EXTERNAL REPORT overdue  && REPORT overdue must exist
STORE 'overdue' TO reportfile
REPORT FORM (reportfile) PREVIEW
```

EXTERNAL

SCREEN

If a screen definition file is included in an indirect file reference or macro substitution, include **SCREEN** and the screen file name. Screens can be created with the FoxPro Screen Builder. For more information on the screen builder, see the Screen Builder chapter in the *FoxPro User's Guide*.

Example

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

<file list> | <array list>

You can include multiple file names or array names in **EXTERNAL**. Separate the file or array names from each other with commas. All files whose names are listed must be of the specified file type.

FCHSIZE()

Purpose	Changes the size of a file opened with a low-level file function
Syntax	FCHSIZE(<expN1>, <expN2>)
Parameters	<expN1> The file handle <expN2> The size you want the file to be
Returns	Numeric
See Also	FCLOSE(), FCREATE(), FEOF(), FFLUSH(), FGETS(), FOPEN(), FPUTS(), FREAD(), FSEEK(), FWRITE()

Description You can change the size of a file opened with a low-level file function with FCHSIZE(). You can increase the file's size or truncate the file after a specified byte.

When a file's size is increased, FoxPro allocates sectors for the file on the drive where the file is opened. Since FCHSIZE() doesn't initialize the new file space, the space can contain previous data. Be sure to manage the new file space.

The final size of the file in bytes is returned. FoxPro returns -1 if FCHSIZE() is unable to change the file size, for example because of insufficient disk space, an invalid file handle is specified, or the file is read-only.



This function can be used to truncate a file to length 0.

Parameters <expN1>

Include the file handle <expN1> of the file whose size you wish to change. The file handle is returned by FOPEN() when you open the file or by FCREATE() when you create a file. If a file is opened with FOPEN(), it must be opened with write or read/write privileges in order for you to be able to change its size.

<expN2>

Specify the new file size in bytes in <expN2>. If <expN2> is less than the original file size, the file is truncated. If <expN2> is greater than the original file size, the file size is increased.

FCLOSE()

Purpose	Flushes and closes a file or communication port opened with a low-level file function
Syntax	FCLOSE(<expN>)
Returns	Logical
See Also	CLOSE ALL, FCHSIZE(), FCREATE(), FEOF(), FFLUSH(), FGETS(), FOPEN(), FPUTS(), FREAD(), FSEEK(), FWRITE()

Description If the file is successfully closed, FCLOSE() returns true (.T.) and releases the file handle. FCLOSE() returns false (.F.) if the file cannot be closed.

CLOSE ALL also closes low-level files.

Parameter **<expN>**

A numeric file handle is returned when you create a file with FCREATE() or open a file with FOPEN(). To close a low-level file with FCLOSE(), specify the file handle in the numeric expression <expN>.

If you open a communications port with a low-level file function, it can be closed by specifying the file handle assigned to the port.

Ports cannot be opened with low-level file functions in FoxPro for Windows.

FCOUNT()

Purpose	Returns the number of fields in the current or specified table/.DBF
Syntax	FCOUNT([<expN> <expC>])
Returns	Numeric
See Also	DBF(), FSIZE(), NDX(), SYS(7)

Description FCOUNT() returns the number of fields in an open table/.DBF.

Parameter <expN> | <expC>

You can return the number of fields in a table/.DBF open in another work area by specifying the work area number <expN> or the table/.DBF or work area alias <expC>.

0 is returned if a table/.DBF isn't open in the work area you specify.

If you don't specify a work area or alias, the number of fields is returned for the table/.DBF open in the selected work area.

Examples

```
CLOSE DATABASES
SELECT 0
USE salesman
SELECT 0
USE customer
? FCOUNT('CUSTOMER') && The number of fields in Customer.dbf
12

? FCOUNT('SALESMAN') && The number of fields in Salesman.dbf
10
```

FCREATE()

Purpose	Creates and opens a low-level file
Syntax	FCREATE(<expC> [, <expN>])
Parameters	<expC> <expN> The name of the file to be created MS-DOS attributes to be assigned to the file See the table below for possible values.
Returns	Numeric
See Also	CLOSE ALL, FCHSIZE(), FCREATE(), FEOF(), FFLUSH(), FGETS(), FOPEN(), FPUTS(), FREAD(), FSEEK(), FWRITE()

Description FCREATE() creates a low-level file and opens the file for use. If a file with the name you specify already exists, it is overwritten.

FCREATE() assigns a file handle number to the file, which you can use to identify the file in other FoxPro low-level file functions. FCREATE() returns the file handle number when a file is created, or returns -1 if a file cannot be created.



Assign the file handle number to a memory variable you can use with other low-level file I/O functions.

You cannot open a communication port with FCREATE(). Use FOPEN() to open a communication port.

Parameters **<expC>**

Specify the name of the file to create with <expC>. You can include a drive designator and path with the file name. If a drive designator or path isn't included, the file is created in the default directory.

<expN>

Include the numeric expression <expN> to specify the MS-DOS attributes of the file. The following table lists the file attributes you can specify.

File Attribute Numbers	
<expN>	File Attribute
0	Read/Write (default)
1	Read Only
2	Hidden
3	Read Only/Hidden
4	System
5	Read Only/System
6	System/Hidden
7	Read Only/Hidden/System

Use DISPLAY STATUS or LIST STATUS to display or print information about files created and opened with FCREATE(). DISPLAY STATUS and LIST STATUS give the following information about each file opened or created with a low-level file function:

- The drive, directory and file name.
- The file handle number.
- The file pointer position.
- The read/write attributes.

Example

```

IF FILE('errors.txt')                                && Does file exist?
    errfile = FOPEN('errors.txt',12)                 && If so, open read/write
ELSE
    errfile = FCREATE('errors.txt')                   && If not create it
ENDIF
IF errfile < 0                                         && Check for error opening file
    WAIT 'Cannot open or create output file' WINDOW NOWAIT
ELSE
    =FWRITE(errfile, 'Error information to be written here') && If no error, write to file
ENDIF

=FCLOSE(errfile)                                       && Close file
MODIFY FILE errors.txt NOWAIT                          && Open file in edit window

```

feof()

Purpose	Determines whether or not the file pointer is positioned at the end of a file
Syntax	feof(<expN>)
Returns	Logical
See Also	fchsize(), fclose(), fcreate(), fgets(), fopen(), fputc(), fread(), fseek(), fwrite()

Description This low-level file function returns true (.T.) if the file pointer is positioned at the end of a file opened with a low-level file function. feof() returns false (.F.) if the file pointer isn't at the end of the file.

Parameter **<expN>**
Include the file handle number <expN> of the file to check for the end of file condition. feof() always returns true (.T.) if you specify a file handle number of a communications port opened with fopen().

Example

```
*** Open the file test.txt ***

file_handle = fopen('test.txt')

*** Move the file pointer to BOF ***

m_position = fseek(file_handle, 0)

*** If file pointer is at BOF and EOF, the file is empty ***
*** Otherwise the file must have something in it ***

IF feof(file_handle)
    WAIT WINDOW 'This file is empty!' NOWAIT
ELSE
    WAIT WINDOW 'This file has something in it!' NOWAIT
ENDIF

= fclose(file_handle)
```


FERROR()

Purpose	Returns a number corresponding to the most recent low-level file function error
Syntax	FERROR()
Returns	Numeric
See Also	FCHSIZE(), FCLOSE(), FCREATE(), FEOF(), FFLUSH(), FGETS(), FOPEN(), FPUTS(), FREAD(), FSEEK(), FWRITE()

Description FERROR() returns 0 if a low-level file function executes successfully. A positive value is returned if the function didn't execute successfully. The following is a list of each error number returned by FERROR() and the cause of the error:

Error Numbers and Causes	
Error number	Error Cause
2	File not found
4	Too many files open (out of file handles)
5	Access denied
6	Invalid file handle given
8	Out of memory
25	Seek error (can't seek before the start of a file)
29	Disk full
31	Error opening file

Example The following example creates and opens a temporary file. If the file is successfully created, the program displays an input window where you can enter your first name, last name and position. This information is written to the temporary file.

FERROR()

```
SET TALK OFF
CLEAR
tmpfile = SYS(3)           && Create a temporary file name
handle = FCREATE(tmpfile)  && Open the temporary file

IF handle < 0               && Error generated creating file
    DO CASE                && Determine which error
        CASE FERROR( ) = 4
            reason = 'Too many files open (out of handles)'
        CASE FERROR( ) = 5
            reason = 'Access denied'
        CASE FERROR( ) = 8
            reason = 'Out of memory'
        CASE FERROR( ) = 29
            reason = 'Disk full'
        CASE FERROR( ) = 31
            reason = 'General Failure'
    ENDCASE
    *** Display the error ***
    WAIT WINDOW 'Can't create file: ' + reason NOWAIT
    RETURN
ELSE                        && File created successfully
    DEFINE WINDOW wingets FROM 10,20 TO 18,60 ;
    CLOSE FLOAT SHADOW SYSTEM ;
    COLOR SCHEME 8 TITLE ' Name Entry '
    ACTIVATE WINDOW wingets
    @ 1,5 SAY "First Name:" GET fname DEFAULT SPACE (15)
    @ 3,5 SAY "Last Name :" GET lname DEFAULT SPACE (15)
    @ 5,5 SAY "Position  :" GET pos  DEFAULT SPACE (15)
    READ
    =FWRITE(handle,'First name: ') && Output info entered by the user
    =FPUTS(handle,fname)
    =FWRITE(handle,'Last name: ')
    =FPUTS(handle,lname)
    =FWRITE(handle,'Position: ')
    =FPUTS(handle,pos)
    =FCLOSE(handle)           && Close the file
    DEACTIVATE WINDOW wingets
    WAIT 'Press a key to view the file' WINDOW
ENDIF
IF !WEXIST('wingets')
    DEFINE WINDOW wingets FROM 10,20 TO 18,60 ;
    CLOSE FLOAT SHADOW SYSTEM ;
    COLOR SCHEME 8 TITLE ' Name Entry '
ENDIF
MODIFY FILE (tmpfile) NOWAIT WINDOW wingets && View the file
CLEAR WINDOW && Remove the window
```

FFLUSH()

Purpose	Flushes to disk a file opened with a low-level function
Syntax	FFLUSH(<expN>)
Returns	Logical
See Also	FCHSIZE(), FCLOSE(), FCREATE(), FEOF(), FGETS(), FOPEN(), FPUTS(), FREAD(), FSEEK(), FWRITE()
Description	FFLUSH() flushes a file opened with a low-level file function to disk. FFLUSH() also releases the memory used by the file's buffer. FLUSH is different from the FFLUSH() <i>function</i>. FLUSH doesn't operate on low-level files but rather on tables/.DBFs and indexes.
Parameter	<expN> Include the file handle <expN> of the file to flush to disk.
Example	The following example opens and writes to a file named INPUT.DAT. After writing the first two strings, the program flushes the buffers to ensure that the strings are written to disk. It then writes the next two strings, flushes the buffers again and closes the file. <pre> IF FILE('input.dat') test_file = FOPEN('input.dat',2) ELSE test_file = FCREATE('input.dat') ENDIF io_bytes = FWRITE(test_file,'Test output') io_bytes = FWRITE(test_file,' for low level file I/O') flush_ok = FFLUSH(test_file) io_bytes = FWRITE(test_file,'Test output2') io_bytes = FWRITE(test_file,' for low level file I/O') flush_ok = FFLUSH(test_file) close_ok = FCLOSE(test_file) MODIFY FILE input.dat NOWAIT </pre>

FGETS()

Purpose	Returns a series of bytes from a file or a communications port opened with a low-level file function
Syntax	FGETS(<expN1> [, <expN2>])
Parameters	<expN1> The file handle <expN2> The number of bytes to be returned
Returns	Character
See Also	FCHSIZE(), FCLOSE(), FCREATE(), FEOF(), FFLUSH(), FOPEN(), FPUTS(), FREAD(), FSEEK(), FWRITE()

Description FGETS() returns a series of bytes from a file or communications port until it encounters a carriage return. You can read a file line by line by issuing a series of FGETS().

FGETS() returns a series of bytes as a character string. Data is returned starting from the current file's pointer position and continuing until a carriage return is encountered. The file pointer is then positioned on the byte immediately following the carriage return. The carriage return isn't returned as part of the string, and line feeds are discarded.

Parameters **<expN1>**

<expN1> specifies the numeric file handle of the file or communication port from which FGETS() returns data.

<expN2>

You can specify the number of bytes returned by FGETS() with <expN2>. FGETS() returns <expN2> bytes unless a carriage return is encountered first. FGETS() returns data between the starting file-pointer position and the carriage return if a carriage return is encountered within <expN2> bytes.

If <expN2> is omitted, FGETS() by default returns a maximum of 254 bytes.

Example

```

*** TEST.TXT must exist ***

STORE FOPEN('test.txt') TO file_handle    && Open the file

STORE FSEEK(file_handle, 0, 2) TO ifp_size && Move pointer to EOF

STORE FSEEK(file_handle, 0) TO ifp_top    && Move pointer to BOF

IF ifp_size <= 0 && Is File empty?
    WAIT WINDOW 'This file is empty!' NOWAIT

ELSE    && If not
    l_string = FGETS(file_handle, ifp_size) && Store contents to memory
    ? l_string
ENDIF

= FCLOSE(file_handle)    && Close the file

```

FIELD()

Purpose	Returns the name of a field, referenced by number, in the current or specified table/.DBF
Syntax	FIELD(<expN1> [, <expN2> <expC>])
Parameters	<expN2> <expN2> <expC> The field number The work area number for a table/.DBF The table/.DBF or work area alias
Returns	Character
See Also	DISPLAY STRUCTURE, FCOUNT()

Description Use FIELD() to return the name of a field in an open table/.DBF. FCOUNT() can be used to determine the number of fields in a table/.DBF.

Parameters **<expN1>**

Use <expN1> to specify a field number. If <expN1> is 1, the name of the first field in the table/.DBF is returned, if <expN1> is 2 the name of the second field is returned and so on. The null string is returned if <expN1> is greater than the number of fields. Field names are returned in upper-case characters.

<expN2> | <expC>

If you don't specify a work area number, or a table/.DBF or work area alias, a field name is returned from the table/.DBF open in the current work area.

You can return a field name from a table/.DBF open in another work area by specifying the work area number <expN> or the table/.DBF or work area alias <expC>. The null string is returned if a table/.DBF isn't open in the work area you specify.

Example

```
CLOSE DATABASES
CLEAR
USE customer
FOR mcount = 1 TO FCOUNT( )
    ? FIELD(mcount)
NEXT
```

FILE()

Purpose	Returns a logical true (.T.) if the specified file is found on disk
Syntax	FILE(<expC>)
Returns	Logical
See Also	FULLPATH(), LOCFILE(), SYS(3)
Description	Use FILE() to find a file on disk. A true value (.T.) is returned if the file can be found. If the file cannot be found, FILE() returns false (.F.).
Parameter	<expC> Specify the name of the file to locate with <expC>. The character expression <expC> must include the file's extension. FoxPro looks in the current directory for the file. If the file cannot be found in the current directory, the path established with SET PATH is searched. You can include a path with the file name to search for a file in a directory or on a drive other than the current directory or drive. The path established with SET PATH is searched if you specify a path with the file name and COMPATIBLE is set OFF. When COMPATIBLE is set OFF, drive and directory specifiers are ignored.

FILER

Purpose	Opens the file maintenance desk accessory
Syntax	FILER [LIKE <skel>] [NOWAIT] [IN [WINDOW] <window name> SCREEN]
See Also	COPY FILE, DELETE FILE, DIRECTORY, ERASE, RENAME

Description Issuing FILER opens the Filer desk accessory.

The Filer provides an efficient mechanism for maintaining order on your hard disk or floppies. You can copy files, create directories, move files and delete files. You can also view and change file attributes, find specific files, rename files or check the size of a file or a group of files. For more information on the Filer, see the *FoxPro User's Guide*.

Clauses **LIKE <skel>**

Include a file specification skeleton <skel> to display only those files that match the file skeleton. The skeleton can contain the wildcard characters * and ?.

NOWAIT

When FILER is issued in a program, the Filer is opened and program execution pauses until the user closes the Filer. Program execution then continues with the line immediately following FILER. Include NOWAIT to continue program execution after the Filer is opened. The program doesn't wait for the Filer to be closed, but continues execution on the line immediately following FILER NOWAIT.

IN [WINDOW] <window name>

IN WINDOW <window name> opens the Filer in the parent window named <window name>. The parent window must first be defined with DEFINE WINDOW.

The Filer can only be moved within the border of the parent window. If you move the parent window, the Filer moves with it.

The Filer doesn't assume the characteristics of the parent window. You can move, close and minimize the Filer in the parent window, and the Filer doesn't assume the colors of the parent window.

IN [WINDOW] SCREEN

You can place the Filer on the desktop in FoxPro for MS-DOS, or in the main FoxPro window in FoxPro for Windows, by including IN SCREEN or IN WINDOW SCREEN.

FILTER()

Purpose	Returns the filter expression specified in SET FILTER for the current or specified table/.DBF
Syntax	FILTER([<expN> <expC>])
Parameters	<expN> The work area number for a table/.DBF <expC> The table/.DBF or work area alias
Returns	Character
See Also	SET FILTER

Description FILTER() returns the filter expression specified in SET FILTER. The null string is returned if a filter isn't in effect. For additional information about creating a filter, see SET FILTER in this chapter.

Parameter **<expN> | <expC>**

You can create a filter for every FoxPro work area. If you don't specify a work area or alias, the filter expression for the currently selected work area is returned.

To return the filter expression for another work area, include the work area number <expN> or the table/.DBF or work area alias <expC>.

Example

```
CLOSE DATABASES
SET TALK ON
SELECT 0
USE customer

SET FILTER TO ono = '1'
?FILTER( )
ono = '1'

STORE FILTER('customer') TO savefilt  && Save filter expression
SET FILTER TO state = 'OH'
?FILTER( )
STATE="OH"

SET FILTER TO &savefilt  && Restore filter
?FILTER( )
ono = '1'
```

FIND

Purpose	Searches an indexed table/.DBF
Syntax	FIND <expC>
Remarks	Included for backward compatibility — Use SEEK
See Also	EOF(), FOUND(), INDEX, LOCATE, RECNO(), SEEK, SET EXACT, SET NEAR

Description FIND moves the record pointer to the first record in the table/.DBF whose index key matches the character expression <expC>. FIND requires the selected table/.DBF to be indexed. The match with the index expression must be exact unless SET EXACT is OFF.

If a matching record is found, RECNO() returns the record number of the matching record, FOUND() returns true (.T.) and EOF() returns false (.F.).

If a match isn't found, RECNO() returns the number of records in the table/.DBF plus one, FOUND() returns false (.F.) and EOF() returns true (.T.).

If SET NEAR is ON and FIND is unsuccessful, the record pointer is positioned immediately after the closest matching record. If SET NEAR is OFF and FIND is unsuccessful, the record pointer is positioned at the end of the file. In either case, RECNO() issued with an argument of 0 returns the record number of the closest matching record.

Example

```

CLOSE DATABASES
USE customer ORDER company && Open file and set order
SET EXACT OFF
STORE 'Jeannie' TO mval
FIND &mval
IF FOUND( )
    DISPLAY
ENDIF

```

FKLABEL()

Purpose	Returns the name of the function key corresponding to a numeric expression
Syntax	FKLABEL(<expN>)
Returns	Character
See Also	FKMAX(), SET COMPATIBLE, SET FUNCTION

Description FKLABEL() returns the name of the function key (F1, F2, F3 ...) from its corresponding function key number. Function keys can be programmed with SET FUNCTION.

The value returned by FKLABEL() is affected by SET COMPATIBLE. When COMPATIBLE is set to FOXPLUS (the default), the function keys are returned. When COMPATIBLE is set to DB4, the function key and function key combinations (F1, Ctrl+F1, Shift+F1, F2, Ctrl+F2, Ctrl+F2, Shift+F2 ...) are returned.

Parameter <expN>
The function key number is specified in <expN>. The value of <expN> should be from 0 through the number of function keys minus 1. FKLABEL() returns the null string if <expN> is greater than the number of function keys minus 1. The number of function keys can be determined with FKMAX().

Examples ? FKLABEL(0)
F1

? FKLABEL(45) && greater than FKMAX(), returns the null string

FKMAX()

Purpose	Returns the number of programmable function keys or function key combinations on your keyboard
Syntax	FKMAX()
Returns	Numeric
See Also	FKLABEL(), SET COMPATIBLE, SET FUNCTION

Description FKMAX() returns the number of programmable function keys or function key combinations available for your keyboard.

The value returned by FKMAX() is affected by SET COMPATIBLE. When COMPATIBLE is set to FOXPLUS (the default), the number of function keys is returned. When COMPATIBLE is set to DB4, the number of function key and function key combinations (F1, Ctrl+F1, Shift+F1, F2, Ctrl+F2, Shift+F2 ...) is returned.

Example

```
SET COMPATIBLE DB4
? FKMAX( )
29

SET COMPATIBLE FOXPLUS
? FKMAX( )
12
```

FLOCK()

Purpose	Attempts to lock the current or specified table/.DBF and returns true (.T.) if successful
Syntax	FLOCK([<expN> <expC>])
Parameters	<expN> The work area number <expC> The table/.DBF or work area alias
Returns	Logical
See Also	LOCK(), RLOCK(), SET REPROCESS, SET RELATION, UNLOCK, USE, SET EXCLUSIVE

Description FLOCK() attempts to place a lock on a table/.DBF file. True (.T.) is returned if the table/.DBF is successfully locked. False (.F.) is returned if the table/.DBF or a record in the table/.DBF is locked by another user.



If FLOCK() fails to lock a table/.DBF, it returns false (.F.) and does not generate an error. As a result, you can't use FLOCK() to trigger an ON ERROR routine.

When a table/.DBF is locked, the table/.DBF is available for both read and write access by the user who placed the lock. Other users on the network have read-only access to the table/.DBF. See SET EXCLUSIVE and USE for information on how to lock a table/.DBF and prevent access to it by other users.

A table/.DBF remains locked until it is unlocked by the user who placed the lock. The table/.DBF can be unlocked by issuing UNLOCK, closing the table/.DBF or quitting FoxPro. Tables/.DBFs can be closed with USE, CLEAR ALL or CLOSE DATABASES.

By default, FLOCK() attempts to lock a table/.DBF once. Use SET REPROCESS to automatically retry a table/.DBF lock when the first attempt fails. SET REPROCESS determines the number of lock attempts, or the length of time lock attempts are made, when the initial lock attempt is unsuccessful. See SET REPROCESS in this chapter for more information about using SET REPROCESS with table/.DBF locking.

You can establish relations between two or more tables/.DBFs with SET RELATION. Placing a file lock on a table/.DBF that is related to one or more tables/.DBFs doesn't place a file lock on the related tables/.DBFs. You must explicitly place and remove locks on the related tables/.DBFs.

Parameter <expN> | <expC>

FLOCK() attempts to place a lock on the table/.DBF in the current work area if you don't specify a work area number or a table/.DBF or work area alias. Include the work area number <expN> or the table/.DBF or work area alias <expC> to place a lock on a table/.DBF in another work area.

For additional information about record and file locking and sharing tables/.DBFs on a network, see the FoxPro in a Multi-User Environment chapter in the *FoxPro Developer's Guide*.

Example

```
CLOSE DATABASES
SET REPROCESS TO 3 SECONDS
SELECT * FROM parts INTO TABLE newparts

IF FLOCK( )

    *** Year end sales & commission initialization ***

    REPLACE ALL onhand WITH 0.00
    REPLACE ALL onorder WITH 0.00
    WAIT 'Initialization Complete - Press any key' WINDOW NOWAIT

ELSE

    *** File is locked, warn user ***

    WAIT WINDOW 'Unable to open sales file, try again later!' NOWAIT
ENDIF

BROWSE
CLOSE DATABASES
ERASE newparts.dbf
```

FLOOR()

Purpose	Returns the nearest integer that is less than or equal to the specified numeric expression
Syntax	FLOOR(<expN>)
Returns	Numeric
See Also	CEILING(), ROUND(), INT()

Description FLOOR() returns the nearest integer that is less than or equal to a numeric expression. FLOOR() rounds a positive number with a fractional portion down to the integer portion of the number, and rounds a negative number with a fractional portion down to the next lowest integer.

Parameter **<expN>**
FLOOR() returns the nearest integer that is less than or equal to <expN>.

Example

```

STORE 10.9 TO num1
STORE -10.1 TO num2
STORE 10.0 TO num3
STORE -10.0 TO num4

? FLOOR(num1)
10

? FLOOR(num2)
-11

? FLOOR(num3)
10

? FLOOR(num4)
-10

WAIT WINDOW STR(FLOOR(num1+num3)) NOWAIT
@ 5,0 SAY FLOOR (18.1)

```


FLUSH

Purpose	Saves table/.DBF and index modifications to disk
Syntax	FLUSH
See Also	CLOSE DATABASES, FFLUSH(), SET AUTOSAVE

Description FLUSH ensures that modifications you make to tables/.DBFs and their indexes are saved to disk. Modifications to all open tables/.DBFs and indexes are written to disk.

FoxPro automatically saves changes to disk when:

- You close a table/.DBF with USE or CLOSE DATABASES. Only information for the file or files you close is flushed to disk.
- You unlock a record or file. Only information for the record or file unlocked is flushed to disk.

FONTMETRIC()

Purpose	Returns font attributes for installed fonts for Windows
Syntax	FONTMETRIC(<expN1> [, <expC1>, <expN2> [, <expC2>]])
Parameters	<expN1> The font attribute <expC1> The font name <expN2> The font point size <expC2> The font style code
Returns	Numeric
Remarks	FoxPro for Windows only
See Also	AFONT(), GETFONT(), SYSMETRIC(), TXTWIDTH(), WFONT()

Description FONTMETRIC() returns font attributes for the current font for the active output window. WFONT() can be used to determine the current window font.

For additional information on installing or removing fonts for Windows, refer to the Control Panel chapter in your *Microsoft Windows User's Guide*.

Parameters **<expN1>**

<expN1> determines the font attribute returned by FONTMETRIC(). An attribute for the current font for the active output window is returned if <expC1> and <expN2> are omitted.

The following table lists values for <expN1> and the corresponding font attributes returned.

<expN1>	Attribute	<expN1>	Attribute
1	Character height in pixels	11	Strikeout (0 = no, non-zero = yes)
2	Character ascent (units above baseline) in pixels	12	First character defined in font
3	Character descent (units below baseline) in pixels	13	Last character defined in font
4	Leading (space between lines) in pixels	14	Default character (substituted for characters not in font)
5	Extra leading in pixels	15	Word break character
6	Average character width in pixels	16	Pitch and family
7	Maximum character width in pixels	17	Character set
8	Font weight	18	Overhang (extra added width)
9	Italic (0 = no, non-zero = yes)	19	Horizontal aspect for font device
10	Underlined (0 = no, non-zero = yes)	20	Vertical aspect for font device

For more information about the numeric values returned by FONTMETRIC(), refer to the TEXTMETRIC function in the *Microsoft Windows Programmer's Reference*.

<expC1>

To return an attribute for a specific installed font, specify the font's name with <expC1>.

<expN2>

Specify the point size with <expN2> of the font specified with <expC1>.

<expC2>

Specify a font style code with <expC2> to return an attribute for a specific font style. If <expC2> is omitted, the attribute is returned for the normal font style.

<expC2> can be a character or a combination of characters listed in the following font style table. For example, the combination BI specifies the bold italic font style.

Character	Font Style
B	Bold
I	Italic
N	Normal
O	Outline
Q	Opaque
S	Shadow
—	Strikeout
T	Transparent
U	Underline

FOPEN()

Purpose	Opens a file or communication port for use with low-level file functions
Syntax	FOPEN(<expC> [, <expN>])
Parameters	<expC> The communication port or file name <expN> The file's read/write privileges or buffering scheme
Returns	Numeric
Remarks	Ports cannot be opened in FoxPro for Windows
See Also	CLOSE ALL, FCHSIZE(), FCLOSE(), FCREATE(), FEOF(), FFLUSH(), FGETS(), FPUTS(), FREAD(), FSEEK(), FWRITE()

Description If FOPEN() successfully opens a file or communications port, the file handle number of the file or port is returned. FOPEN() returns -1 if the file or port cannot be opened.



Assign the file handle number to a memory variable for use with other low-level file functions.

Information about files opened with FOPEN() can be displayed or sent to a printer with DISPLAY STATUS or LIST STATUS. DISPLAY STATUS and LIST STATUS give the following information about each file opened with a low-level function:

- The drive, directory and file name.
- The file handle number.
- The file pointer position.
- The read/write attributes.

Use CLOSE ALL to close all files opened with a low-level function.

Parameters <expC>

Include in <expC> the name of the file or communication port to open. If <expC> is a file name, it can include a path to open files in directories or drives not in the current search path. If a path isn't included, FoxPro searches for the file in the following order:

- The current directory.
- The path established with SET PATH.

If <expC> is a communications port, the port must first be initialized with the MS-DOS MODE command. Specify the name of the port, for example COM1 or COM2.

Ports cannot be opened with low-level file functions in FoxPro for Windows.

<expN>

Use <expN> to specify read/write privileges or buffering scheme for the file you open. The following table lists each number you can include in <expN>, and the read/write file privileges and buffering scheme it establishes.

Privileges and Buffering		
<expN>	Read/Write Privileges	Buffering Scheme
0	Read Only (default)	Buffered
1	Write Only	Buffered
2	Read and Write	Buffered
10	Read Only	Unbuffered
11	Write Only	Unbuffered
12	Read and Write	Unbuffered

If <expN> isn't included, or if <expN> evaluates to 0, the file is opened as read only and is buffered. Communication ports should always be opened unbuffered.

Example

```

IF FILE('errors.txt')                && Does file exist?
    errfile = FOPEN('errors.txt',12) && If so, open read/write
ELSE
    errfile = FCREATE('errors.txt')  && If not create it
ENDIF
IF errfile < 0                        && Check for error opening file
    WAIT 'Cannot open or create output file' WINDOW NOWAIT
ELSE                                  && If no error, write to file
    =FWRITE(errfile, 'Error information to be written here')
ENDIF

=FCLOSE(errfile)                     && Close file
MODIFY FILE errors.txt NOWAIT        && Open file in edit window

```

FOR ... ENDFOR

Purpose	Executes the commands after FOR and before ENDFOR a specified number of times
Syntax	FOR <memvar> = <expN1> TO <expN2> [STEP <expN3>] <statements> [EXIT] [LOOP] ENDFOR NEXT
See Also	DO CASE, DO WHILE, IF, SCAN

Description FOR ... ENDFOR executes a set of statements within a loop a specified number of times. A memory variable or an array element is used as a counter to specify how many times the statements inside the loop are executed.

Program statements after FOR are executed until ENDFOR or NEXT is reached. The counter <memvar> is then incremented by the value of <expN3>. The counter is incremented by 1 if the STEP clause is omitted. The counter is then compared with the final value <expN2>. If the counter is less than or equal to the final value <expN2>, the statements following the FOR clause are executed again. If the counter is greater than the final value <expN2>, program execution branches outside the FOR ... ENDFOR loop and continues with the first command following ENDFOR or NEXT.



The values of <expN1>, <expN2> and <expN3> are read initially only. However, changing the value of the counter <memvar> inside the loop affects the number of times the loop is executed.

If the value of <expN3> is negative and the initial value <expN1> is greater than the final value <expN2>, the counter is decreased each time through the loop.

Clauses **<memvar>**
<memvar> is a memory variable or an array element that acts as a counter. The memory variable or array element doesn't have to exist before FOR ... ENDFOR is executed.

<expN1> TO <expN2>

<expN1> is the initial value of the counter, and <expN2> is the final value of the counter.

STEP <expN3>

<expN3> is the amount the counter is incremented or decremented. If <expN3> is negative, the counter is decremented. If the STEP clause is omitted, the counter is incremented by 1.

<statements>

This clause specifies the FoxPro commands to be executed. <statements> can include any number of commands.

EXIT

EXIT transfers control from within the FOR ... ENDFOR loop to the command immediately following the ENDFOR key word. You can place EXIT anywhere between FOR and ENDFOR.

LOOP

LOOP can be placed anywhere between FOR and ENDFOR. Including LOOP returns control directly back to the FOR clause without executing the statements between LOOP and ENDFOR. The counter is incremented or decremented as if ENDFOR were reached.

Examples

In the first example, the numbers 1 through 10 are displayed. The second example uses memory variables for the initial, final and STEP values to display all even-numbered records from 2 through 10 in CUSTOMER.DBF.

```

CLOSE DATABASES
CLEAR
FOR mcount = 1 TO 10
    ? mcount
ENDFOR

SET TALK OFF
USE customer
GO TOP
STORE 2 TO I
STORE 10 TO J
STORE 2 TO K
FOR mcount = I TO J STEP K
    GOTO mcount
    DISPLAY UPPER (company)
ENDFOR
USE && Close customer

```

FOUND()

Purpose	Returns true if CONTINUE, FIND, LOCATE or SEEK is successful
Syntax	FOUND([<expN> <expC>])
Parameters	 <expN> The work area number for a table/.DBF <expC> The table/.DBF alias
Returns	Logical
See Also	CONTINUE, EOF(), FIND, LOCATE, SEEK

Description FOUND() returns a logical value that indicates whether the most recently executed CONTINUE, FIND, LOCATE or SEEK was successful or the record pointer was moved in a related table/.DBF.

True (.T.) is returned if the search is successful, and false (.F.) is returned if the search is unsuccessful.



This function is useful for determining whether a child table/.DBF has a record that matches the parent record.

Parameter <expN> | <expC>

FOUND() can return a value for each work area. If you don't specify a work area or table/.DBF alias, FOUND() returns a value for the table/.DBF in the current work area. To return a value for a table/.DBF in a specific work area, include the work area number <expN> or the table/.DBF alias <expC>.

If a table/.DBF isn't open in the specified work area, FOUND() returns false (.F.). If no table/.DBF has the alias you specify, the message "Alias not found" is displayed.

Examples

```
SET TALK WINDOW
SET SAFETY OFF
CLOSE DATABASES
USE invoices ORDER cno
SELECT 0
USE customer
SET RELATION TO cno INTO invoices
COPY TO newfile FOR FOUND('invoices')
SELECT 0
USE newfile
BROWSE
CLOSE DATABASES
ERASE newfile.dbf
```

In the following example, all customers from New York City are counted.

```
SET TALK OFF
CLOSE DATABASES
USE customer
STORE 0 TO mcount
LOCATE FOR city = 'New York'
DO WHILE FOUND( )
    mcount = mcount + 1
    CONTINUE
ENDDO
WAIT WINDOW 'Total customers from New York City: ' ;
    + LTRIM(STR(mcount)) NOWAIT
USE && Close table/.DEF
```

FPUTS()

Purpose	Writes a character string, carriage return and line feed to a file or a communication port opened with a low-level file function
Syntax	FPUTS(<expN1>, <expC> [, <expN2>])
Parameters	<expN1> <expC> <expN2> The file handle of the file or port The character string to be written to the file or port The number of characters to write to the file or port
Returns	Numeric
See Also	FCHSIZE(), FCLOSE(), FCREATE(), FEOF(), FFLUSH(), FGETS(), FOPEN(), FREAD(), FSEEK(), FWRITE()

Description While similar to FWRITE(), FPUTS() places a carriage return and a line feed at the end of the character string it writes to the file or communication port.

FPUTS() returns the number of bytes written to the file or port. 0 is returned if FPUTS() can't write to the file or port for any reason.

Parameters **<expN1>**
FPUTS() writes to the file or communication port specified with the file handle number <expN1>.

<expC>
The character expression that FPUTS() writes to the file or port is specified with <expC>.

<expN2>
FPUTS() writes the entire character expression <expC> to the file or port unless <expN2> is included. When <expN2> is included, <expN2> characters are written to the file or port. If <expN2> is less than the number of characters in <expC>, only <expN2> characters are written to the file or port. All of <expC> is written to the file or port if <expN2> is equal to or greater than the number of characters in <expC>.

Example The following example creates and opens a temporary file. If the file is successfully created, a window is displayed in which you can enter your first name, last name and position. This information is written to the temporary file, which the FoxPro editor displays. If the file isn't created, the FoxPro editor displays the empty file.

```

SET TALK OFF
CLEAR
tmpfile = SYS(3) && Create a temporary file name
handle = FCREATE(tmpfile) && Open the temporary file

IF handle < 0
    DEFINE WINDOW alert FROM 7,17 TO 12,60 DOUBLE COLOR SCHEME 7

    DO CASE && Unable to open the temporary file.
        CASE FERROR( ) = 4
            reason = 'Too many files open (out of handles)'
        CASE FERROR( ) = 5
            reason = 'Access denied'
        CASE FERROR( ) = 8
            reason = 'Out of memory'
        CASE FERROR( ) = 29
            reason = 'Disk full'
        CASE FERROR( ) = 31
            reason = 'General Failure'
    ENDCASE

    ACTIVATE WINDOW alert
    @ 1,7 SAY 'Unable to open file'
    @ 2,7 SAY 'Reason: ' + reason
    @ 3,7 SAY 'Press a key to exit'
    =INKEY(0)
    DEACTIVATE WINDOW alert
    CANCEL
ELSE
    STORE SPACE(15) TO fname, lname, pos
    DEFINE WINDOW wingets FROM 10,20 TO 18,60 ;
    CLOSE FLOAT SHADOW SYSTEM COLOR SCHEME 8 TITLE ' Name Entry '
    ACTIVATE WINDOW wingets
    @ 1,5 SAY 'First Name:' GET fname
    @ 3,5 SAY 'Last Name : ' GET lname
    @ 5,5 SAY 'Position  :' GET pos
    READ
    =FWRITE(handle,'First name: ')
    =FPUTS(handle,fname)
    =FWRITE(handle,'Last name: ')
    =FPUTS(handle,lname)
    =FWRITE(handle,'Position: ')
    =FPUTS(handle,pos)
    =FCLOSE(handle) && Close the file
    DEACTIVATE WINDOW wingets
    WAIT 'Press a key to view the file' WINDOW
ENDIF

```

FPUTS()

```
IF !WEXIST('wingets')
    DEFINE WINDOW wingets FROM 10,20 TO 18,60 ;
        CLOSE FLOAT SHADOW SYSTEM ;
        COLOR SCHEME 8 TITLE ' Name Entry '
ENDIF

MODIFY FILE (tmpfile) WINDOW wingets NOWAIT && See created file
CLEAR WINDOW
```

FREAD()

Purpose	Returns a specified number of bytes from a file or a communication port opened with a low-level function
Syntax	FREAD(<expN1>, <expN2>)
Parameters	<expN1> The file handle number of the file or port <expN2> The number of bytes to be returned
Returns	Character
See Also	FCHSIZE(), FCLOSE(), FCREATE(), FEOF(), FFLUSH(), FGETS(), FOPEN(), FPUTS(), FSEEK(), FWRITE()

Description FREAD() returns, as a character string, a specified number of bytes from a file or communication port.

Parameters **<expN1>**
FREAD() returns data from the file or communication port specified by the file handle number <expN1>.

<expN2>
The number of bytes returned by FREAD() is determined by <expN2> and the position of the file pointer. FREAD() returns data starting from the current file pointer position and continues until it returns <expN2> bytes, or until it encounters the end of the file, whichever is first.

Example

```

*** TEST.TXT must exist ***

STORE FOPEN('test.txt') TO file_handle      && Open the file

STORE FSEEK(file_handle, 0, 2) TO ifp_size && Move pointer to EOF

STORE FSEEK(file_handle, 0) TO ifp_top      && Move pointer to BOF

IF ifp_size <= 0          && Is File empty?
    WAIT WINDOW 'This file is empty!' NOWAIT

ELSE
    && If not empty
    l_string = FREAD(file_handle, ifp_size) && Store to memory
    ? l_string
ENDIF

= FCLOSE(file_handle)    && Close the file

```

FSEEK()

Purpose	Moves the file pointer in a file opened with a low-level file function
Syntax	FSEEK(<expN1>, <expN2> [, <expN3>])
Parameters	<expN1> The file handle <expN2> The number of bytes to move the file pointer <expN3> If 0, file pointer moves relative to the beginning of the file (the default) If 1, file pointer moves relative to the current file pointer position If 2, file pointer moves relative to the end of the file
Returns	Numeric
See Also	FCHSIZE(), FCLOSE(), FCREATE(), FEOF(), FFLUSH(), FGETS(), FOPEN(), FPUTS(), FREAD(), FWRITE()

Description Use FSEEK() to move the file pointer within a file.

After moving the file pointer, FSEEK() returns the number of bytes the file pointer is from the beginning of the file. The file pointer can also be moved with FREAD() and FWRITE().

Parameters **<expN1>**

FSEEK() moves the file pointer in the file with the file handle number <expN1>. A file handle number is returned by FCREATE() or FOPEN() when the file is created and/or opened.

<expN2>

The number of bytes to move the file pointer is specified by <expN2>. The file pointer is moved toward the end of the file if <expN2> is positive. The file pointer is moved toward the beginning of the file if <expN2> is negative.

<expN3>

By default, the file pointer is moved relative to the beginning of the file. You can alternatively move the file pointer relative to the beginning or the end of the file by including <expN3>. The following table lists each value for <expN3> and how it causes the file pointer to move.

FSEEK() File Pointer Movement	
<expN3>	Movement Relative to
0	The beginning of the file (default)
1	The current file pointer position
2	The end of the file

Example

The following user-defined function uses FSEEK() to return the size of a file. If you don't pass parameters to the UDF, it returns -2. If the file cannot be found, the UDF returns -1.

```

FUNCTION fsize2
PARAMETERS mfile && File to be checked
PRIVATE mhandle, fsize

IF PARAMETERS( ) = 0
    RETURN -2                && Return -2 if no parameter passed
ELSE
    IF !FILE(mfile)
        RETURN -1            && Return -1 if file does not exist
    ENDIF
    mhandle = FOPEN(mfile)    && Open file
    fsize = FSEEK(mhandle,0,2) && Determine file size, assign to fsize
    =FCLOSE(mhandle)         && Close file
    RETURN fsize              && Return value

```

FSIZE()

Purpose	Returns the size, in bytes, of a specified field
Syntax	FSIZE(<expC1> [, <expN> <expC2>])
Parameters	<expC1> The name of the field <expN> The work area number for a table/.DBF <expC2> The table/.DBF alias
Returns	Numeric
See Also	DISPLAY STRUCTURE, FCOUNT(), LIST

Description FSIZE() can be used to return the size of character and numeric fields. Date, logical, memo and general fields have a constant default size of 8, 1, 10 and 10 bytes respectively.

The size of a field can be displayed with DISPLAY STRUCTURE and LIST STRUCTURE.

Parameters **<expC1>**
Specify the name of the field with the character expression <expC1>.

<expN> | <expC2>
The size of the specified field is returned from the table/.DBF in the current work area if you don't include a work area or table/.DBF alias. To return a field size from a table/.DBF in another work area, include the work area number <expN> or the table/.DBF alias <expC>.

If no table/.DBF has the alias you specify, the message "Alias not found" is displayed.

Examples

```
USE customer
? FSIZE('contact')
20

? FSIZE('cno')
5
```

FULLPATH()

Purpose	Returns the MS-DOS path for a specified file
Syntax	FULLPATH(<file name1> [, <expN> <file name2>])
Parameters	<file name1> The name of a file <expN> Specifies which path to search <file name2> The name of a file
Returns	Character
See Also	DBF(), FILE(), LOCFILE(), SYS(2014)

Description FULLPATH() returns the MS-DOS path for a file or the full path relative to another file.

When used with SYS(2014), FULLPATH() can make applications portable. That is, given the location of the current program and the location of other files used by the program, FULLPATH() and SYS(2014) can be used to obtain the paths for these files. Once the paths are determined, the files can be accessed by the program.

Clauses **<file name1>**

The FoxPro path is searched for the file specified with <file name1>. Be sure to include the file name extension. If the file is located in the FoxPro path, the path is returned with the file name. The FoxPro path can be set with SET PATH.

If the file can't be found in the FoxPro path, a path and the file name is returned as if the file is found in the current default directory. The default directory can be specified with SET DEFAULT.

<expN>

If <expN> is included, the MS-DOS path is searched instead of the FoxPro path for the specified file. <expN> can have any numeric value. If the file can't be located in the MS-DOS path, a path and the file name is returned as if the file had been found in the current default directory.

<file name2>

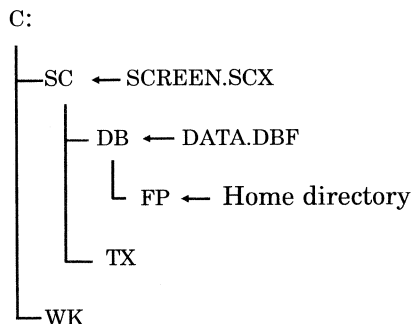
You can specify a second file name with <file name2>. Be sure to include the file name extension. FULLPATH() returns the path for the first file relative to the second file.

Example

This example illustrates how to use FULLPATH() and SYS(2014) to locate a .DBF used in the creation of a screen.

When you create a screen with the FoxPro Screen Builder, information about the screen is stored in a screen tabl (in this example, SCREEN.SCX). Fields in another .DBF (DATA.DBF) can be used in the screen you create. The location of DATA.DBF is stored in the SCREEN table when you save the screen.

The following directory structure is used for this example. The SCREEN table is located in the SC directory off the root of the C: drive (C:\SC\SCREEN.SCX). DATA.DBF is located in the DB directory, and the home directory is the FP directory.



The SCREEN table stores the location of DATA.DBF relative to the SCREEN table – DB\DATA.DBF.

A program can generate USE to open DATA.DBF. It must locate DATA.DBF from the path information stored in the SCREEN table. The program can FULLPATH():

```
FULLPATH('DB\DATA.DBF', 'C:\SC\SCREEN.SCX')
```

FULLPATH() returns

```
'C:\SC\DB\DATA.DBF'
```

This is then used in SYS(2014) with the home directory:

```
SYS(2014, 'C:\SC\DB\DATA.DBF', 'C:\SC\DB\FP')
```

SYS() returns

```
'\SC\DB\DATA.DBF'
```

The following command then opens the DATA table/.DBF:

```
USE LOCFILE('\SC\DB\DATA.DBF', 'DBF', 'Where is ... ')
```

If the .DBF has been moved from the location stored in the SCREEN table, the Open dialog is displayed to locate the file.

The following lists the paths SYS(2014) returns for different home directories when given 'C:\SC\DB\DATA.DBF':

Home Directory	Path Returned
C:\	SC\DB\DATA.DBF
C:\WK\	\SC\DB\DATA.DBF
C:\SC\TX\	\SC\DB\DATA.DBF
C:\SC\DB\FP	\SC\DB\DATA.DBF
C:\SC\DB\	DATA.DBF
C:\SC\	DB\DATA.DBF

Use FULLPATH(SET('DEFAULT')) to obtain the default directory and drive.

FUNCTION

Purpose	Identifies the start of a user-defined function definition
Syntax	FUNCTION <function name>
See Also	PARAMETERS, PARAMETERS(), PRIVATE, PROCEDURE, PUBLIC, RETURN, SET PROCEDURE, SET UDFPARMS, User-Defined Functions

Description In many programs, certain routines are frequently repeated. Defining commonly used routines as separate functions reduces program size and complexity and facilitates program maintenance.

FUNCTION <function name> is a statement within a program. It designates the beginning of a function in a program and identifies the function by name. Function names can be up to 10 characters long, must begin with a letter or an underscore, and can contain any combination of letters, numbers and underscores.

FUNCTION <function name> is followed by a series of commands that make up the function. You can include RETURN as the last line of a function, although an implicit RETURN is automatically executed.

The next FUNCTION statement or the end of the file, whichever comes first, signals the end of the function.

When a function is executed with DO, the function is searched for in a specific order:

1. First, FoxPro searches the file containing the DO command.
2. If the function isn't found there, FoxPro searches the current procedure file (if one is open).
3. If the function isn't found in the current procedure file, FoxPro searches the programs in the execution chain. Program files are searched from the most recently executed program through the first program executed.
4. If the function is still not found, FoxPro searches for a stand-alone program. If a matching program file is found, the program is executed. Otherwise, the error message "File does not exist" is generated.

Issue DO IN <file name> to search only the specified file.

By default, parameters are passed to functions by value. See SET UDFPARMS in this chapter for information on passing parameters to functions by reference. A maximum of 24 parameters can be passed to a function.

Example The following example creates two functions: LONGDATE and QUARTER. LONGDATE converts a date variable to a character string suitable for use in reports. QUARTER returns the quarter of the year when passed a month number. These two functions can be used to print a heading in a report.

```

SET CENTURY ON
@ 5,0 SAY longdate({08/19/89})
@ 6,5 SAY quarter(DATE( ))

FUNCTION longdate
PARAMETER mdate
RETURN CDOW(mdate) + ', ' + MDY(mdate)

FUNCTION quarter
PARAMETER qdate
DO CASE
    CASE MONTH (qdate) > 9
        RETURN 'Fourth Quarter'
    CASE MONTH (qdate) > 6
        RETURN 'Third Quarter'
    CASE MONTH (qdate) > 3
        RETURN 'Second Quarter'
    OTHERWISE
        RETURN 'First Quarter'
ENDCASE

```

FV()

Purpose	Returns the future value of an investment
Syntax	FV(<expN1>, <expN2>, <expN3>)
Parameters	<expN1> The constant periodic payment <expN2> The periodic interest rate <expN3> The number of periods over which payments are made
Returns	Numeric
See Also	CALCULATE, PV()

Description FV() computes the future value of a series of constant periodic payments earning fixed compound interest. The future value is the total of all payments and the interest.

Parameters **<expN1>**
The constant periodic payment (which can be negative or positive) is specified with <expN1>.

<expN2>
The periodic interest rate is specified with <expN2>. If the interest rate is annual but the payments are made monthly, divide the annual interest rate by 12.

<expN3>
<expN3> specifies the number of periods over which payments are made. FV() assumes that the periodic payments are made at the end of each period.

Example

STORE 500 TO payment	&& Monthly payment
STORE .075/12 TO interest	&& 7.5% annual interest rate
STORE 48 TO periods	&& Four years (48 months)
? FV(payment, interest, periods)	&& Calculate future value
27887.93	

FWRITE()

Purpose	Writes a character string to a file or a communication port opened with a low-level file function
Syntax	FWRITE(<expN1>, <expC> [, <expN2>])
Parameters	<expN1> <expC> <expN2> The file handle of the file or port The character string to be written to the file or port The number of characters to write to the file or port
Returns	Numeric
See Also	FCHSIZE(), FCLOSE(), FCREATE(), FEOF(), FFLUSH(), FGETS(), FOPEN(), FPUTS(), FREAD(), FSEEK()

Description FWRITE() writes a character string to a file or a communication port. Unlike FPUTS(), FWRITE() doesn't place a carriage return and a line feed at the end of the character string.

FWRITE() returns the number of bytes written to the file or communication port. If FWRITE() can't write to the file or port for any reason, 0 is returned.

Parameters **<expN1>**
FWRITE() writes to the file or communication port specified by the file handle number <expN1>.

<expC>
FWRITE() writes the character expression <expC> to the file or port specified with <expN1>.

<expN2>
FWRITE() writes the entire character expression to the file or port unless <expN2> is included. When <expN2> is included, <expN2> characters are written to the file or port. If <expN2> is less than the number of characters in <expC>, only <expN2> characters are written to the file or port. All of the characters in <expC> are written to the file or port if <expN2> is equal to or greater than the number of characters in <expC>.

Example For an example using FWRITE() see the example for FPUTS(), earlier in this chapter.

GATHER

Purpose	Stores the contents of memory variables or array elements to fields
Syntax	GATHER FROM <array> MEMVAR [FIELDS <field list>] [MEMO]
See Also	APPEND FROM ARRAY, COPY TO ARRAY, DIMENSION, SCATTER

Description GATHER stores data from a memory variable array or a set of memory variables to the current record of the active table/.DBF.

Clauses **FROM <array>**

If data is being copied from an array, the elements of the array are copied, starting with the first element, into the corresponding fields of the record. The contents of the first array element is copied to the first field of the record; the contents of the second array element is copied to the second field and so on.

If the array has fewer elements than the table/.DBF has fields, the additional fields are ignored. If the array has more elements than the table/.DBF has fields, the additional array elements are ignored.

MEMVAR

If MEMVAR is included, data is transferred to the current record from memory variables with the same names as the fields in the table/.DBF.



You can create memory variables with the same names as fields by including MEMVAR or BLANK in SCATTER.

The contents of each memory variable are copied into the field with the same name as the memory variable. If a memory variable with the same name as a field doesn't exist, the field isn't changed.

FIELDS <field list>

If you include the FIELDS clause, only the fields specified in <field list> are replaced by the contents of the array elements or memory variables.

MEMO

Include MEMO to include memo fields in the GATHER command. If MEMO isn't included, memo fields are skipped when GATHER stores data from an array or memory variables to the field.

Example

The following example illustrates the use of SCATTER and GATHER. SCATTER copies the data in the current record into memory variables, which are displayed in a window where they can be edited. When the last field is exited, the data is copied into the current record with GATHER. The TO clause isn't used with SCATTER and the FROM clause isn't used with GATHER in this example because you use the TO and FROM clauses only when copying data into and from arrays.

```

CLOSE DATABASES
SET TALK OFF
USE customer
SCATTER MEMVAR
DEFINE WINDOW menter FROM 5,10 TO 20,70 PANEL
ACTIVATE WINDOW menter
@ 1,3   SAY 'Customer: '   GET m.company
@ 3,3   SAY 'Address: '   GET m.address
@ 5,3   SAY 'City: '      GET m.city
@ 7,3   SAY 'State: '     GET m.state
@ 7,18  SAY 'Zip: '       GET m.zip
@ 13,13 SAY 'Press <Escape> to discard changes'
READ

IF LASTKEY( ) != 27
    GATHER MEMVAR
ENDIF
DEACTIVATE WINDOW menter
USE

```

GETBAR()

Purpose	Returns the number of bars on a popup defined with DEFINE POPUP or a FoxPro system menu popup
Syntax	GETBAR(<expC>, <expN>)
Parameters	<expC> The name of the popup <expN> The position of the bar on the popup
Returns	Numeric
See Also	DEFINE BAR, DEFINE POPUP, RELEASE BAR

Description Use GETBAR() to determine the position of an option on a popup. This function is useful when options on a popup are added, removed or rearranged. Use DEFINE BAR to add an option to a popup or RELEASE BAR to remove an option. Popup options can be rearranged interactively if MOVER is included when the popup is created with DEFINE POPUP.

Parameters **<expC>, <expN>**
Specify the popup name with <expC> and its position within the popup with <expN>. GETBAR() returns the bar number of the option in popup position <expN>. <expN> can range from 1 through the number of rows within the popup. 1 corresponds to the first row on the popup, 2 to the second row and so on.

Example The following example creates a popup named DEMOPOP. The MOVER key word is included so the options in the popup can be rearranged. (See MOVER in DEFINE POPUP in this chapter for information about rearranging popup options.)

The popup is activated, and a series of GETBAR() functions return the prompts for each option. After you rearrange the options, press the F5 key to display the new option order.

```
CLEAR
ON KEY LABEL F5 DO showorder
WAIT WINDOW "Press F5 to refresh." NOWAIT
DEFINE POPUP demopop MOVER FROM 2,2
DEFINE BAR 1 OF demopop PROMPT 'One'
DEFINE BAR 2 OF demopop PROMPT 'Two'
DEFINE BAR 3 OF demopop PROMPT 'Three'
DEFINE BAR 4 OF demopop PROMPT 'Four'
DO showorder
ACTIVATE POPUP demopop
```

```
PROCEDURE showorder
CLEAR
@ 3,12 SAY '1 ' + PRMBAR('demopop', GETBAR('demopop',1))
@ 4,12 SAY '2 ' + PRMBAR('demopop', GETBAR('demopop',2))
@ 5,12 SAY '3 ' + PRMBAR('demopop', GETBAR('demopop',3))
@ 6,12 SAY '4 ' + PRMBAR('demopop', GETBAR('demopop',4))
RETURN
```

GETDIR()

Purpose	Displays the Select Directory dialog
Syntax	GETDIR([<expC1> [, <expC2>]])
Parameters	<expC1> The directory initially selected <expC2> The prompt displayed at the top of the dialog
Returns	Character
See Also	GETEXPR

Description GETDIR() displays the Select Directory dialog from which a directory can be chosen. GETDIR() returns the name of the directory you choose as a character string.

If you do not choose a directory (you choose **Cancel** or press Escape), GETDIR() returns the null string.

Parameters **<expC1>**

Use <expC1> to specify the directory that is initially displayed in the Select Directory dialog. When <expC1> is not specified, the Select Directory dialog opens with the default directory displayed.

<expC2>

The prompt for the directory list in the Select Directory dialog is specified with <expC2>.

GETENV()

Purpose	Returns the contents of the specified MS-DOS environment variable
Syntax	GETENV(<expC>)
Returns	Character
See Also	DISKSPACE(), OS(), VERSION()

Description Two environment variables are always available — COMSPEC and PATH. You can create your own environment variables with the MS-DOS SET command.

See your MS-DOS manual for additional information on creating environment variables.

Parameter <expC>

The name of the MS-DOS environment variable is specified with <expC>. The null string is returned if the MS-DOS environment variable doesn't exist.

In FoxPro for Windows you can locate the Windows directory with the WINDIR environmental variable which is set by Windows when it starts.

Example

```
? GETENV('PATH')  
C:\COMMAND;C:\FOXPRO25;C:\FOXPROW
```

GETEXPR

Purpose	Invokes the Expression Builder dialog and stores the user-constructed expression to a memory or array element
Syntax	GETEXPR [<expC1>] TO <memvar> [TYPE <expC2> [; <expC3>]] [DEFAULT <expC4>]
See Also	GETFILE(), LOCFILE(), PUTFILE()

Description GETEXPR displays the Expression Builder so you can create an expression and store the expression to a memory variable or array element.

Clauses **<expC1>**
<expC1> is the prompt that appears in the Expression Builder. The prompt can be used to remind you what kind of expression to build.

TO <memvar>
The expression you create is stored as a character string to the memory variable or array element specified with **<memvar>**. If the memory variable doesn't already exist, it is created. GETEXPR won't create an array element.

If you exit the Expression Builder by pressing Escape or by choosing **Cancel**, the null string is stored to the memory variable or array element. If a default expression is created with the **DEFAULT** clause, the default expression is stored to the memory variable if you exit the Expression Builder by pressing Escape or choosing **Cancel**.

TYPE <expC2> [; **<expC3>****]**
<expC2> specifies the expression type: C for character, N for numeric, D for date and L for logical.

You can specify the error message **<expC3>** to be displayed if the expression isn't valid. If **<expC3>** is included with **<expC2>**, **<expC2>** and **<expC3>** must be separated by a semicolon (;). The combination of **<expC2>**, the semicolon and **<expC3>** must be enclosed in quotes.

DEFAULT <expC4>

The **DEFAULT** clause lets you display the initial default expression <expC4> in the Expression Builder. You can accept the default expression or overwrite it with your own expression. If the **DEFAULT** clause is included, <expC4> is stored to the memory variable or array element if you exit the Expression Builder by pressing **Escape** or choosing **Cancel**.

Example

In this example, **GETEXPR** is used to get a **LOCATE** expression of the proper type. If **LOCATE** is successful, the company name is displayed; otherwise, a message is displayed.

```

CLOSE DATABASES
USE customer
GETEXPR 'Enter condition to locate ' TO temp;
      TYPE 'L' DEFAULT 'COMPANY = " "'
LOCATE FOR &temp
IF FOUND( )
    ? 'Company: ' + company
ELSE
    ? 'Condition ' + temp + ' was not found '
ENDIF

```

GETFILE()

Purpose	Displays the Open dialog and returns the name of the file you chose
Syntax	GETFILE([<expC1>] [, <expC2>] [, <expC3>] [, <expN>])
Parameters	<expC1> <expC2> <expC3> <expN> The file name extensions of files displayed in the dialog The prompt displayed at the top of the dialog The text in the Open push button Specifies the number and type of push buttons in the dialog
Returns	Character
See Also	FULLPATH(), GETEXPR, LOCFILE(), PUTFILE()

Description GETFILE() displays the FoxPro Open dialog from which a file anywhere on the disk can be chosen. GETFILE() returns the name of the file you choose and returns the null string if you press Escape or choose **Cancel**.

Parameters **<expC1>**
The character expression <expC1> specifies the file extensions of the files displayed in the scrollable list when the All Files check box isn't checked. For a complete list of the file name extensions used by FoxPro, see the file name extension table in the FoxPro for Windows *Developer's Guide* or in the FoxPro for MS-DOS *Update* manual.

<expC1> can take a variety of forms:

- If <expC1> contains a single extension (for example, .PRG), only files with that extension are displayed.
- <expC1> can contain a list of file extensions separated by semicolons. For example, including PRG;FXP displays all files with the extension .PRG or .FXP.

If the files have the same name but different extensions (for example, CUSTOMER.PRG and CUSTOMER.FXP), only the file with the extension listed first in <expC1> is displayed.

- <expC1> can contain a list of file extensions separated by vertical bars (for example, PRG|FXP). In this case, all files with these extensions are displayed, even if the files have the same names.

- If <expC1> contains just a semicolon (;), all files without an extension are displayed.
- If <expC1> is the null string, all files in the current directory are displayed.
- <expC1> can also contain MS-DOS wildcard characters (* and ?). All files with extensions that meet the wildcard criteria are displayed. For example, if <expC1> is ?X?, all files with the extension .FXP, .EXE or .TXT are displayed.

<expC2>

The character expression <expC2> is the prompt displayed at the top of the Open dialog.

<expC3>

The character expression <expC3> replaces the text in the **Open** push button.

<expN>

The numeric expression <expN> specifies the number and type of push buttons that appear in the Open dialog. The following buttons appear in the dialog when <expN> is 0 (or is omitted), 1 or 2:

<expN>	Push Buttons FoxPro for MS-DOS	Push Buttons FoxPro for Windows
0 (or omitted)	Open Cancel	Open Cancel
1	Open New Cancel	New Open Cancel
2	Open None Cancel	None Open Cancel

The null string is returned if <expN> is 1 or 2 and the **New** or **None** push button is chosen.

GETFILE()

Example

```
CLOSE DATABASES
SELECT 0
dbf_file = GETFILE('DBF', 'Browse or Create a .DBF:', 'Browse',1)

DO CASE
    CASE 'Untitled' $ dbf_file
        CREATE (dbf_file)
    CASE EMPTY(dbf_file)
        RETURN
    OTHERWISE
        USE (dbf_file)
        BROWSE
ENDCASE
```

GETFONT()

Purpose	Displays the FoxPro for Windows Font dialog and returns the name of the font you choose
Syntax	GETFONT()
Returns	Character
Remarks	FoxPro for Windows only
See Also	FONTMETRIC(), TXTWIDTH(), SYSMETRIC(), WFONT()

Description GETFONT() displays the FoxPro for Windows Font dialog, from which an installed font can be chosen. GETFONT() returns the name, size and style of the font you choose. Your choice is returned as a character string with the font name, size and style separated by commas.

GETFONT() returns the null string if you choose **Cancel**, **Close** from the **Control** menu or press Escape.



FoxPro commands and functions can be abbreviated to four characters. In the case of GETFONT() and GETFILE(), which both begin with the same four letters, precedence is given to GETFILE(). Issuing GETF() displays the Open dialog.

GETPAD()

Purpose	Returns the name of a menu pad from its position on a user-defined menu bar or on the FoxPro system menu bar
Syntax	GETPAD(<expC>, <expN>)
Parameters	<expC> The menu name <expN> The position of the menu pad on the menu bar
Returns	Character
See Also	DEFINE MENU, DEFINE PAD, RELEASE PAD

Description GETPAD() lets you determine the position of a pad on a menu bar. Pads on a menu bar may be added, removed or rearranged. Use DEFINE PAD to add a pad to a menu bar or RELEASE PAD to remove a pad.

Parameters <expC>, <expN>
Specify the menu name with <expC> and the numeric pad position within the menu bar with <expN>. GETPAD() returns the pad name for the menu bar position specified in <expN>, which can range from 1 through the number of pads on the menu bar. 1 corresponds to the first pad on the menu bar, 2 to the second pad and so on.

Example The following program uses GETPAD() to test if the Edit menu pad is on the FoxPro system menu bar. If it is, the pad is released. (To restore the Edit menu bar to its default state, issue the command SET SYSMENU TO DEFAULT.)

```
FOR count = 1 TO CNTPAD('_msysmenu')  && Number of pads
  IF PRMPAD('_msysmenu', GETPAD('_msysmenu', count)) = 'Edit'
    RELEASE PAD (GETPAD('_msysmenu', count)) OF _msysmenu
  EXIT
ENDIF
ENDFOR
```

GO/GOTO

Purpose	Positions the record pointer on the specified record number in a table/.DBF
Syntax	GO [RECORD] <expN1> [IN <expN2> IN <expC>] GO TOP BOTTOM [IN <expN2> IN <expC>] or GOTO [RECORD] <expN1> [IN <expN2> IN <expC>] GOTO TOP BOTTOM [IN <expN2> IN <expC>]
See Also	RECNO(), SELECT, SKIP
Description	GO and GOTO can be used interchangeably. These commands operate on the table/.DBF in the current work area unless you specify another work area with the IN clause.
Clauses	[RECORD] <expN1> Moves the record pointer to the physical record number <expN1>. Note that you can leave out GO or GOTO entirely — you can specify just the record number. If you specify just the record number, you can move the record pointer only within the current work area. IN <expN2> IN <expC> By default, GO or GOTO moves the record pointer in the table/.DBF open in the current work area. To move the record pointer in a table/.DBF in another work area, include the work area number <expN> or the table/.DBF alias <expC>. TOP BOTTOM GO TOP or GOTO TOP and GO BOTTOM or GOTO BOTTOM position the record pointer on the first and last records, respectively, in the current table/.DBF. If the table/.DBF has an ascending index in use, the first record is the record with the lowest key value, and the last record is the record with the highest key value. If the index is in descending order, the first record is the record with the highest key value, and the last record is the record with the lowest key value.

Example

```
CLOSE DATABASES
USE invoices
USE customer IN 0
GO BOTTOM IN invoices
? RECNO('invoices')
```

54

```
GO BOTTOM
? RECNO( )
```

51

```
GO 5
? RECNO( )
```

5

GOMONTH()

Purpose	Returns the date that is a specified number of months before or after a given date
Syntax	GOMONTH(<expD>, <expN>)
Parameters	<expD> A date expression <expN> The number of months before/after the specified date
Returns	Date
See Also	CMONTH()

Description GOMONTH() returns the date that is a number of months after or before a given date.

Parameters **<expD>**
 <expD> is a date expression.

<expN>
 <expN> is the number of months from the date specified with <expD>. If <expN> is positive, the date returned by GOMONTH() is <expN> months after <expD>. If <expN> is negative, the date returned is <expN> months before <expD>.

Examples

```
SET CENTURY ON
STORE GOMONTH({08/02/92}, 5) TO deadline
? deadline
01/02/1993

? GOMONTH({12/31/92}, 2)
02/28/1993

? GOMONTH({12/31/92}, -2)
10/31/1992
```

HEADER()

Purpose	Returns the number of bytes in the header of the current or specified table/.DBF file
Syntax	HEADER([<expN> <expC>])
Parameters	<expN> The work area number <expC> The table/.DBF alias
Returns	Numeric
See Also	FSIZE(), RECSIZE()

Description HEADER() returns the size (in bytes) of a table/.DBF header. A table/.DBF header contains information about the table/.DBF itself, such as the field names and sizes and the presence of a memo file or structural index.

Parameters <expN> | <expC>

If you omit the work area and alias, HEADER() returns the size of the header for the table/.DBF open in the current work area. To return the header size for a table/.DBF open in a specific work area, include the work area number <expN> or the table/.DBF alias <expC>.

HEADER() returns 0 if a table/.DBF isn't open in the work area you specify. If a table/.DBF doesn't have the alias you specify, the message "Alias not found" is displayed.

HELP

Purpose	Invokes the Help window
Syntax	HELP [<topic>] [IN [WINDOW] <window name> IN [WINDOW] SCREEN] [NOWAIT]
See Also	SET HELP, SET HELPFILTER, SET TOPIC
Description	HELP opens the Help window. The main Help window contains a list of help topics. In FoxPro for MS-DOS, you can open the Help window by typing HELP in the Command window, pressing the F1 key or choosing Help... from the System menu. In FoxPro for Windows, you can open the Help window by typing HELP in the Command window, pressing the F1 key or choosing Help... from the Help menu. For additional information on the FoxPro online help facility, refer to the FoxPro <i>User's Guide</i>.
Clauses	<topic> In FoxPro for MS-DOS, to open the help window with a specific help subject already displayed, use <topic> to specify the topic you would like more information about. If you include just a partial spelling of a <topic>, FoxPro opens the Help window and displays the topic with the closest matching title. In FoxPro for Windows, including the <topic> parameter displays the Windows-style Search dialog with the topic selected. If you include just a partial spelling of a topic, FoxPro opens the Search dialog with the closest matching topic selected. IN [WINDOW] <window name> The Help window can be opened from within a parent window, but it doesn't assume the characteristics of the parent window it is placed in. If the Help window is activated inside a parent window, it cannot be moved outside the parent window. If the parent window is moved, the Help window moves with it. Before you can open the Help window from within a parent window, the parent window must first be defined with DEFINE WINDOW.

IN [WINDOW] SCREEN

You can also place the Help window on the desktop in FoxPro for MS-DOS or the main FoxPro window in FoxPro for Windows by including `IN WINDOW SCREEN` or `IN SCREEN`.

NOWAIT

When `HELP` is issued from within a program, the Help window is opened and program execution pauses until the Help window is closed. Including `NOWAIT` continues program execution after the Help window has been opened. The program doesn't wait for the Help window to be closed but continues program execution on the program line immediately following `HELP NOWAIT`.

HIDE MENU

Purpose	Hides one or more active user-defined menu bars
Syntax	HIDE MENU <menu name1> [, <menu name2> ...] ALL [SAVE]
See Also	ACTIVATE MENU, DEFINE MENU, DEFINE PAD, SHOW MENU

Description HIDE MENU removes the specified menu bar, a set of menu bars or all menu bars from the desktop in FoxPro for MS-DOS, the main FoxPro window in FoxPro for Windows or a user-defined window without removing the menu definition from memory. Before it can be hidden, a menu bar must first be created with DEFINE MENU. Hiding a menu bar isn't the same as deactivating it. When a menu bar is hidden, it stays resident in memory and can be displayed on the desktop, the main FoxPro window or a user-defined window with ACTIVATE MENU or SHOW MENU.

Clauses <menu name1> [, <menu name2> ...]
Include the name of the menu bar or a list of menu bars (separated by commas) to hide.

ALL
Include ALL to hide all defined menu bars.

SAVE
You can place an image of a menu bar on the screen or in a window by including SAVE. Placing an image of a menu bar on the screen is useful during program development and testing. Menu bar images can be cleared from the desktop, the main FoxPro window or a user-defined window with CLEAR.

HIDE POPUP

Purpose	Hides one or more active popups defined with DEFINE POPUP
Syntax	HIDE POPUP <popup name1> [, <popup name2> ...] ALL [SAVE]
See Also	ACTIVATE POPUP, DEFINE BAR, DEFINE POPUP, SHOW POPUP

Description HIDE POPUP removes the specified popup, a set of popups or all popups from the desktop in FoxPro for MS-DOS, the main FoxPro window in FoxPro for Windows or a user-defined window without removing the popup definition from memory. Before it can be hidden a popup must first be created with DEFINE POPUP. Hiding a popup isn't the same as deactivating it. When a popup is hidden, it stays resident in memory and can be displayed on the desktop, the main FoxPro window or a user-defined window with ACTIVATE POPUP or SHOW POPUP.

Clauses <popup name1> [, <popup name2> ...]

Include the name of the popup or a list of popups (separated by commas) to hide.

ALL

Include ALL to hide all defined popups.

SAVE

You can place an image of a popup on the desktop, main FoxPro window or a user-defined window by including SAVE. Placing an image of a popup on the screen is useful during program development and testing. Popup images can be cleared from the desktop, the main FoxPro window or a user-defined window with CLEAR.

HIDE WINDOW

Purpose	Hides an active user-defined window or FoxPro system window
Syntax	HIDE WINDOW [<window name1> [, <window name2>] ...] ALL [IN [WINDOW] <window nameN> IN [WINDOW] SCREEN] [SAVE]
See Also	ACTIVATE WINDOW, DEACTIVATE WINDOW, DEFINE WINDOW, RELEASE WINDOW, SHOW WINDOW

Description HIDE WINDOW removes a window or a set of windows from the desktop in FoxPro for MS-DOS, the main FoxPro window in FoxPro for Windows or a user-defined window. You can use HIDE WINDOW to hide system windows, such as the Command window, the Calculator, the Calendar/Diary, and so on.

Hiding a window isn't the same as closing it. When a window is hidden, it stays resident in memory and active. Output can be sent to a hidden window but you cannot see it.

Releasing a window removes it from memory. Windows removed from memory must be defined again to be redisplayed. A window can be displayed with ACTIVATE WINDOW or SHOW WINDOW.

You can also hide a window by choosing **Hide** from the **Window** menu popup. You can redisplay a window by choosing its name from the **Window** menu.

For additional information on hiding windows, see the FoxPro *User's Guide*.

Clauses <window name1> [, <window name2> ...]

Include the name of the window or a list of windows (separated by commas) to hide.

If HIDE WINDOW is issued without any arguments, the active window is hidden.

ALL

Include ALL to hide all windows.

IN [WINDOW] <window nameN>

Windows can be hidden within a parent window. You can specify the parent window within which a child window is hidden by including IN WINDOW. Desk accessories (Filer, the Calculator, and so on) must be active before they can be hidden.

IN [WINDOW] SCREEN

You can also specifically hide a window on the desktop or main FoxPro window by including IN WINDOW SCREEN or IN SCREEN.

SAVE — FoxPro for MS-DOS only

SAVE displays an image of a window on the desktop or in a user-defined window after the window has been hidden. Normally, windows are removed from the desktop or a user-defined window after they are hidden. Use CLEAR to remove the window image from the desktop or a user-defined window. You cannot save an image of the current output window.

SAVE is ignored in FoxPro for Windows.

Example

In this example, a window named Output is defined and activated. The program waits for you to press a key and then hides the window. The program waits for you to press a key again and then displays the window. Then press a key to remove the window from the screen and from memory.

```
DEFINE WINDOW output FROM 6,1 TO 19,75 TITLE 'Output' ;
    CLOSE FLOAT GROW SHADOW ZOOM
ACTIVATE WINDOW output
WAIT WINDOW 'Press a key to hide this window'
HIDE WINDOW output
WAIT WINDOW 'Press a key to see the window again'
SHOW WINDOW output
WAIT WINDOW 'Press a key to remove the window'
DEACTIVATE WINDOW output
RELEASE WINDOW output
```


#IF ... #ENDIF

Purpose	Conditionally includes source code at compile time
Syntax	<pre>#IF <expN1> #IF <expL1> <statements> [#ELIF <expN2> #ELIF <expL2> <statements> ... #ELIF <expNn> #ELIF <expLn> <statements>] [#ELSE <statements>] #ENDIF</pre>
See Also	COMPILE, #DEFINE ... #UNDEF

Description The #IF ... #ENDIF preprocessor directives let you conditionally include source code in a compiled program. #IF ... #ENDIF can improve the readability of source code, reduce compiled program size and improve performance in some cases.

When the #IF ... #ENDIF structure is compiled, successive logical or numeric expressions within the structure are evaluated; the evaluation results determine which set of FoxPro statements (if any) are included in the compiled code.

Clauses

```
#IF <expN1> | #IF <expL1>
    <statements>
```

The numeric expression <expN1> or logical expression <expL1> is evaluated. If the expression is numeric and non-zero or the expression is logical and true (.T.), the statements immediately following #IF are included in the compiled code. The #IF ... #ENDIF structure is exited, and the first program line following #ENDIF is then compiled.

If the expression following #IF is numeric and 0 or logical and false (.F.), the statements immediately following #IF are not included in the compiled code. In this case, any following #ELIF directives are evaluated.

#IF ... #ENDIF

```
[#ELIF <expN2> | #ELIF <expL2>
  <statements>
...
#ELIF <expNn> | #ELIF <expLn>
  <statements>]
```

If <expN1> is 0 or <expL1> is false, the #ELIF directives are then evaluated. The first #ELIF expression <expN2> or <expL2>, if present, is evaluated. If <expN2> is non-zero or <expL2> is true, the statements immediately following #ELIF are included in the compiled code. The #IF ... #ENDIF structure is exited, and the first program line following #ENDIF is then compiled.

If <expN2> is 0 or <expL2> is false, the statements following #ELIF are not included in the compiled code, and the next #ELIF directive is evaluated.

```
[#ELSE
  <statements>]
```

If no #ELIF directives are included or if those that are included evaluate to 0 or false, the presence or absence of #ELSE determines whether any additional statements are included in the compiled code:

- If #ELSE is included, the statements following #ELSE are included in the compiled code.
- If #ELSE isn't included, none of the program statements between #IF and #ENDIF is included in the compiled code. The #IF ... #ENDIF structure is exited, and compilation continues on the first program line following #ENDIF.

Example

In the following example, the #IF ... #ENDIF structure determines which version of FoxPro compiles the program and then displays the appropriate message.

```
#IF 'WINDOWS' $ UPPER(VERSION( ))
  ? 'This is FoxPro for Windows'
#ELIF 'MAC' $ UPPER(VERSION( ))
  ? 'This is FoxPro for Mac'
#ELIF 'UNIX' $ UPPER(VERSION( ))
  ? 'This is FoxPro for UNIX'
#ELSE
  ? 'This is FoxPro for DOS'
#ENDIF
```

IF ... ENDIF

Purpose	Conditionally executes a set of commands based on the outcome of a logical expression
Syntax	IF <expL> <statements> [ELSE <statements>] ENDIF
See Also	DO CASE, DO WHILE, FOR ... ENDFOR, IIF(), SCAN

Description In this structured programming command, the logical expression <expL> is evaluated. If <expL> evaluates to true (.T.), any statements following IF and prior to ELSE or ENDIF (whichever occurs first) are executed.

If <expL> is false and ELSE is included, any statements after ELSE and before ENDIF are executed.

If <expL> is false and ELSE isn't included, all statements between IF and ENDIF are ignored. In this case, program execution continues with the first command following ENDIF.

You can nest an IF ... ENDIF block within another IF ... ENDIF block.

Comments can be placed on the same line after IF, ELSE and ENDIF. These comments are ignored during compilation and program execution.

Example

```
CLOSE DATABASES
USE customer
GETEXPR 'Enter condition to locate ' TO temp;
    TYPE 'L' DEFAULT 'COMPANY = ""'
LOCATE FOR &temp                && Enter LOCATE expression
IF FOUND( )                    && Was it found?
    ? 'Company: ' + company    && If so display a message
ELSE                            && If not found
    ? 'Condition ' + temp + ' was not found ' && Display a message
ENDIF
USE
```

IIF()

Purpose	Returns one of two values depending on the value of a logical expression
Syntax	IIF(<expL>, <expr1>, <expr2>)
Parameters	<expL> The logical expression to be evaluated <expr1> The expression to be returned if <expL> is true <expr2> The expression to be returned if <expL> is false
Returns	Character, Date, Logical or Numeric
See Also	IF ... ENDIF

Description This function, also known as Immediate IF, evaluates a logical expression and then returns one of two expressions. If the logical expression evaluates to a true value (.T.), IIF() returns the first expression. If the logical expression evaluates to a false value (.F.), IIF() returns the second expression.

Parameters **<expL>**
The logical expression that IIF() evaluates is specified with <expL>.

<expr1>, <expr2>
If <expL> evaluates to true (.T.), <expr1> is returned. If <expL> evaluates to false (.F.), <expr2> is returned.



This function can be used in place of IF ... ENDIF for simple conditional expressions and is especially useful in report and label expressions that conditionally specify field contents. The IIF() function also executes considerably faster than an equivalent IF ... ENDIF.

Example This example uses IIF() to check if the NOTES field in the SALESMAN table/DBF is empty. If it is empty, “No comments” is displayed; otherwise, the contents of the memo field are displayed.

```
CLOSE DATABASES
USE salesman
SCAN
    ? IIF(EMPTY(notes), 'No notes', notes) && Empty memo field?
ENDSCAN
```

IMPORT

Purpose	Imports data from an external file format to create a new FoxPro table/.DBF
Syntax	IMPORT FROM <file> [TYPE] FW2 MOD PDOX RPD WK1 WK3 WKS WR1 WRK XLS
See Also	APPEND FROM, COPY TO, EXPORT

Description Most software packages store their data in a file format that cannot be opened directly in FoxPro. **IMPORT** creates a new FoxPro table/.DBF from data stored in file formats that FoxPro cannot read.

A new table/.DBF is created with the same name as the file the data is imported from. A .DBF extension is assigned to the newly created table/.DBF.

Clauses **<file>**
<file> is the name of the file to import data from. If you don't include an extension with the file name, the default extension for the specified file type is assumed.

TYPE

The key word **TYPE** is optional, but you must include one of the following file types described below.

FW2

Include **FW2** to import **FW2** files, created by Framework II.

MOD

Include **MOD** to import **MOD** files, created by Microsoft Multiplan version 4.01.

PDOX

Include **PDOX** to import Paradox files. Database files in Paradox versions 3.5 and 4.0 by Borland can be imported by including the **PDOX** option.

RPD

Include **RPD** to import **RPD** files, created by RapidFile.

WK1 | WK3 | WKS

Include WK1 to import data from a Lotus 1-2-3 spreadsheet. Columns from the spreadsheet become fields in the table/.DBF, and the spreadsheet rows become records in the table/.DBF. A WK1 extension is assigned to spreadsheets created in Lotus 1-2-3 revision 2.x; a WK3 extension is assigned to spreadsheets created in Lotus 1-2-3 revision 3.x; and a .WKS extension is assigned to spreadsheets created in Lotus 1-2-3, revision 1-A.

WR1 | WRK

Include WR1 to import data from a Lotus Symphony spreadsheet. Columns from the spreadsheet become fields in the table/.DBF, and the spreadsheet rows become records in the table/.DBF. A WR1 extension is assigned to spreadsheets created in Symphony version 1.10, and a .WRK extension is assigned to spreadsheets created in Symphony version 1.01.

XLS

Include XLS to import data from Microsoft Excel spreadsheets versions 2.0, 3.0 and 4.0. Columns from the spreadsheet become fields in the table/.DBF, and the spreadsheet rows become records in the table/.DBF. Spreadsheet files created in Microsoft Excel have an .XLS extension.

INDEX

Purpose	Creates an index file to display table/.DBF records in logical order
Syntax	INDEX ON <expr> TO <idx file> TAG <tag name> [OF <cdx file>] [FOR <expl>] [COMPACT] [ASCENDING DESCENDING] [UNIQUE] [ADDITIVE]
See Also	COPY INDEX, COPY TAG, REINDEX, SET INDEX, SET ODOMETER, SET ORDER, SET TALK, SET UNIQUE, SORT, USE

Description INDEX is used to create an INDEX file or tag for the current table/.DBF. Records in a table/.DBF that has an index file are displayed and accessed in the order specified by the index expression. The physical order of the records in the table/.DBF isn't changed by an index file.

If TALK is SET ON, FoxPro reports how many records are indexed during the indexing process. The record interval displayed during indexing can be specified with SET ODOMETER.

DISPLAY STATUS gives extensive information about open index files. This information includes the names of all open index files, their types (structural, .CDX, .IDX), their index expressions and the name of the master index file or master tag.

The number of index files (.IDX or .CDX) you can open is limited only by memory and system resources. The number of files you can open is determined by the FILES setting in the MS-DOS CONFIG.SYS configuration file. See your MS-DOS manual for more information on the MS-DOS FILES setting.

Index Types

FoxPro lets you create two types of index files:

- Compound .CDX index files containing multiple index entries called tags
- .IDX index files containing one index entry

You can also create a structural compound index file, which is automatically opened with the table/.DBF.



Because structural compound index files are automatically opened when the table/.DBF is opened, they are the preferred index type.

Include **COMPACT** to create compact .IDX index files. Compound index files are always compact.

If you are sharing files between FoxPro and FoxBASE+ or FoxBASE+/Mac, you must use non-compact .IDX index files. Otherwise, if you build .IDX files, include **COMPACT** to benefit from FoxPro fast-access index technology.

Index Order and Updating

Only one index file (the master index file) or tag (the master tag) controls the order in which the table/.DBF is displayed or accessed. Certain commands (**SEEK**, for example) use the master index file or tag to search for records. However, all open .IDX and .CDX index files are updated as changes are made to the table/.DBF. You can designate the master index file or tag with the **INDEX** clause of **USE** or with **SET INDEX** and **SET ORDER**. See each of these commands in this chapter for additional information on setting a master index file or master tag.

User-Defined Functions

The index expression can contain a user-defined function (UDF).



Care should be exercised when referencing a UDF in an index expression. Referencing a user-defined function in an index expression increases the time it takes to create the index.

For example, a UDF can be used to extract the street name from a single field that includes both the street number and street name. Assuming the street number is always numeric, the UDF can return the character portion of the field and pad it with spaces as needed to create a constant-length index key. Records in the table/.DBF can then be accessed in street name order.

FoxPro must be able to locate the UDF included in an index expression to update the index. When FoxPro creates an index, the index expression is saved in the index file, but only a *reference* to the UDF is included in the index expression.

Clauses**<expr>**

The index expression <expr> can include the name of a field or fields from the current table/.DBF. An index key based on the index expression is created in the index file for each record in the table/.DBF. FoxPro uses these keys to display and access records in the table/.DBF.

Warnings

Although not recommended, <expr> can also be a memory variable, an array element, or a field or field expression from a table/.DBF in another work area. Memo fields cannot be used alone in index file expressions; they must be combined with other character expressions. If you access an index that contains a variable or field that no longer exists or cannot be located, the error message "Variable not found" is displayed.

The message "Warning: Index expression contains aliased fields" is displayed if a field you include in the index expression is prefaced by a table/.DBF alias or work area letter. FOR clauses are optimized with Rushmore technology if aliased fields are included. However, it is still highly recommended that you avoid using aliased fields when creating indexes. In several cases (USE ... AGAIN, SQL queries, and so on), a different alias is automatically assigned to a table/.DBF, and the index may not be properly updated or used. Refer to the discussion of Rushmore technology in the *Optimizing Your Application* chapter in the *FoxPro Developer's Guide*.

If you attempt to build an index with a key that varies in length, the key will be padded with spaces. Variable-length index keys aren't supported in FoxPro.

TO <idx file>

You can create an .IDX index file by including the TO <idx file> clause. The index file is given the default extension .IDX, which can be overridden by explicitly including a different extension or by changing the default index extension in the FoxPro configuration file. Standard MS-DOS rules for naming files must be observed when creating index files.

TAG <tag name> [OF <cdx file>]

You can create a compound index file by including TAG <tag name>. A compound index file is a single-index file that can consist of any number of separate tags (index entries). Each tag is identified by its unique tag name. Tag names, like memory variables, must begin with a letter or an underscore and can consist of any combination of up to 10 letters, digits or underscores. The number of tags in a compound index file is limited only by available memory and disk space.

Multiple-entry compound index files are always compact. It isn't necessary to include COMPACT when creating a compound index file. Names of compound index files are given a .CDX extension.

Two types of compound index files can be created. The first type, a *structural* compound index file, is created when you include TAG <tag name> without including the optional OF <cdx file> clause. A structural compound index file *always* has the same base name as the table/.DBF and is automatically opened when the table/.DBF is opened.

If a table/.DBF's structural compound index file cannot be located or is deleted or renamed, a dialog is displayed when you try to open the table/.DBF. If you choose the default **Cancel** push button, the table/.DBF isn't opened. Choosing **Ignore** opens the table/.DBF and removes the flag in the table/.DBF's header that indicates an associated structural compound index file is present.

To reassociate a structural compound index that has become dissociated from its table/.DBF, issue the following command:

```
USE <filename> INDEX <filename>
```

The next time you open the table/.DBF, the structural compound index is opened too. Be sure to reindex the table/.DBF if it has been modified since the structural compound index was dissociated.

You can create a *non-structural* compound index file by including OF <cdx file> after TAG <tag name>. Unlike a structural compound index file, a non-structural compound index file must be explicitly opened with SET INDEX or the INDEX clause in USE.

If a compound index file has already been created and opened, issuing INDEX with TAG <tag name> adds a tag to the compound index file. If a compound index file hasn't already been created, one is automatically created.



FOR <expL>

The FOR clause can be included to make an index file act as a filter for the table/.DBF. Only records that satisfy the filter expression <expL> are available for display and access; index keys are created in the index file for just those records matching the filter expression.

Rushmore optimizes an INDEX ... FOR <expL1> command if <expL1> is an optimizable expression. For best performance, use an optimizable expression in the FOR clause. A discussion of Rushmore optimizable expressions appears in the Optimizing Your Application chapter in the FoxPro *Developer's Guide*.

COMPACT

COMPACT creates a compact .IDX file.

ASCENDING | DESCENDING

.CDX files can be built in ascending or descending order. By default, .CDX tags are created in ascending order (you can include ASCENDING as a reminder of the index file's order). A table/.DBF can be indexed in reverse order by including DESCENDING.

You can't include DESCENDING when creating .IDX index files. You can, however, specify a descending order for an .IDX index file with SET INDEX and SET ORDER.

UNIQUE

Use UNIQUE to specify that only the first record encountered with a particular index key value is included in an .IDX file or a .CDX tag. UNIQUE can be used to prevent the display of or access to duplicate records. All records added with duplicate index keys are excluded from the index file. Using the UNIQUE option of INDEX is identical to executing SET UNIQUE ON before issuing INDEX or REINDEX.

When a UNIQUE index or index tag is active and a duplicate record is changed in a manner that changes its index key, the index or index tag is updated. However, the next duplicate record with the ORIGINAL index key cannot be accessed or displayed until you reindex the file using REINDEX.

ADDITIVE

When you create an index file or files for a table/.DBF with INDEX, any previously opened index files (except the structural compound index) are closed. If the ADDITIVE key word is included, previously opened index files remain open.

Examples This example opens CUSTOMER.DBF and creates an index file named COMPLIST, which displays and processes records in the alphabetic order of the COMPANY field.

```
CLOSE DATABASES
USE customer
INDEX ON company TO complist
51 records indexed
```

In the next example, CUSTOMER.DBF is again opened and an index file named CITYCOMP is created from a substring of the first five characters of the CITY field and the first six characters of the COMPANY field. When this index file is used, records in the table/DBF are ordered according to the CITY field and, secondarily, according to the COMPANY field.

```
CLOSE DATABASES
USE customer
INDEX ON SUBSTR(city,1,5) + SUBSTR(company,1,6) TO citycomp
51 records indexed
```

Here, a filtered index file is created. INVOICES.DBF is opened. The table/DBF is indexed on the IDATE field, but only records that have an amount greater than 500 are available for processing and display.

```
CLOSE DATABASES
USE invoices
INDEX ON idate FOR itotal > 500 TO bigbalnc
164 records indexed
```

Index tags are created in this example. The first tag is a structural compound index tag for ADDRESS1. The second tag is created in a non-structural index file named CUSTCDX. The last tag is also created in the non-structural CUSTCDX index file but in descending order.

```
CLOSE DATABASES
USE customer
INDEX ON address TAG address
INDEX ON company TAG company OF custcdx
```

INKEY()

Purpose	Returns a number corresponding to the first key pressed or mouse click in the typeahead buffer
Syntax	INKEY([<expN>] [, <expC>])
Parameters	<expN> The number of seconds INKEY waits for a key press or a mouse click <expC> If S, INKEY shows the cursor If H, INKEY hides the cursor If M, INKEY checks for a mouse click or key press If E, INKEY expands a keyboard macro
Returns	Numeric
See Also	_DBLCLICK, LASTKEY(), ON KEY, READKEY(), SET TYPEAHEAD

Description INKEY() returns a number corresponding to a key press. It returns 0 if a key isn't pressed. If there are several keys in the typeahead buffer, the value of the first key entered in the buffer is returned.

Parameters <expN>

<expN> specifies how many seconds INKEY() waits for a keystroke. If <expN> isn't included, INKEY() immediately returns a value for a keystroke. INKEY() waits indefinitely for a keystroke if <expN> is 0.

<expC>

Include <expC> to show or hide the cursor or check for a mouse click. To show the cursor, include S in <expC>. To hide the cursor, include H in <expC>. If both S and H are included in <expC>, the last character in <expC> takes precedence.

By default, INKEY() doesn't detect a mouse click. To check for a mouse click, include M in <expC>. If M is included in <expC>, INKEY() returns the value 151 for a single mouse click. Refer to the second example below to see how you can check for a double click.

To check for a mouse click and show the cursor, include both M and S. To check for a mouse click and hide the cursor, include H and M.

When a keyboard macro is assigned to a key or key combination, you can include E in <expC> to expand the keyboard macro. When E is included, INKEY() returns a value corresponding to the first keystroke assigned to the keyboard macro. You can return successive values for each keystroke in a macro by repeatedly executing INKEY() with E included. If E is not included, INKEY() returns the value for the key or key combination that triggers the keyboard macro.

Any characters besides H, M, S and E in <expC> are ignored.

Examples

In this example, a data entry window is displayed, and FoxPro waits for 20 seconds for keyboard input. If a key isn't pressed or the mouse isn't clicked, the program is cancelled.

```
SET TALK OFF
STORE 0 TO holdkey
STORE SPACE(40) TO mcustomer, maddress
STORE SPACE(24) TO mcity
STORE SPACE(2) TO mstate
STORE SPACE(10) TO mzip
DEFINE WINDOW menter FROM 7,10 TO 17,70 PANEL
ACTIVATE WINDOW menter
@ 1,3 SAY 'Customer: ' GET mcustomer
@ 3,3 SAY 'Address: ' GET maddress
@ 5,3 SAY 'City: ' GET mcity
@ 7,3 SAY 'State: ' GET mstate
@ 7,18 SAY 'Zip: ' GET mzip
@ 1,14 SAY ''
STORE INKEY(20, 'M') TO mkey
IF mkey = 0
    DEACTIVATE WINDOW menter
    CLEAR GETS
    CLOSE ALL
ELSE
    IF mkey # 151
        KEYBOARD CHR(mkey)
    ENDIF
    READ
    DEACTIVATE WINDOW menter
ENDIF

*** Mouse click and keyboard detection example ***
*** Establish the program environment ***
SET TALK OFF
SET ESCAPE OFF
CLEAR
```

```

*** Create and open a window ***
DEFINE WINDOW mousechk FROM 4,4 TO 20,75 DOUBLE
ACTIVATE WINDOW mousechk
@ 6,16 SAY 'Press a key or click the mouse to start'
@ 8,16 SAY '          or press the Escape key to quit'
inval1 = 0                      && Initialize inkey memory variable
*** Main program loop ***
DO WHILE .T.
    inval1 = INKEY(.06,'HM') && Get INKEY( ), hide cursor, checkmouse
    IF inval1 = 0              && No key or mouse press
        LOOP
    ENDIF
    IF inval1 = 27              && Escape key pressed, exit main loop
        EXIT
    ENDIF
    IF inval1 = 151              && Single mouse click
        row = MROW( )           && Get mouse pointer position
        col = MCOL( )
        timelimit = SECONDS( ) + _DBLCLICK && Time for double click
        inval2 = 0
        *** Check for mouse click and mouse pointer inside window
        DO WHILE SECONDS( ) < timelimit
            inval2 = INKEY('HM')
            *** double mouse click
            IF inval2 = 151 AND col != -1 AND row != -1
                CLEAR
                @ 7,11 SAY 'Double click at '
                ?? 'Row ' + ALLTRIM(STR(row)) + ', Column ' ;
                + ALLTRIM(STR(col)) + ' of this window'
                EXIT
            ENDIF
        ENDDO
        *** Single mouse click and mouse pointer inside window ***
        IF inval2 = 0 AND col != -1 AND row != -1
            CLEAR
            @ 7,11 SAY 'Single click at '
            ?? 'Row ' + ALLTRIM(STR(row)) + ', Column ' ;
            + ALLTRIM(STR(col)) + ' of this window'
        ENDIF
        *** Mouse pointer outside of window ***
        IF col = -1 OR row = -1
            CLEAR
            @ 7,17 SAY 'Mouse clicked outside of this window'
            LOOP
        ENDIF
    ENDIF
ENDIF

```

INKEY()

```
*** Keyboard character pressed ***
IF inval1 # 151 AND BETWEEN(inval1, 0, 256) && Good CHR( ) value
  CLEAR
  @ 7,18 SAY 'Character typed:'
  ?? ' ' + CHR(inval1) + ', ASCII code ' + ALLTRIM(STR(inval1))
ENDIF
ENDDO      End of Main loop
*** Restore original environment ***
SET TALK ON
SET ESCAPE ON
CLEAR WINDOWS
```

INKEY() Return Values				
Key	Alone	Shift	Ctrl	Alt
F1	28	84	94	104
F2	-1	85	95	105
F3	-2	86	96	106
F4	-3	87	97	107
F5	-4	88	98	108
F6	-5	89	99	109
F7	-6	90	100	110
F8	-7	91	101	111
F9	-8	92	102	112
F10	-9	93	103	113
F11	133	136	137	139
F12	134	136	138	140
1	49	33	—	120
2	50	64	33	121
3	51	35	—	122
4	52	36	—	123
5	53	37	—	124
6	54	94	30	125
7	55	38	—	126
8	56	42	—	127
9	57	40	—	128
0	48	41	—	19

INKEY() Return Values				
Key	Alone	Shift	Ctrl	Alt
a	97	65	1	30
b	98	66	2	48
c	99	67	3	46
d	100	68	4	32
e	101	69	5	18
f	102	70	6	33
g	103	71	7	34
h	104	72	127	35
i	105	73	9	23
j	106	74	10	36
k	107	75	11	37
l	108	76	12	38
m	109	77	13	50
n	110	78	14	49
o	111	79	15	24
p	112	80	16	25
q	113	81	17	16
r	114	82	18	19
s	115	83	19	31
t	116	84	20	20
u	117	85	21	22
v	118	86	22	47
w	119	87	23	17
x	120	88	24	45
y	121	89	25	21
z	122	90	26	44
INS	22	22	146	162
HOME	1	1	29	151
DEL	7	7	147	163
END	6	6	23	159
PGUP	18	18	31	153
PGDN	3	3	30	161
UP ARROW	5	5	141	152
DN ARROW	24	24	145	160
RT ARROW	4	4	2	157
LT ARROW	19	19	26	155
ESCAPE	27	27	27	1
ENTER	13	13	10	28
BACKSPACE	127	127	127	14
TAB	9	15	148	165

Example INLIST() determines the quarter of the year for the current month. The current month is stored to the variable MONTH. Each CASE statement uses INLIST() to determine whether the contents of MONTH can be found in a list of month names. The name of the quarter returned is stored to the variable REPORTITLE.

```

SET TALK ON
STORE CMONTH( DATE( ) ) TO month
DO CASE
    CASE INLIST( month, 'January', 'February', 'March' )
        STORE 'First Quarter' TO reportitle
    CASE INLIST( month, 'April', 'May', 'June' )
        STORE 'Second Quarter' TO reportitle
    CASE INLIST( month, 'July', 'August', 'September' )
        STORE 'Third Quarter' TO reportitle
    OTHERWISE
        STORE 'Fourth Quarter' TO reportitle
ENDCASE
WAIT WINDOW reportitle

```

INPUT

Purpose	Inputs data into a memory variable
Syntax	INPUT [<expC>] TO <memvar>
Remarks	Included for backward compatibility. Use @ ... GET instead
See Also	@ ... GET, ACCEPT READ, WAIT

Description Use INPUT to input data from the keyboard into a memory variable or an array element.

INPUT is similar to ACCEPT, which doesn't require that character strings be delimited and creates only character-type memory variables.

Clauses **<expC>**
Include <expC> to display a prompting message. <expC> is the message you want to display.

<memvar>
<memvar> is the memory variable or an array element into which you want to input data entered from the keyboard. If <memvar> is a memory variable that doesn't exist, INPUT creates it.

The expression input from the keyboard determines the type of memory variable or array element created. If you input a numeric value, a numeric memory variable or array element is created; if you input a character value, a character memory variable or array element is created and so on. If you input a character value, it must be delimited by brackets or single or double quotation marks.

Examples

```
INPUT to mnum_exp
* enter the value 38
? mnum_exp
38

INPUT 'Enter company: ' TO mcompany
Enter company: 'XYZ Company'

? mcompany
XYZ Company
```

INSERT

Purpose	Inserts a new record into a table/.DBF
Syntax	INSERT [BEFORE] [BLANK]
Remarks	Included for backward compatibility. Use APPEND or INSERT – SQL instead
See Also	APPEND, CHANGE, EDIT, INSERT – SQL, SET CARRY

Description INSERT places a new record into the current table/.DBF immediately after the current record and displays the new record for editing.

If CARRY is SET ON and BLANK isn't included, data in the previous record is automatically copied into the new record.

If the file is indexed, INSERT works like APPEND.



INSERT isn't recommended for use with large tables/.DBFs because an insertion near the front of the table/.DBF forces rewriting of nearly every record. This can take a very long time. Use INSERT – SQL instead.

Clauses **BEFORE**

Issuing INSERT BEFORE inserts a new record into the current table/.DBF immediately before the current record. The record is then displayed for editing. Data can be entered only into the new record.

BLANK

Issuing INSERT BLANK inserts a blank record immediately after the current record. An editing window isn't displayed.

INSERT – SQL

Purpose	Appends a record to the end of a table/.DBF that contains the specified field values
Syntax	INSERT INTO <dbf name> [(<field name1> [, <field name2> [, ...]])] VALUES (<expr1> [, <expr2> [, ...]]) or INSERT INTO <dbf name> FROM ARRAY <array name> FROM MEMVAR
See Also	CREATE QUERY, CREATE TABLE – SQL, MODIFY QUERY, SELECT – SQL

Description INSERT appends a record to the end of an existing table/.DBF. The new record includes data listed in the INSERT command or included in the specified array.

Clauses **INSERT INTO <dbf name>**

<dbf name> specifies the name of the table/.DBF to which a record will be appended. <dbf name> can include a path and can be a name expression.

If the table/.DBF you specify isn't already open, it is opened exclusively in a new work area and the record is appended. The new work area isn't selected; the current work area remains selected.

If the table/.DBF you specify is open, INSERT appends the record to the table/.DBF. If the table/.DBF is open in a work area other than the current work area, it isn't selected after the record is appended; the current work area remains selected.

**[(<field name1> [, <field name2> [, ...]])]
VALUES (<expr1> [, <expr2> [, ...]])**

This clause specifies values (<expr1>, <expr2>, ...) to include in the record that is being added to the table/.DBF. You must specify values in the order defined by the table/.DBF structure unless you specify field names.

If you specify field names with <field name1>, <field name2> and so on, you can list the field names in any order. Each field name can be a name expression.

FROM ARRAY <array name>

<array name> is the name of an existing array containing data for the record being added.

FROM MEMVAR

FROM MEMVAR copies data from memory variables into the new record in the table/DBF specified with <dbf name>. Data is copied only into fields that have corresponding memory variables. If memory variables exist only for certain fields in the specified table/DBF, fields without corresponding memory variables remain empty.

Example

```
INSERT INTO parts (pno, descript, onhand, onorder, price, cost) ;  
VALUES ('B6722', 'Royal Reindeer', 2, 6, 27.99, 20.65)
```

```
USE customer  
COPY STRUCTURE TO cust2  
SCATTER MEMVAR  
INSERT INTO cust2 FROM MEMVAR
```

INSMODE()

Purpose	Returns the current mode of the Insert key, or sets the Insert key mode on or off
Syntax	INSMODE([<expL>])
Returns	Logical
See Also	CAPSLLOCK(), NUMLOCK(), SYS(2008), SYS(2009)

Description Issuing INSMODE() returns true (.T.) or false (.F.) depending on the current mode of the Insert key.

If the Insert key mode is on (a character is inserted in front of the character at the cursor), true (.T.) is returned. If the Insert key mode is off (a character overwrites the character at the cursor), false (.F.) is returned.

Parameter **<expL>**

The logical expression <expL> can be included to turn the Insert key mode on or off. INSMODE(.T.) turns it on, and INSMODE(.F.) turns it off. INSMODE() returns a logical value corresponding to the Insert key mode setting *before* INSMODE(.T.) or INSMODE(.F.) is issued.

Examples The first example uses INSMODE() to set the Insert key mode on, and the second example toggles the Insert key mode to the opposite state:

```
SET TALK ON
=INSMODE(.T.)  && Set insert mode on
? INSMODE( )

= INSMODE(!INSMODE( ))  && Toggle insert mode to opposite state
? INSMODE( )
```


INT()

Purpose	Returns the integer portion of a numeric expression
Syntax	INT(<expN>)
Returns	Numeric
See Also	CEILING(), FLOOR(), ROUND()

Description INT() evaluates a numeric expression and returns the integer part of the expression.

Parameter **<expN>**
INT() returns the integer portion of the numeric expression <expN>.

Examples

```
? INT(12.5)
12

? INT(6.25 * 2)
12

? INT(-12.5)
-12

STORE -12.5 TO number
? INT(number)
-12
```

ISALPHA()

Purpose	Returns true (.T.) if the leftmost character in the specified character expression is an alphabetic character
Syntax	ISALPHA(<expC>)
Returns	Logical
See Also	ISLOWER(), ISUPPER(), LOWER(), UPPER()

Description ISALPHA() returns true (.T.) if the first character is alphabetic (a – z or A – Z); otherwise, false (.F.) is returned.

Any characters after the first character in <expC> are ignored by ISALPHA().

Parameter <expC>
The character expression that ISALPHA() tests is specified with <expC>.

Examples

```
CLOSE DATABASES
USE salesman
? ISALPHA(name)
.T.

DISPLAY name
Record#  NAME
      1  Janet A. McCarthy

USE customer
? ISALPHA(cno)
.F.

DISPLAY cno
Record#  CNO
      1  14000
```

ISCOLOR()

Purpose	Determines whether or not your computer's graphics card can display color
Syntax	ISCOLOR()
Returns	Logical
See Also	SET COLOR, SET DISPLAY, SYS(2006)

Description ISCOLOR() returns true (.T.) if your graphics card has color capability (whether or not a color monitor is actually being used). If your graphics card doesn't provide color support, false (.F.) is returned.

Example

```
IF ISCOLOR( )
    SET COLOR SET TO color
ELSE
    SET COLOR SET TO mono
ENDIF
```

ISDIGIT()

Purpose	Returns true (.T.) if the leftmost character of the specified character expression is a digit (0 – 9)
Syntax	ISDIGIT(<expC>)
Returns	Logical
See Also	ISALPHA()

Description ISDIGIT() returns a logical value indicating whether a character expression begins with a number. ISDIGIT() returns true (.T.) if the first character is a digit (0 – 9); otherwise, false (.F.) is returned.

Any characters after the first character in <expC> are ignored by ISDIGIT().

Parameter <expC>

The character expression that ISDIGIT() tests is specified with <expC>.

Example

```
CLEAR
SET TALK ON
STORE SPACE(1) TO item
@ 10,10 GET item SIZE 1,40
READ

IF NOT ISALPHA(item) AND (NOT ISDIGIT(item))
  ? 'First character must be digit or alphabetic'
ENDIF
```

ISLOWER()

Purpose	Returns true (.T.) if the leftmost character in the specified character expression is a lower-case alphabetic character
Syntax	ISLOWER(<expC>)
Returns	Logical
See Also	ISALPHA(), ISUPPER(), LOWER(), UPPER()

Description ISLOWER() returns true (.T.) if the first character is a lower-case alphabetic character (a – z); otherwise, false (.F.) is returned.

Any characters after the first character in <expC> are ignored by ISLOWER().

Parameter <expC>
The character expression that ISLOWER() tests is specified with <expC>.

Examples

```
? ISLOWER('redmond')  
.T.  
  
? ISLOWER('Redmond')  
.F.
```

ISUPPER()

Purpose	Returns true (.T.) if the first character in a character expression is an upper-case alphabetic character
Syntax	ISUPPER(<expC>)
Returns	Logical
See Also	ISALPHA(), ISLOWER(), LOWER(), UPPER()
Description	ISUPPER() returns true (.T.) if the first character is an upper-case alphabetic character (A – Z); otherwise, false (.F.) is returned. Any characters after the first character in <expC> are ignored by ISUPPER().
Parameter	<expC> The character expression that ISUPPER() tests is specified with <expC>.
Examples	? ISUPPER('Ready?') .T. ? ISUPPER('ready?') .F.

JOIN

Purpose	Creates a table/.DBF by joining two existing tables/.DBFs
Syntax	JOIN WITH <expN> WITH <expC> TO <file> FOR <expL> [FIELDS <field list>] [NOOPTIMIZE]
Remarks	Included for backward compatibility. Use SELECT – SQL
See Also	SET FIELDS, SET RELATION

Description JOIN creates a new table/.DBF from two other tables/.DBFs: the current table/.DBF and a second table/.DBF identified by its work area number or alias. JOIN positions the record pointer on the first record in the current table/.DBF and searches through the records in the second table/.DBF.



This command can take a long time to execute depending on the size of the two tables/.DBFs being joined. In addition, it is possible to exceed the available disk space with JOIN even if the sizes of the two tables/.DBFs are modest.

Clauses **<expN> | <expC>**
Specify the work area number of the second table/.DBF with <expN>. Specify the table/.DBF alias with <expC>.

TO <file>
<file> is the name of the table/.DBF to create from the joined tables/.DBFs.

FOR <expL>

JOIN evaluates the logical expression <expL> for each record. If <expL> is true, a new record is written to the new file. JOIN then moves to the second record in the current table/.DBF and repeats the procedure.

Rushmore optimizes a query created with JOIN ... FOR if <expL1> is an optimizable expression. For best performance, use an optimizable expression in the FOR clause. A discussion of Rushmore optimizable expressions appears in the Optimizing Your Application chapter in the *FoxPro Developer's Guide*.

FIELDS <field list>

You can specify a list of fields to be included in the new table/.DBF by including a field list. <field list> can include the names of fields from both the current table/.DBF and from the aliased table/.DBF.

NOOPTIMIZE

Include NOOPTIMIZE to disable Rushmore optimization of JOIN. For further information on NOOPTIMIZE, see SET OPTIMIZE in this manual, or consult the discussion of Rushmore optimization in the Optimizing Your Application chapter in the *FoxPro Developer's Guide*.

Example

```
CLOSE DATABASES
USE customer ORDER cno

USE invoices ORDER cno IN 0

JOIN WITH invoices TO custinv FOR customer.cno = invoices.cno ;
    FIELDS customer.cno, customer.company, invoices.ino
USE custinv
DISPLAY STRUCTURE
BROWSE
USE
ERASE custinv.dbf
```


KEY()

Purpose	Returns the key index expression of the master (controlling) index file
Syntax	KEY([<cdx file>], <expN1> [, <expN2> <expC>])
Parameters	<cdx file> The name of the compound index file <expN1> Specifies which index expression to return from the open index files <expN2> The work area number for a table/.DBF <expC> The table/.DBF alias
Returns	Character
See Also	INDEX, REINDEX, SET INDEX, SYS(14), USE

Description KEY() returns the index expression of an open index file. An index expression is specified when an index file is created with INDEX. The index expression determines how a table/.DBF is displayed and accessed when the index file is opened as the master controlling index file.

For additional information on creating index files and index expressions, see INDEX in this chapter.

Parameters **<cdx file>**

KEY() can be used to return the index key expressions for tags in multiple entry compound index files. Specify the name of the compound index file with <cdx file>. The compound index file you specify can be the structural compound index file automatically opened with the table/.DBF or an independent compound index file.

<expN1>

USE and SET INDEX both support an index file list that lets you open multiple indexes for a table/.DBF. Any combination of single-entry .IDX index files, structural compound index files or independent compound index files can be included in the index file list.

The numeric expression <expN1> specifies which index expression to return from the open index files. KEY() returns index expressions from open index files in the following order:

1. Index expressions from single-entry .IDX index files (if any are open) are returned first. The order in which the single-entry index files are included in USE or SET INDEX determines how the index expressions are returned.
2. Index expressions for each tag in the structural compound index (if one is present) are returned next. The index expressions are returned from the tags in the order in which the tags are created in the structural compound index.
3. Index expressions for each tag in any open independent compound indexes are returned last. The index expressions are returned from the tags in the order in which the tags are created in the independent compound indexes.

The null string is returned if <expN1> is greater than the total number of open single-entry .IDX files and structural compound and independent compound index tags.

<expN2> | <expC>

If you omit the work area and alias, index expressions are returned from index files open for the table/.DBF in the current work area. To return index expressions from index files open in a specific work area, include the work area number <expN2> or the table/.DBF alias <expC>.

If no table/.DBF has the alias you specify, the message "Alias not found" is displayed.

Example

```
CLEAR
CLOSE DATABASES
USE customer    && customer has a structural compound index
?KEY(1)
CNO

?KEY(2)
COMPANY

?KEY(3)
CONTACT
```

KEYBOARD

Purpose	Places the specified character expression in the keyboard buffer
Syntax	KEYBOARD <expC> [PLAIN]
See Also	CHRSAW(), ON KEY LABEL, PLAY MACRO, SET FUNCTION

Description Use KEYBOARD to stuff the keyboard buffer with an arbitrary character string that remains in the buffer until FoxPro looks for input. At that point, the data in the keyboard buffer is read and acted upon as if the data had been entered directly from the keyboard.

You can use KEYBOARD to create self-running demonstration systems that showcase your applications.

Clauses **<expC>**

The character expression <expC> is stuffed into the keyboard buffer. The character expression can be a character string, a key label, a set of key labels, or a user-defined function (UDF) that returns a character expression.

If <expC> is a key label, it must be enclosed in braces and quotes. For example,

```
KEYBOARD '{CTRL+LEFTARROW}'
```

For a listing of key labels, see ON KEY LABEL in this chapter.

The keyboard buffer can be stuffed with up to 128 keystrokes. Once the keyboard buffer is full, additional keystrokes are discarded.

PLAIN

If you have keyboard macros defined or ON KEY LABELs active, you can include PLAIN to bypass these key assignments. PLAIN stuffs the keyboard with the literal key character, not the key assignment.

For example, if you have assigned a routine to the A key with ON KEY LABEL and A is included in <expC>, use PLAIN to stuff the keyboard with the letter A. The routine assigned to A isn't executed.

Example In this example, a window is displayed and data is entered. If the entered customer name is found in the CUSTOMER table/DBF, the remaining fields are automatically filled by stuffing the KEYBOARD buffer from the remaining fields.

```

CLOSE DATABASES
SET TALK OFF
SET SAFETY OFF
STORE SPACE(40) TO mcompany, maddress, mcomments
STORE SPACE(24) TO mcity
STORE SPACE(2) TO mstate
STORE SPACE(10) TO mzip

SELECT 1
USE customer
SET ORDER TO TAG company

DEFINE WINDOW menter FROM 7,10 TO 19,70 PANEL
ACTIVATE WINDOW menter

@ 1,3    SAY 'Company: '    GET mcompany VALID v_cust (TRIM(mcompany))
@ 3,3    SAY 'Address: '   GET maddress
@ 5,3    SAY 'City: '      GET mcity
@ 7,3    SAY 'State: '     GET mstate
@ 7,18   SAY 'Zip: '       GET mzip
@ 9,3    SAY 'Comments: '  GET mcomments
READ

DEACTIVATE WINDOW menter
USE && Close table/.DBF

FUNCTION v_cust
PARAMETER mcomp
SEEK UPPER(mcomp)

IF FOUND( )
    KEYBOARD address + city + state + zip
ENDIF
RETURN .T.

```

LABEL

Purpose	Prints labels from a table/.DBF file and a label definition file
Syntax	LABEL [FORM <file1> ?] [ENVIRONMENT] [<scope> [FOR <expL1> [WHILE <expL2> [NOCONSOLE] [NOOPTIMIZE] [PDSETUP] [PREVIEW] [SAMPLE] [TO PRINTER [PROMPT] TO FILE <file2>]
Remarks	PROMPT clause — FoxPro for Windows only
See Also	MODIFY LABEL, CREATE LABEL

Description LABEL displays or prints labels under the control of a label definition file. Label definition files are created with MODIFY LABEL or CREATE LABEL.

The default extension for a label definition file is .LBX. If the label definition file is on a drive other than the default drive or in a directory other than the current directory, the drive designator and directory must also be specified.

If LABEL is issued without any additional arguments, the Open dialog appears, displaying a list of existing label files to choose from.

Clauses **FORM <file1>**

You can specify which labels to print by including the name of a label definition table/.DBF with <file1>.

?

If you include the ? clause, the Open dialog appears, displaying a list of existing label files to choose from.

ENVIRONMENT

When you create or modify labels, you can save the current FoxPro environment with the label definition file. Saving the FoxPro environment places an additional record in the label definition table. This record contains the names of all open tables/.DBFs and index files, the index order, and any relationships between the tables/.DBFs.

Include ENVIRONMENT to restore the environment saved in the label definition file.

<scope>

The scope clauses are: ALL, NEXT <expN>, RECORD <expN>, and REST. These are explained in the Overview of the FoxPro Language chapter of this manual. Commands which include <scope> operate only on the table/.DBF in the active work area.

You can specify a scope of records printed. Only the records that fall within the range of records specified by the scope are printed.

The default scope for LABEL is ALL records.

FOR <expL1>

Including the FOR clause lets you conditionally print records, filtering out undesired records. If the FOR clause is included, only the records that satisfy the logical condition <expL1> are printed.

Rushmore optimizes a query created with LABEL ... FOR if <expL1> is an optimizable expression. For best performance, use an optimizable expression in the FOR clause. A discussion of Rushmore optimizable expressions appears in the Optimizing Your Application chapter in the *FoxPro Developer's Guide*.

WHILE <expL2>

If the WHILE clause is included, records are printed for as long as the logical expression <expL2> evaluates to true (.T.).

NOCONSOLE

Include NOCONSOLE to suppress the display of label output to the desktop in FoxPro for MS-DOS, the main FoxPro window in FoxPro for Windows or a user-defined window when labels are being printed or sent to a file.

NOOPTIMIZE

To disable Rushmore optimization, include **NOOPTIMIZE**. For further information on **NOOPTIMIZE**, see **SET OPTIMIZE** in this chapter, or consult the discussion of Rushmore technology in the *Optimizing Your Application* chapter in the *FoxPro Developer's Guide*.

PDSETUP

In FoxPro for MS-DOS, when you create a label definition with the FoxPro Label Designer, you can specify a printer driver setup that determines the printed appearance of labels. If you save the environment with the label definition, the printer driver setup is also saved. If you include the **PDSETUP** key word when printing labels, the printer driver setup is loaded and used when the labels are printed. In FoxPro for Windows, you can include **PDSETUP** to use a printer driver setup for printing character-based labels created in FoxPro for MS-DOS. **PDSETUP** is ignored when you print graphics-based labels created in FoxPro for Windows.

PREVIEW

Including **PREVIEW** displays the labels in a preview window before printing but doesn't print the labels. To print the labels, you must issue **LABEL** without the **PREVIEW** clause.

SAMPLE

SAMPLE can be used to test label alignment in character-based labels created in FoxPro for MS-DOS. If you include **SAMPLE**, the prompt "Do you want more samples?" is displayed after a sample label is displayed or printed. Choose **Yes** to retest your label alignment.

TO PRINTER [PROMPT] | TO FILE <file2>

Include **TO PRINTER** to send labels to the printer.

In FoxPro for Windows, include the optional **PROMPT** clause to display a dialog before printing starts. Place the **PROMPT** key word immediately after **TO PRINTER**. **PROMPT** is ignored in FoxPro for MS-DOS.

In this dialog you can adjust printer settings. You can specify the number of copies and page numbers to print and direct print output to a file. The Windows Printer Setup dialog can be opened to adjust additional printer settings. The printer settings you can adjust depend on the currently installed printer driver.

Include **TO FILE** to send character-based labels created in FoxPro for MS-DOS to the text file specified with **<file2>**. In FoxPro for Windows, you cannot use the **TO FILE** option to send graphics-based labels created in FoxPro for Windows to a text file.

LASTKEY()

Purpose	Returns a number corresponding to the last key pressed
Syntax	LASTKEY()
Returns	Numeric
See Also	CHRSAW(), INKEY(), READKEY()

Description LASTKEY() returns an integer corresponding to the last key pressed. The values returned by LASTKEY() are the same as those returned by INKEY(). See INKEY() in this chapter for a list of keys and their return values.

The LASTKEY() is updated when you move between controls during a READ.

Example In this example, the data from the current record in the CUSTOMER table/DBF is placed in an array with SCATTER. You can edit each field. Press Escape to cancel your changes. LASTKEY() tests to see whether the READ was exited by pressing Escape. If Escape has been pressed, GATHER restores the original values to the fields.

```
CLOSE DATABASES
SET TALK OFF
USE salesman
COPY TO temp
USE temp
SCATTER MEMVAR MEMO      && save old values
DEFINE WINDOW menter FROM 5,10 to 20,70 PANEL
ACTIVATE WINDOW menter
@ 1,3   SAY 'Name: '   GET m.name
@ 3,3   SAY 'Address: ' GET m.address
@ 5,3   SAY 'City: '   GET m.city
@ 7,3   SAY 'State: '  GET m.state
@ 7,18  SAY 'Zip: '    GET m.zip
@ 9,13  SAY 'Notes:'
@ 9,23  EDIT m.notes SIZE 3,26
@ 13,13 SAY 'Press Escape to discard changes'
READ
```



```
IF LASTKEY( ) != 27
    GATHER MEMVAR    && save changes
    WAIT WINDOW 'Changes saved' NOWAIT
ELSE
    && no GATHER
    WAIT WINDOW 'Changes not saved' NOWAIT
ENDIF
DEACTIVATE WINDOW menter
BROWSE
USE
ERASE temp.dbf
```

LEFT()

Purpose	Returns a specified number of characters from a character expression, starting with the leftmost character
Syntax	LEFT(<expC>, <expN>)
Parameters	<expC> The character expression from which characters are returned <expN> The number of characters returned
Returns	Character
See Also	AT(), LTRIM(), RIGHT(), RTRIM(), SUBSTR()

Description LEFT() returns characters from the character expression beginning with the first character on the left and continuing for a specified number of characters.

Parameters **<expC>**
 Characters are returned from the character expression specified with <expC>.

<expN>
 The number of characters returned from the character expression is specified with <expN>. If <expN> is greater than the length of <expC>, all of the character expression is returned. The null string is returned if <expN> is negative or 0.

LEFT() is identical to SUBSTR() with a starting position of 1.

Example ? LEFT('Redmond, WA', 7)
Redmond

LEN()

Purpose	Returns the number of characters in a character expression
Syntax	LEN(<expC>)
Returns	Numeric
See Also	FSIZE()

Description LEN() returns the length of the character expression <expC>.

Parameter <expC>
 <expC> is the character expression whose length you want to determine.

Examples In the following example, the length of the field ONO is used to store spaces to a memory variable. When you are prompted for input, enter a single digit number, such as 1. When the READ command is terminated, LEN() is used again to see whether the memory variable is empty. If it is empty, the program is exited.

```

CLOSE DATABASES
SET TALK OFF
CLEAR
USE customer ORDER ono
STORE .F. TO mexit
DO WHILE NOT mexit
    STORE SPACE(LEN(ono)) TO mcustono
    @ 10,10 SAY 'Enter customer ono to locate: ' GET mcustono
    @ 12,15 SAY 'Leave entry blank to exit'
    WAIT WINDOW 'Try numbers from 1-7' NOWAIT
    READ
    @ 20,0 CLEAR
    IF LEN(TRIM(mcustono)) = 0
        STORE .T. TO mexit
    ELSE
        SEEK mcustono
        IF FOUND( )
            @ 20,10 SAY 'Company is ' + company
        ELSE
            @ 20,10 SAY 'No company for customer: ' + mcustono
        ENDIF
    ENDIF
ENDDO
USE

```

LIKE()

Purpose	Determines whether or not a character expression is contained in another character expression
Syntax	LIKE(<expC1>, <expC2>)
Parameters	<expC1> The character expression to match <expC2> The character expression being compared with <expC1>
Returns	Logical
See Also	\$, AT(), ATC(), OCCURS, RAT(), SET COMPATIBLE

Description LIKE() returns a logical value indicating whether a character expression <expC1> matches a second character expression, <expC2>. LIKE() returns true (.T.) if <expC1> matches <expC2>. LIKE() returns false (.F.) if <expC1> doesn't match <expC2>.

SET COMPATIBLE determines how LIKE() evaluates <expC1> and <expC2>. If COMPATIBLE is SET ON, <expC1> and <expC2> have all trailing blanks removed before they are compared. If COMPATIBLE is SET OFF, any trailing blanks in <expC1> and <expC2> are used in the comparison.

Parameters <expC1>

<expC1> is the character expression that can contain the wildcard characters * and ?. The question mark matches any single character in <expC2> and the asterisk matches any number of characters. You can mix any number of wildcard characters in any combination in <expC1>.

<expC2>

LIKE() compares <expC2> with <expC1>. <expC2> must match <expC1> letter for letter for LIKE() to return true.

Example In this example, all descriptions from PARTS.DBF with the first two letter “Ab” are displayed.

```
CLOSE DATABASES
USE parts
? 'All parts descriptions with first two letters "Ab"'
?
SCAN FOR LIKE('Ab*', descript)
? descript
ENDSCAN
USE
```

LINENO()

Purpose	Returns the line number of a line being executed in a program relative to the first line of the main program
Syntax	LINENO([1])
Returns	Numeric
See Also	ERROR(), MESSAGE(), PROGRAM(), SYS(16)

Description LINENO() returns the current program line number in an executing program. Program lines are counted starting from the top of the program. Comment lines, continuation lines and blank lines are included in the line number count. If a program is suspended during execution, LINENO() returns the number of the program line at which program execution is suspended. LINENO() returns 0 if a program is cancelled.

By default, line numbers are returned relative to the beginning of the main program. If a procedure is called, line numbering continues from the top of the calling program.

LINENO() is useful for debugging programs. You can set a breakpoint to stop program execution at a specific line number by specifying the following command in the Debug window:

```
LINENO( ) = <expN>
```

Program execution is suspended when the value of LINENO() is equal to <expN>.

Parameter 1

To return the line number relative to the first line of the current program or procedure, include 1.

Example This is a section of a simple error-handling routine.

```
ON ERROR DO bug_proc WITH LINENO( )
BRWS      && causes error
ON ERROR

*** Bug_Proc error handler ***
PROCEDURE bug_proc
PARAMETERS badline
WAIT WINDOW 'Error occurred at line:' + STR(badline)
RETURN
```

LIST

Purpose	Continuously displays .DBF or environment information
Syntax	LIST [FIELDS <expr list>] [<scope>] [FOR <expL1>] [WHILE<expL1>] [OFF] [NOCONSOLE] [NOOPTIMIZE] [TO PRINTER TO FILE <file>] LIST FILES [ON <drive>] [LIKE <skel>] [TO PRINTER TO FILE <file>] LIST MEMORY [LIKE <skel>] [NOCONSOLE] [TO PRINTER TO FILE <file>] LIST STATUS [NOCONSOLE] [TO PRINTER TO FILE <file>] LIST STRUCTURE [NOCONSOLE] [TO PRINTER TO FILE <file>]
See Also	DISPLAY, DISPLAY FILES, DISPLAY MEMORY, DISPLAY STATUS, DISPLAY STRUCTURE, SET DELETED
Description	These LIST commands are identical to the DISPLAY commands except for the following differences: • The scope for LIST defaults to ALL. • LIST doesn't prompt you after filling the desktop in FoxPro for MS-DOS, the main FoxPro window in FoxPro for Windows or a user-defined window with information. • LIST doesn't display records flagged for deletion when DELETED is SET ON. For additional information about the LIST commands, see the corresponding DISPLAY commands in this chapter.

LOAD

Purpose	Places a binary file in memory
Syntax	LOAD <file> [SAVE] [NOSAVE]
See Also	CALL

Description Use LOAD to place binary routines from disk in memory. A maximum of sixteen 64K files can be loaded into memory at one time.

Clauses **<file>**
<file> is the name of the file to load into memory. The default extension for a binary file is .BIN. When you call a binary routine, omit the extension. If a file you load has the same file name of a previously loaded file but a different file name extension, the new file replaces the previously loaded file.

SAVE **NOSAVE**

LOAD and CALL support optional key words for use with binary routines that write to the desktop in FoxPro for MS-DOS or to the main FoxPro window in FoxPro for Windows. You can include SAVE (NOSAVE is the default) when loading a binary routine, or both SAVE and NOSAVE when calling a binary routine. If included with CALL, SAVE and NOSAVE override the key word included with LOAD.

If SAVE is included, FoxPro copies the current contents of video RAM into the FoxPro desktop or main FoxPro window upon return from the binary routine (assuming that the binary routine is writing directly to video RAM).

This means that anything written to the FoxPro desktop or main FoxPro window by the binary routine is known to FoxPro and is treated as if FoxPro had written it. In particular, such externally written data isn't erased when a FoxPro object is dragged over it.

Unless necessary, don't include SAVE, because it takes FoxPro some time to scan and save the screen contents upon each return from the binary routine.

Creating a Binary File

Follow these steps when creating a binary file:

1. Assemble the program and create an object file with this command:

```
MASM <file>
```

2. Link the object file and create an executable file with this command:

```
LINK <file>
```

3. Create a binary file with this command:

```
EXE2BIN <file>
```

In addition, follow these rules when creating a FoxPro binary file:

1. The first executable instruction must originate at an offset of 0.
2. The binary program must not change the length of memory variables passed as arguments.
3. The amount of memory used or allocated can't exceed the actual program size because the file size is used to determine the amount of memory allocated.
4. The SS and CS registers must be restored before control is returned to FoxPro.
5. A far return should be used to return control to FoxPro when the program is executed with LOAD and CALL.
6. When a program is terminated by an exit system call rather than a far return, you must execute assembly programs with the RUN command.



Carefully read the following assembly language interface warnings:

When using string variables, don't change the length of the strings. It is possible that the free memory list could be corrupted if the length of a string variable is changed by even 1 byte.

When saving a string address from one call to the next, you should also reset the string address you store to just prior to using the string variable.

LOCATE

Purpose	Sequentially searches the table/.DBF for the first record that matches a given expression
Syntax	LOCATE FOR <expL1> [<scope>] [WHILE <expL2>] [NOOPTIMIZE]
See Also	CONTINUE, EOF(), FIND, FOUND(), SEEK, SEEK(), RECNO()

Description LOCATE sequentially searches the current table/.DBF for the first record that matches a logical expression. The table/.DBF doesn't have to be indexed.

If LOCATE finds a matching record, its record number is displayed unless SET TALK is OFF, in which case, you can use RECNO() to return the number of the matching record. If a matching record is found, FOUND() returns true (.T.), and EOF() returns false (.F.).

After LOCATE finds a matching record, you can issue CONTINUE to search the remainder of the table/.DBF for additional matching records. When CONTINUE is executed, the search process resumes, starting with the record immediately following the matching record. You can issue CONTINUE repeatedly until the end of the scope or the end of the table/.DBF is reached.

If a match isn't found, RECNO() returns the number of records in the table/.DBF plus one, FOUND() returns false, and EOF() returns true.

LOCATE and CONTINUE are specific to the current work area. If another work area is selected, the search process can be continued when the original work area is reselected.

You can also execute LOCATE by choosing **Locate...** from the **Record** menu. See the Record Menu chapter in the *FoxPro User's Guide* for more information on using LOCATE interactively.

Clauses FOR <expL1>

LOCATE sequentially searches the current table/.DBF for the first record that matches the logical expression <expL1>.

Rushmore optimizes a query created with LOCATE FOR if <expL1> is an optimizable expression. For best performance, use an optimizable expression in the FOR clause. A discussion of Rushmore optimizable expressions appears in the Optimizing Your Application chapter in the *FoxPro Developer's Guide*.

<scope>

The scope clauses are: ALL, NEXT <expN>, RECORD <expN>, and REST. These are explained in the Overview of the FoxPro Language chapter of this manual. Commands which include <scope> operate only on the table/.DBF in the active work area.

You can specify a scope of records to locate. Only the records that fall within the range of records specified by the scope are located.

The default scope for LOCATE is ALL records.

WHILE <expL2>

If the WHILE clause is included, records are searched for as long as the logical expression <expL2> evaluates to true (.T.).

NOOPTIMIZE

To disable Rushmore optimization of LOCATE, include NOOPTIMIZE. For further information on NOOPTIMIZE, see SET OPTIMIZE in this chapter, or consult the discussion of Rushmore optimization in the Optimizing Your Application chapter in the *FoxPro Developer's Guide*.

Example

In this example, all records for customers from New York state are located. The total count is then displayed on the screen.

```

CLOSE DATABASES
SET TALK OFF
USE customer
STORE 0 TO mcount
LOCATE FOR ALLTRIM(UPPER(state)) = 'NY'
DO WHILE FOUND( )
    mcount = mcount + 1
    CONTINUE
ENDDO
? 'Total customers from New York: ' + LTRIM(STR(mcount))

```

LOCFILE()

Purpose	Returns the path of the specified file name
Syntax	LOCFILE(<expC1> [, <expC2>] [, <expC3>])
Parameters	<expC1> The file to locate <expC2> A list of file name extensions <expC3> Open dialog prompt
Returns	Character
See Also	FILE(), GETFILE(), PUTFILE(), SET PATH

Description LOCFILE() locates a file on disk and returns the file name with its path. If the file can't be located, the Open dialog is brought forward so you can search for the file.

Parameters <expC1>

The name of the file to locate is specified with <expC1>. If <expC1> includes only a file name, LOCFILE() searches the default directory first. If the file isn't found in the default directory, the FoxPro path is then searched. Use SET PATH to specify the FoxPro path.

If <expC1> includes a path and a file name, the specified directory is searched. If the file can't be found in the specified directory, LOCFILE() searches the default directory and then the FoxPro path.

If the file is located, its file name and path are returned.

If the file can't be located in the default directory, the FoxPro path or a specified directory, the Open dialog is displayed. The Open dialog can be used to locate the file. When a file is chosen from the dialog, the file name is returned with the file's path, which is appended to the FoxPro path.

If you exit the Open dialog by choosing **Cancel** or pressing Escape, an alert "File doesn't exist" is displayed and LOCFILE() doesn't return a value.

<expC2>

If the file name you specify with <expC1> doesn't include an extension, FoxPro applies the file extensions listed in <expC2> to the file name and searches for the file again.

<expC2> also specifies the file name extensions of the files displayed in the Open dialog when the **All Files** check box isn't checked. For a complete list of the file name extensions used by FoxPro, see the file name extension table in the FoxPro *Developer's Guide*.

<expC2> can take a variety of forms:

- If <expC2> contains a single extension (for example, .PRG), only files with that extension are displayed.
- <expC2> can contain a list of file extensions separated by semicolons. For example, including PRG;FXP displays all files with the extensions .PRG and .FXP.

If files have the same name but different extensions (for example, CUSTOMER.PRG and CUSTOMER.FXP), only the file with the extension listed first in <expC2> is displayed.

- <expC2> can contain a list of file name extensions separated by vertical bars (for example, PRG|FXP). In this case, all files with any of these extensions are displayed, even if the files have the same names.
- If <expC2> contains just a semicolon (;), all files without an extension are displayed.
- If <expC2> is the null string, all files in the current directory are displayed.
- <expC2> can also contain MS-DOS wildcard characters (* and ?). All files with extensions that meet the wildcard criteria are displayed. For example, if <expC2> is ?X?, all files with the extension .FXP, .EXE, or .TXT are displayed.

<expC3>

The optional character expression <expC3> is the prompt displayed at the top of the Open dialog.

LOCK()

Purpose	Attempts to lock one or more records in the current or specified table/.DBF
Syntax	LOCK([<expN> <expC1>] [<expC2>, <expN> <expC1>])
Parameters	<expN> The work area number for a table/.DBF <expC1> The table/.DBF alias <expC2> A list of one or more record numbers separated by commas
Returns	Logical
See Also	CLEAR, CLOSE, FLOCK(), RLOCK(), SET MULTILOCKS, SET REPROCESS, UNLOCK, USE

Description Use LOCK() to place a lock on a record or a set of records in a table/.DBF. LOCK() is identical to RLOCK().

If the lock or locks are successfully placed, a true (.T.) value is returned. Locked records are available for both read and write access to the user who placed the locks; they are available for read-only access to all other users on the network.

Executing LOCK() doesn't guarantee that the record lock or locks will be successfully placed. A record lock can't be placed on a record already locked by another user or in a table/.DBF locked by another user. If the record lock or locks can't be placed for any reason, a false value (.F.) is returned.

By default, LOCK() makes one attempt to lock a record. Use SET REPROCESS to automatically retry a record lock when the first attempt fails. SET REPROCESS determines the number of lock attempts or the length of time over which lock attempts are made when the initial lock attempt is unsuccessful. See SET REPROCESS in this chapter for more information.

SET MULTILOCKS determines whether you can lock multiple records in a table/.DBF. If MULTILOCKS is SET OFF (the default), you can lock only a single record in a table/.DBF. When SET MULTILOCKS is ON, you can lock multiple records in a table/.DBF. See SET MULTILOCKS in this chapter for further information.

Parameters <expN> | <expC1>

If you don't specify a work area or table/.DBF alias, LOCK() attempts to lock the current record in the table/.DBF in the current work area.

To attempt to lock the current record in a table/.DBF open in another work area, include the work area number <expN> or the table/.DBF alias <expC1>.

<expC2>

You must include <expC2> to attempt to lock multiple records. MULTILOCKS must be SET ON and you must include the work area or alias of the table/.DBF for which you are attempting to place multiple record locks.

<expC2> is a list of one or more record numbers. LOCK() attempts to lock all of these records. The record numbers specified with <expC2> are separated by commas. For example, to attempt record locks on the first four records in a table/.DBF, <expC2> must contain 1,2,3,4.

You can also lock multiple records by moving the record pointer to the record you would like to lock, issuing LOCK() or RLOCK() and then repeating these steps for each additional record.

If all the records specified with <expC> are successfully locked, true (.T.) is returned. If even one of the records specified with <expC2> can't be locked, false (.F.) is returned and none of the records is locked. However, any existing record locks remain in place. Multiple record locking is an additive process — placing additional record locks doesn't release locks on other records.

The maximum number of records that can be locked in each work area is approximately 8,000. It is always faster to lock the entire table/.DBF rather than even a small number of records.

Unlocking Records

A table/.DBF record can be unlocked only by the user who placed the lock. You can release record locks by issuing UNLOCK, closing the table/.DBF or exiting FoxPro.

UNLOCK can be used to release record locks in the current work area, a specific work area or in all work areas. See UNLOCK in this chapter for additional information.

Toggling MULTILOCKS from ON to OFF or from OFF to ON implicitly performs UNLOCK ALL — all record locks in all work areas are released.

Tables/DBFs can be closed with USE, CLEAR ALL or CLOSE DATABASES.

For additional information about record and file locking and sharing tables/DBFs on a network, see the FoxPro in a Multi-User Environment chapter in the *FoxPro Developer's Guide*.

Example The following example locks and unlocks the first four records in CUSTOMER.DBF and SALESMAN.DBF.

```
CLOSE DATABASES
SET REPROCESS TO 3 AUTOMATIC
STORE '1,2,3,4' TO mreclist
old_exc = SET('EXCLUSIVE')
SET EXCLUSIVE OFF
SELECT 0
USE salesman
SELECT 0
USE customer
? LOCK('1,2,3,4', 'customer') && Lock 1st 4 records in customer
? RLOCK(mreclist, 'salesman') && Lock 1st 4 records in salesman
UNLOCK IN customer
UNLOCK IN salesman
SET EXCLUSIVE &old_exc
```


LOG()

Purpose	Returns the natural logarithm (base e) of the specified numeric expression
Syntax	LOG(<expN>)
Returns	Numeric
See Also	EXP(), LOG10(), SET DECIMALS

Description The mathematical function LOG() returns the natural logarithm of a numeric expression. The base for the natural logarithm is the constant e . The number of decimal places returned in the result is specified with SET DECIMALS.

Parameter **<expN>**
LOG() returns the value of x in the equation $e^x = \text{<expN>}$. The numeric expression <expN> must be greater than 0.

Examples

```
? LOG(1)
0.00

STORE EXP(2) TO esquared
? LOG(esquared)
2.00
```

LOG10()

Purpose	Returns the common logarithm (base 10) of the specified numeric expression
Syntax	LOG10(<expN>)
Returns	Numeric
See Also	EXP(), LOG(), SET DECIMALS

Description The mathematical function LOG10() returns the common logarithm of a numeric expression. The base for the common logarithm is 10. The number of decimal places returned in the result is specified with SET DECIMALS.

Parameter <expN>
LOG10() returns the value of x in the equation $10^x = \text{<expN>}$. The numeric expression <expN> must be greater than 0.

Examples

```
? LOG10(10)
1.00

STORE 100 TO baseten
? LOG10(baseten)
2.00

? LOG10(baseten^2)
4.00
```

LOOKUP()

Purpose	Searches a table/.DBF for the first record with a field matching the specified expression
Syntax	LOOKUP(<field1>, <search expr>, <field2> [, <expC>])
Returns	Character, Numeric, Logical or Date
See Also	FIND, INDEX, LOCATE, SEEK, SEEK()

Description LOOKUP() searches the table/.DBF in the current work area for the first occurrence of a specified search expression. If the search is successful, LOOKUP() moves the record pointer to the matching record and returns the contents of a specified field in the record.

If the search expression isn't found, LOOKUP() returns an empty string of the same length and data type as <field1>. The record pointer is positioned at the end of the file.

If LOOKUP() is used to search a parent table/.DBF, record pointers in all related child tables/.DBFs are moved to the related records.

This function can't be optimized with Rushmore optimization.

Clauses **<field1>**

If the search is successful, the value of the field specified with <field1> is returned. If the search is unsuccessful, an empty string of the same length and data type as <field1> is returned.

<search expr>

The search expression is specified with <search expr>. The search expression is usually the contents of a field in the table/.DBF, or it can correspond to the index expression of the active index or compound index tag.

<field2>

If the table/.DBF doesn't have an active index, LOOKUP() performs a sequential search through the field specified with <field2>.

If the table/.DBF has an active index, LOOKUP() performs a faster search of the primary index.

<expC>

<expC> is the name of a compound index tag for LOOKUP() to search. A compound index search is the fastest search LOOKUP() can perform.

Example

In the following example, LOOKUP() uses the active index tag, CITY, to search the CITY field for the first occurrence of the search string "Dallas." If the search is successful, LOOKUP() returns the contents of the COMPANY field. @ ... SAY displays the LOOKUP() return value.

```
CLOSE DATABASES
```

```
CLEAR
```

```
USE customer ORDER city
```

```
@ 2,2 SAY LOOKUP(company, 'Dallas', city, 'city')
```

LOWER()

Purpose	Returns a specified character expression in lower-case letters
Syntax	LOWER(<expC>)
Returns	Character
See Also	ISALPHA(), ISLOWER(), ISUPPER(), PROPER(), UPPER()

Description LOWER() converts all upper-case letters (A – Z) in the character expression to lower-case (a – z). All other characters in the character expression remain unchanged.

Parameter **<expC>**
<expC> is the character expression you want to convert to lower-case.

Example STORE 'FOX' TO name
 ? LOWER(name)
 fox

LTRIM()

Purpose	Returns the specified character expression with leading blanks removed
Syntax	LTRIM(<expC>)
Returns	Character
See Also	ALLTRIM(), LEFT(), RIGHT(), RTRIM(), SUBSTR(), TRIM()

Description LTRIM() returns a character string that results from removing the leading blanks from a character expression. This function is especially useful for removing the leading blanks that are inserted when you use STR() to convert a numeric value to a character string.

Parameter <expC>
Include the character expression <expC> from which you want to trim the leading blanks.

Example

```
STORE 'Redmond' TO mcity
STORE '      Washington' TO mstate
? mcity, mstate
Redmond      Washington

? mcity, LTRIM(mstate)
Redmond Washington
```

LUPDATE()

Purpose	Returns the date on which the current or specified table/.DBF was last updated
Syntax	LUPDATE([<expN> <expC>])
Parameters	<expN> The work area number of a table/.DBF <expC> The table/.DBF alias
Returns	Date
See Also	DIR, FILER

Description LUPDATE() returns the date on which a table/.DBF was last updated. This function is useful in update procedures.

Parameters <expN> | <expC>

LUPDATE() returns the date of the last update to the table/.DBF in the current work area. To return the last update to a table/.DBF open in a different work area, include the work area number <expN> or the table/.DBF alias <expC>.

If no table/.DBF is open in the work area you specify, LUPDATE() returns a blank date. If no table/.DBF has the alias you specify, the message “Alias not found” is displayed.

Example

```
USE customer
? LUPDATE( )
```

MAX()

Purpose	Returns the expression with the highest ASCII or numeric value or the latest data from a list of character, numeric or date expressions
Syntax	MAX(<expr1>, <expr2> [, <expr3> ...])
Parameters	<expr1>, <expr2> [,<expr3>] A set of expressions of the same type
Returns	Character, Date or Numeric
See Also	CALCULATE, MIN()

Description MAX() (maximum) evaluates a set of expressions and returns the expression with the largest value.

When determining the maximum date value from a set of dates, MAX() returns the latest date.

MAX() returns the character expression with the maximum ASCII value from a set of character expressions.

Parameters <expr1>, <expr2> [, <expr3> ...]

<expr1>, <expr2>, and <expr3> are the set of expressions from which you want MAX() to return the expression with the highest value.

All the expressions must be of the same type (character, numeric or date).

Examples ? MAX(54, 39, 40)
54

 ? MAX(2^8, 10*12, PI())
256.00

 ? MAX({07/12/99},DATE())
07/12/99

MCOL()

Purpose	Returns the screen or window column position of the mouse pointer
Syntax	MCOL([<expC>])
Returns	Numeric
See Also	COL(), MROW(), ROW(), WCOLS(), WROWS()

Description MCOL() returns the column position of the mouse pointer on the desktop in FoxPro for MS-DOS, the main FoxPro window in FoxPro for Windows, or a user-defined window.

Parameter <expC>

MCOL() returns the desktop or main FoxPro window column position of the mouse pointer if <expC> is omitted and there is no active user-defined window.

If there is an active user-defined window and <expC> is omitted, MCOL() returns the mouse pointer column coordinate relative to the active user-defined window. If you specify the window name with <expC>, MCOL returns the column coordinate relative to the specified window. MCOL() returns -1 if the mouse pointer is positioned outside the user-defined window.

In FoxPro for MS-DOS, MCOL() returns 0 if <expC> is the name of a Browse window or the Debug window and the mouse pointer is positioned on the window's left border. MCOL() returns -1 if the mouse pointer is positioned on the left border of any other windows.



MCOL() returns -1 if a mouse driver isn't loaded and there is no output window.

Example See the second example for INKEY() earlier in this chapter.

MDOWN()

Purpose	Determines whether or not the left mouse button is pressed
Syntax	MDOWN()
Returns	Logical
See Also	MCOL(), MROW()

Description MDOWN() returns a true (.T.) or false (.F.) value depending on whether or not the left mouse button is pressed.

If the left mouse button is depressed when MDOWN() is executed, true (.T.) is returned. If the left mouse button isn't depressed when MDOWN() is executed, false (.F.) is returned.

MDX()

Purpose	Returns the name of the open .CDX compound index file that has the specified index position number
Syntax	MDX(<expN1> [, <expN2> <expC>])
Parameters	<expN1> A number corresponding to an index position that tells MDX() what to return <expN2> The work area number for a table/.DBF <expC> The table/.DBF alias
Returns	Character
See Also	CDX(), INDEX, NDX(), SET INDEX, SET FULLPATH, SYS(14), TAG(), USE

Description MDX() is identical to CDX().

Index files can be opened for a table/.DBF with the INDEX clause of the USE command or with SET INDEX. A structural compound index file is automatically opened with its table/.DBF. Any .IDX index files specified with USE or in SET INDEX are ignored by this function.

TAG() can be used to return tag names from a compound index file; NDX() can be used to return the name of an open .IDX index file.

Parameters <expN1>

If the table/.DBF has a structural compound index file and <expN1> is 1, the name of the structural compound index file (which is always the same as the name of the table/.DBF) is returned. If <expN1> is 2, the name of the first compound index file specified with USE or SET INDEX is returned. If <expN1> is 3, the second compound index file name is returned, and so on. If <expN1> is greater than the number of open compound index files, the null string is returned.

If the table/.DBF doesn't have a structural compound index file and <expN1> is 1, the name of the first compound index file specified with USE or SET INDEX is returned. If <expN1> is 2, the second compound index file name is returned, and so on. If <expN1> is greater than the number of open compound index files, the null string is returned.

<expN2> | <expC>

If you omit the optional <expN2> and <expC> arguments, names of compound index files are returned for the current work area. You can return the names of compound index files open in work areas other than the current one by specifying the work area number <expN2> or the table/.DBF alias <expC>.

MDY()

Purpose	Returns the specified date expression in Month Day Year format
Syntax	MDY(<expD>)
Returns	Character
See Also	DMY(), SET CENTURY, SET DATE

Description MDY() returns a date expression in Month Day Year format with the name of the month spelled out. If SET CENTURY OFF, the resulting character expression has the format Month DD, YY. If CENTURY is SET ON, the format is Month DD, YYYY.

Parameter <expD>
 <expD> is the date expression you want to return in Month Day Year format.

Example The following example creates a user-defined function that returns a date with the corresponding weekday.

```
*** LongDate ***
```

```
FUNCTION longdate
PARAMETERS mdate
RETURN CDOW(mdate) + ', ' + MDY(mdate)
```

To use this UDF:

```
SET CENTURY OFF
```

```
? Longdate({08/16/92})
```

Sunday, August 16, 92

```
SET CENTURY ON
```

```
? Longdate({08/16/92})
```

Sunday, August 16, 1992

MEMLINES()

Purpose	Returns the number of lines in a memo field
Syntax	MEMLINES(<memo field>)
Returns	Numeric
See Also	MLINE(), SET MEMOWIDTH

Description MEMLINES() returns the number of lines in a memo field. The number of lines in a memo field is determined by the current value of SET MEMOWIDTH.

Parameter <memo field>

<memo field> is the name of the memo field. If the memo field is in a table/.DBF that isn't open in the current work area, preface the memo field name with the table/.DBF alias and a period.

Example The following example scans three records in the SALESMAN table/.DBF and uses MEMLINES() to determine whether or not there is data in the NOTES memo field and when page breaks should occur. The NAME data for the record is displayed, along with the notes (if there is data in the memo field) or a message indicating that there are no notes for that record.

```
CLOSE DATABASES
CLEAR
SET TALK OFF
USE salesman
SET MEMOWIDTH TO 65
mline = 1
GOTO 2
```

```

SCAN NEXT 3
  memo_size = MEMLINES(notes)
  IF memo_size = 0
    STORE .T. TO nomemo
    STORE 1 TO memo_size
  ELSE
    STORE .F. TO nomemo
  ENDIF
  IF mline + memo_size > 65
    EJECT
    mline = 1
  ENDIF
  @ mline,2 SAY 'Name : ' + name
  mline = mline+1
  @ mline,2 SAY 'Notes: '
  ?? IIF(nomemo, 'No notes ',notes)
  mline = mline + memo_size + 2
  IF mline > 24
    mline = 1
    CLEAR
  ENDIF
ENDSCAN

```

MEMORY()

Purpose	Returns the amount of available memory to run an external program
Syntax	MEMORY()
Returns	Numeric
See Also	SYS(12), SYS(1001)

Description In FoxPro for MS-DOS, MEMORY() returns the amount of memory below 640KB which is available to RUN an external program. Any available expanded memory is not included in the value returned by this function, and the value will vary throughout a FoxPro for MS-DOS session.

In FoxPro for Windows, MEMORY() always returns 640.

MEMORY() is similar to SYS(12), with two exceptions:

- MEMORY() returns the amount of available memory in kilobytes; SYS(12) returns the amount of memory in bytes.
- MEMORY() returns a numeric expression. SYS(12) returns its value as a character string.

Example ? MEMORY()

MENU — System Menu Names

Purpose	The FoxPro System Menu names
See Also	ACTIVATE MENU, ACTIVATE POPUP, DEFINE BAR, DEFINE MENU, DEFINE POPUP, SET SYSMENU, SYS(2013)

Description The following tables provide the names of system menu pads, menu popups and menu options.

These names let you use the FoxPro system menu bar. The SYS(2013) function returns a space-delimited string containing these names. You can also create a quick menu in the Menu Builder to see the system menu names. For further information on the system menu bar, see the Menus chapter in the *FoxPro Developer's Guide* or the Menu Builder chapter in the *FoxPro User's Guide*.

The name of the FoxPro system menu bar is _MSYSMENU.

System Menu Pad Names	
Menu Pad	Name
System (FoxPro for MS-DOS) Help (FoxPro for Windows)	_MSM_SYSTM
File	_MSM_FILE
Edit	_MSM_EDIT
Database	_MSM_DATA
Record	_MSM_RECRD
Program	_MSM_PROG
Text (FoxPro for Windows only)	_MSM_TEXT
Window	_MSM_WINDO

System Menu Popup Names FoxPro for MS-DOS	
Popup or Option	Name
System popup	_MSYSTEM
About FoxPro...	_MST_ABOUT
Help...	_MST_HELP
Macros...	_MST_MACRO
First separator	_MST_SP100
Filer	_MST_FILER
Calculator	_MST_CALCUL
Calendar/Diary	_MST_DIARY
Special Characters	_MST_SPECL
ASCII Chart	_MST_ASCII
Capture	_MST_CAPTR
Puzzle	_MST_PUZZL

File Menu Popup Names FoxPro for MS-DOS	
Popup or Option	Name
File popup	_MFILE
New...	_MFI_NEW
Open...	_MFI_OPEN
Close	_MFI_CLOSE
Close All	_MFI_CLALL
First separator	_MFI_SP100
Save	_MFI_SAVE
Save as...	_MFI_SAVAS
Revert	_MFI_REVRT
Second separator	_MFI_SP200
Printer Setup...	_MFI_SETUP
Print...	_MFI_PRINT
Third separator	_MFI_SP300
Quit	_MFI_QUIT

Edit Menu Popup Names FoxPro for MS-DOS	
Popup or Options	Name
Edit popup	_MEDIT
Undo	_MED_UNDO
Redo	_MED_REDO
First separator	_MED_SP100
Cut	_MED_CUT
Copy	_MED_COPY
Paste	_MED_PASTE
Clear	_MED_CLEAR
Second separator	_MED_SP200
Select All	_MED_SLCTA
Third separator	_MED_SP300
Goto Line...	_MED_GOTO
Find...	_MED_FIND
Find Again	_MED_FINDA
Replace And Find Again	_MED_REPL
Replace All	_MED_REPLA
Fourth separator	_MED_SP400
Preferences...	_MED_PREF

Database Menu Popup Names FoxPro for MS-DOS	
Popup or Options	Name
Database popup	_MDATA
Setup..	_MDA_SETUP
Browse	_MDA_BROW
First separator	_MDA_SP100
Append From...	_MDA_APPND
Copy To...	_MDA_COPY
Sort...	_MDA_SORT
Total...	_MDA_TOTAL
Second separator	_MDA_SP200
Average...	_MDA_AVG
Count...	_MDA_COUNT
Sum...	_MDA_SUM
Calculate...	_MDA_CALC
Report...	_MDA_REPRT
Label...	_MDA_LABEL
Third separator	_MDA_SP300
Pack	_MDA_PACK
Reindex	_MDA_RINDX

Record Menu Popup Names FoxPro for MS-DOS	
Popup or Option	Name
Record popup	_MRECORD
Append	_MRC_APPND
Change	_MRC_CHNGE
First separator	_MRC_SP100
Goto...	_MRC_GOTO
Locate...	_MRC_LOCAT
Continue	_MRC_CONT
Seek...	_MRC_SEEK
Second separator	_MRC_SP200
Replace...	_MRC_REPL
Delete...	_MRC_DELETE
Recall...	_MRC_RECAL

Program Menu Popup Names FoxPro for MS-DOS	
Popup or Option	Name
Program popup	_MPROG
Do...	_MPR_DO
First separator	_MPR_SP100
Cancel	_MPR_CANCL
Resume	_MPR_RESUM
Second separator	_MPR_SP200
Compile...	_MPR_COMPL
Generate...	_MPR_GENER
FoxDoc	_MPR_DOCUM
FoxGraph...	_MPR_GRAPH

Window Menu Popup Names FoxPro for MS-DOS	
Popup or Option	Name
Window popup	_MWINDOW
Hide	_MWI_HIDE
Hide All	_MWI_HIDEA
Show All	_MWI_SHOWA
Clear	_MWI_CLEAR
First separator	_MWI_SP100
Move	_MWI_MOVE
Size	_MWI_SIZE
Zoom ↑	_MWI_ZOOM
Zoom ↓	_MWI_MIN
Cycle	_MWI_ROTAT
Color...	_MWI_COLOR
Second separator	_MWI_SP200
Command	_MWI_CMD
Debug	_MWI_DEBUG
Trace	_MWI_TRACE
View	_MWI_VIEW

File Menu Popup Names FoxPro for Windows	
Popup or Option	Name
File popup	_MFILE
New...	_MFI_NEW
Open...	_MFI_OPEN
Close	_MFI_CLOSE
Close All	_MFI_CLALL
First separator	_MFI_SP100
Save	_MFI_SAVE
Save As...	_MFI_SAVAS
Revert	_MFI_REVRT
Second separator	_MFI_SP200
Print Setup...	_MFI_SETUP
Print...	_MFI_PRINT
Third separator	_MFI_SP300
Exit	_MFI_QUIT

Edit Menu Popup Names FoxPro for Windows	
Popup or Option	Name
Edit popup	_MEDIT
Undo	_MED_UNDO
Redo	_MED_REDO
First separator	_MED_SP100
Cut	_MED_CUT
Copy	_MED_COPY
Paste	_MED_PASTE
Paste Special...	_MED_PSTLK
Clear	_MED_CLEAR
Second separator	_MED_SP200
Insert Object...	_MED_INSOB
Object...	_MED_OBJ
Change Link...	_MED_LINK
Convert To Static	_MED_CVTST
Third separator	_MED_SP300
Select All	_MED_SLCTA
Fourth separator	_MED_SP400
Goto Line...	_MED_GOTO
Find...	_MED_FIND
Find Again	_MED_FINDA
Replace And Find Again	_MED_REPL
Replace All	_MED_REPLA
Fifth separator	_MED_SP500
Preferences...	_MED_PREF

Database Menu Popup Names FoxPro for Windows	
Popup or Option	Name
Database popup	_MDATA
Setup...	_MDA_SETUP
Browse	_MDA_BROW
First separator	_MDA_SP100
Append From...	_MDA_APPND
Copy To...	_MDA_COPY
Sort...	_MDA_SORT
Total...	_MDA_TOTAL
Second separator	_MDA_SP200
Average...	_MDA_AVG
Count...	_MDA_COUNT
Sum...	_MDA_SUM
Calculate...	_MDA_CALC
Report...	_MDA_REPRT
Label...	_MDA_LABEL
Third separator	_MDA_SP300
Pack	_MDA_PACK
Reindex	_MDA_RINDX

Record Menu Popup Names FoxPro for Windows	
Popup or Option	Name
Record popup	_MRECORD
Append	_MRC_APPND
Change	_MRC_CHNGE
First separator	_MRC_SP100
Goto...	_MRC_GOTO
Locate...	_MRC_LOCAT
Continue	_MRC_CONT
Seek...	_MRC_SEEK
Second separator	_MRC_SP200
Replace...	_MRC_REPL
Delete...	_MRC_DELETE
Recall...	_MRC_RECAL

Program Menu Popup Names FoxPro for Windows	
Popup or Option	Name
Program popup	_MPROG
Do...	_MPR_DO
Cancel	_MPR_CANCL
Resume	_MPR_RESUM
First separator	_MPR_SP100
Debug	_MWI_DEBUG
Trace	_MWI_TRACE
Second separator	_MPR_SP200
Compile...	_MPR_COMPL
Generate...	_MPR_GENER
Macros...	_MST_MACRO
Third separator	_MPR_SP300
Beautify...	_MPR_BEAUT
FoxDoc	_MPR_DOCUM

Window Menu Popup Names FoxPro for Windows	
Popup or Option	Name
Window popup	_MWINDOW
Hide	_MWI_HIDE
Hide All	_MWI_HIDEA
Show All	_MWI_SHOWA
Clear	_MWI_CLEAR
Cycle	_MWI_ROTAT
First separator	_MWI_SP100
Command	_MWI_CMD
View	_MWI_VIEW

Help Menu Popup Names FoxPro for Windows	
Popup or Option	Name
Help popup	_MSYSTEM
Contents	_MST_HELP
Search for Help on...	_MST_HPSCH
How to Use Help	_MST_HPHOW
First separator	_MST_SP100
About FoxPro...	_MST_ABOUT
Second separator	_MST_SP200
Calculator	_MST_CALC
Calendar/Diary	_MST_DIARY
Filer	_MST_FILER
Puzzle	_MST_PUZZL

MENU

Purpose	Creates a menu system
Syntax	MENU BAR <array1>, <expN1> MENU <expN2>, <array2>, <expN3> [, <expN4>] READ MENU BAR TO <memvar1>, <memvar2> [SAVE]
Remarks	Included for backward compatibility. Use the FoxPro Menu Builder to create menus.
See Also	@ ... MENU, CREATE MENU, DEFINE MENU

Description These commands create a menu system. A *menu system* consists of a menu bar across the top of the desktop in FoxPro for MS-DOS, the main FoxPro window in FoxPro for Windows, or the active user-defined window menu pads. Each menu pad has an associated menu popup containing a list of options.

Commands **MENU BAR <array1>, <expN1>**

MENU BAR is used to create the menu bar and pads. Create a two-dimensional array of memory variables to contain the menu pad prompts and the messages that are displayed when the menu pads are selected. The elements in the first column of the array specified with <array1> are the menu pad prompts, and the elements in the second column of the array are the messages. Specify where the messages are displayed with SET MESSAGE.

The number of menu pads that appear in the menu bar is specified with <expN1>.

MENU <expN2>, <array2>, <expN3> [, <expN4>]

Use a series of MENU commands to create the menu popups that are displayed when a menu pad is selected.

The numeric expression <expN2> specifies which menu pad in the menu bar the popup is assigned to. The one-dimensional array <array2> contains the prompts for the options on the popups.

If the first character of an array element is a backslash (\), the corresponding option can't be selected. If the element contains a backslash and a hyphen (\-), a separator is drawn between the popup option prompts. You can create a hot key for an option by placing a backslash and a less-than sign (\<) before the desired character of the option prompt. A hot key is a highlighted letter in FoxPro for MS-DOS or an underlined letter in FoxPro for Windows that you can type to immediately choose the desired option.

The number of options on the popup is specified with <expN3>.

The number of options displayed on the popup is specified with <expN4>. If there are more options on the popup than can be displayed at one time, you can scroll through the options in the popup.

READ MENU BAR TO <memvar1>, <memvar2>

This command activates the menu bar. The values initially contained in the memory variables or array elements <memvar1> and <memvar2> specify which menu pad and popup option are selected when the menu bar is activated. When you choose an option from a popup, the values stored to <memvar1> and <memvar2> correspond to the popup and popup option you chose. If you press Escape to exit the menu, <memvar1> and <memvar2> contain 0.

SAVE

Include SAVE to display an image of the menu after an option is selected.

Example

*** A program example of popup MENUS ***

```
SET STATUS OFF
SET ECHO OFF
SET TALK OFF
SET MESSAGE TO 24
```

*** Initialize the array for the menu bar ***

```
DIMENSION TOP(3,2)
TOP(1,1) = '  FILE  '
TOP(2,1) = '  EDIT  '
TOP(3,1) = '  DATA  '
TOP(1,2) = 'FILE SELECTIONS'
TOP(2,2) = 'EDIT SELECTIONS'
TOP(3,2) = 'DATA SELECTIONS'
```


*** Dimension and initialize an array for the File menu ***

```
DIMENSION M_FILE(7)
M_FILE(1) = '\<NEW'
M_FILE(2) = '\<OPEN'
M_FILE(3) = '\<CLOSE'
M_FILE(4) = '\-'      && Insert a line between items
M_FILE(5) = '\<SAVE'
M_FILE(6) = '\<PRINT'
M_FILE(7) = '\<QUIT'
```

*** Dimension and initialize an array for the Edit menu ***

```
DIMENSION M_EDIT(5)
M_EDIT(1) = '\<COPY'
M_EDIT(2) = '\<FIND'
M_EDIT(3) = '\<REPLACE'
M_EDIT(4) = '\<UNDO'
M_EDIT(5) = 'C\<LEAR'
```

*** Dimension and initialize an array for the Data menu ***

```
DIMENSION M_DATA(9)
M_DATA(1) = '\<BROWSE'
M_DATA(2) = '\<APPEND'
M_DATA(3) = '\<SORT'
M_DATA(4) = '\<JOIN'
M_DATA(5) = '\-'      && Insert a line between menu items
M_DATA(6) = '\<TOTAL'
M_DATA(7) = 'A\<VERAGE'
M_DATA(8) = '\<COUNT'
M_DATA(9) = 'S\<UM'
```

*** Install the pull-down menu system

```
MENU BAR TOP,3      && Install menu bar
MENU 1,M_FILE,7,7 && Install file menu into menu bar
MENU 2,M_EDIT,5,5 && Install edit menu into menu bar
MENU 3,M_DATA,9,9 && Install data menu into menu bar
```

*** Initialize the placement of the menu option ***

```
COL = 1
ROW = 1
```

*** Activate the pull-down menu system ***

```
READ MENU BAR TO COL, ROW SAVE
CLEAR
```

MENU TO

Purpose	Activates a menu bar
Syntax	MENU TO <memvar>
Remarks	Included for backward compatibility. Use the FoxPro Menu Builder to create menus.
See Also	@ ... PROMPT, CREATE MENU, DEFINE MENU

Description Use MENU TO to activate menu popups created with @ ... PROMPT.

Clause <memvar>

The value initially contained in the memory variable or array element <memvar> specifies which menu pad is selected when the menu bar is activated. When you choose a menu pad, a value is stored to <memvar> corresponding to the menu pad you choose. If you press Escape to exit the menu, <memvar> contains 0.

MENU()

Purpose	Returns the name of the active menu bar
Syntax	MENU()
Returns	Character
See Also	ACTIVATE MENU, CREATE MENU, DEFINE MENU

Description MENU() returns the name of the active menu bar as an upper-case character string. MENU() returns the null string if no menu is active. Use DEFINE MENU and ACTIVATE MENU to create a menu and activate it.

Example This example uses MENU() to pass the name of a menu bar to a procedure. The current system menu bar is saved to memory with SET SYSMENU SAVE, and all system menu pads are removed with SET SYSMENU TO.

Several system menu pads are created with DEFINE PAD. When you choose a menu pad, MENU() passes the name of the FoxPro system menu bar, _MSYSMENU, to the CHOICE procedure. CHOICE displays the name of the menu pad you chose and the name of the system menu bar. If you choose the **Exit** menu pad, the original FoxPro system menu is restored.

```
*** Name this program MENUEXAM.PRG ***
```

```
CLEAR
```

```
SET SYSMENU SAVE
SET SYSMENU TO
```

```
DEFINE PAD syspad    OF _MSYSMENU PROMPT '\<System'  COLOR SCHEME 3 ;
    KEY ALT+S, ''
DEFINE PAD editpad   OF _MSYSMENU PROMPT '\<Edit'     COLOR SCHEME 3 ;
    KEY ALT+E, ''
DEFINE PAD recordpad OF _MSYSMENU PROMPT '\<Record'   COLOR SCHEME 3 ;
    KEY ALT+R, ''
DEFINE PAD windowpad OF _MSYSMENU PROMPT '\<Window'   COLOR SCHEME 3 ;
    KEY ALT+W, ''
DEFINE PAD reportpad OF _MSYSMENU PROMPT 'Re\<ports'  COLOR SCHEME 3 ;
    KEY ALT+P, ''
DEFINE PAD exitpad   OF _MSYSMENU PROMPT 'E\<xit'     COLOR SCHEME 3 ;
    KEY ALT+X, ''
```

MENU()

```
ON SELECTION MENU _MSYSMENU ;
    DO choice IN menuexam WITH PAD( ), MENU( )

PROCEDURE choice
PARAMETER mpad, mmenu

WAIT WINDOW 'You chose ' + mpad + ;
    ' from menu ' + mmenu NOWAIT

IF mpad = 'EXITPAD'
    SET SYSMENU TO DEFAULT
ENDIF
```

MESSAGE()

Purpose	Returns the current error message string or the contents of the program line that caused the error
Syntax	MESSAGE([1])
Returns	Character
See Also	ERROR(), ON ERROR

Description MESSAGE() returns the last error message as a character string. Unlike ERROR(), MESSAGE() isn't reset by RETURN or RETRY.

Parameter 1

If MESSAGE() is used in a routine created with ON ERROR, you can include the optional argument 1 to return the program source code that caused the error. If the program source code isn't available, MESSAGE(1) returns one of the following:

- The entire program line if the line is macro substituted
- A command if the line contains a command without any additional clauses
- A command followed by three dots (...) if the line contains a command and additional clauses

Example This example displays output from MESSAGE() and MESSAGE(1).

```
ON ERROR DO Errhand

*** The next line should generate an error ***

USE Nodatabase

ON ERROR    && restore system error handler

PROCEDURE Errhand
? 'Line of code with error: ' + MESSAGE(1)
? 'Error number: ' + STR(ERROR( ))
? 'Error message: ' + MESSAGE( )
```

MIN()

Purpose	Returns the expression with the lowest ASCII or numeric value or the earliest date in a list of character, numeric or date expressions
Syntax	MIN(<expr1>, <expr2> [, <expr3> ...])
Returns	Character, Date or Numeric
See Also	CALCULATE, MAX()

Description MIN() (minimum) evaluates a set of expressions and returns the expression with the smallest value.

When determining the minimum date value from a set of dates, MIN() returns the earliest date.

MIN() returns the character expression with the minimum ASCII value from a set of character expressions.

Parameters <expr1>, <expr2> [, <expr3> ...]

<expr1>, <expr2>, and <expr3> are the set of expressions from which you want MIN() to return the expression with the lowest value.

All the expressions must be of the same type (character, numeric or date).

Examples ? MIN(54, 39, 40)
39

? MIN(2^8, 10*12, PI())
3.14

? MIN({07/12/18},DATE())
07/12/18

? MIN('a', 'abc')
a

? MIN('a', 'XYZ')
XYZ

MLINE()

Purpose	Returns a specific line from a memo field as a character string
Syntax	MLINE(<memo field>, <expN1> [, <expN2>]
Parameters	<memo field> The memo field from which a line is returned <expN1> The number of the line to return <expN2> The offset from the beginning of the memo field
Returns	Character
See Also	ATCLINE(), ATLINE(), COPY MEMO, MEMLINES(), _MLINE, MODIFY MEMO, SET MEMOWIDTH

Description MLINE() returns a specific line from a memo field. The length and number of the lines in a memo field are determined by the current value of SET MEMOWIDTH (the default line length is 50 characters).

When searching a memo field for a character string, you can use ATLINE() or ATCLINE() to return the line number of the line at which the character string is found. Use this line number in MLINE() to return the contents of the line from the memo field.

Parameters <memo field>

MLINE() returns a line from the memo field whose name you specify with <memo field>. If the memo field is in a table/.DBF open in a noncurrent work area, preface the memo field with a period and the table/.DBF alias.

<expN1>

<expN1> is the number of the line to be returned from the memo field. The null string is returned if <expN1> is negative, 0 or greater than the number of lines in the memo field.

<expN2>

<expN2> is the offset from the beginning of the memo field. If you include <expN2>, the <expN1>th line from offset <expN2> is returned.

The _MLINE system memory variable is typically used for <expN2>. _MLINE is automatically adjusted each time MLINE() is called.

In recursive procedures that return lines from large memo fields, you can obtain the best performance by including _MLINE as <expN2>.

Example

In the example below, two methods are used to return lines from a memo field. Two loops use MLINE() to return lines from the memo field. Note the improvement in performance in the second loop when the system memory variable _MLINE is used in MLINE().

```

CLEAR
SET TALK OFF
SET MEMOWIDTH TO 50
CLOSE DATABASES
CREATE TABLE tmemo (name c(10), notes m)
APPEND BLANK    && Add a record
WAIT WINDOW 'Filling memo field - takes several seconds' NOWAIT

*** Fill the memo field ***
FOR outerloop = 1 TO 5    && loop 5 times
    FOR alphabet = 65 TO 90    && letters A to Z
        REPLACE notes WITH REPLICATE(CHR(alphabet), 40) ;
            + CHR(13) ADDITIVE
    NEXT
NEXT

*** Display all lines from the memo field ***
STORE MEMLINES(notes) TO numlines    && Number of lines in memo
field
STORE SECONDS( ) TO begin    && Beginning time
FOR count = 1 TO numlines    && Loop for number of lines in memo field
    ?MLINE(notes, count)    && Display each line
NEXT
? STR(SECONDS( ) - begin, 4, 2) + ' seconds'&& Total time

*** Preferable method using _MLINE in MLINE( ) ***
*** Display all lines from the memo field ***

WAIT 'Press a key to see the preferred method' WINDOW
CLEAR
STORE 0 TO _MLINE    && Reset _MLINE to zero
STORE SECONDS( ) TO begin    && Beginning time
FOR count = 1 TO numlines    && Loop for # of lines in memo field
    ?MLINE(notes, 1, _MLINE) && Display each line
NEXT
? STR(SECONDS( ) - begin, 4, 2) + ' seconds'&& Total time
SET TALK ON
CLOSE DATABASES
ERASE tmemo.dbf
ERASE tmemo.fpt

```


MOD()

Purpose	Divides one numeric expression by another numeric expression and returns the remainder
Syntax	MOD(<expN1>, <expN2>)
Parameters	<expN1> The dividend (the number being divided) <expN2> The divisor (the number the dividend is divided by)
Returns	Numeric
See Also	%

Description MOD() (modulus) divides two numeric expressions and returns the remainder. MOD() and the % operator return identical results.

Parameters <expN1>

<expN1> is the dividend. The number of decimal places in <expN1> determines the number of decimals places in the result.

<expN2>

<expN2> is the divisor. A positive number is returned if <expN2> is positive, and a negative number is returned if <expN2> is negative.

Examples

? MOD(36, 10)
6

? MOD(4*9, 90/9)
6

? MOD(25.250, 5.0)
0.250

MODIFY COMMAND / MODIFY FILE

Purpose	Opens a text editing window and allows you to modify or create a program or text file
Syntax	MODIFY COMMAND [<file> ?] MODIFY FILE [<file> ?] [NOEDIT] [NOWAIT] [RANGE <expN1>, <expN2>] [[WINDOW <window name1>] [IN [WINDOW] <window name2> SCREEN]] [SAME] [SAVE]
Remarks	SAME clause — FoxPro for Windows only
See Also	DO, NOTE * &&

Description For more information on the FoxPro text editing window opened with MODIFY COMMAND and MODIFY FILE, see the Text Editor section of the Interface Basics chapter in the FoxPro *User's Guide*.

When modifications are made to a program or text file, the updated file is written to disk. A backup file with a .BAK extension is created if you check the **Backup** check box in the Preferences dialog that appears when you choose **Preferences...** from the **Edit** menu popup.

The built-in FoxPro editor is used unless you specify an external editor with TEDIT in your configuration file.

Clauses **MODIFY COMMAND [<file> | ?]**

Use MODIFY COMMAND to edit program files. If <file> is a file name without an extension, a .PRG extension is assumed. MODIFY COMMAND supports a file skeleton that can contain the wildcard characters * and ?. A text editing window is opened for each program whose file name matches the file skeleton.

If you omit the file name, a text editing window opens for a file initially named UNTITLED.PRG. When you close the editing window you can save the file with a different name.

If you include ?, the Open dialog appears, displaying a list of existing .PRG (program), .MPR (menu program), .SPR (screen program) and .QPR (query program) files to choose from.

MODIFY FILE [<file> | ?]

Use MODIFY FILE to edit text files. If <file> is a file name without an extension, no extension is assumed. MODIFY FILE supports a file skeleton that can contain the wildcard characters * and ?. A text editing window is opened for each text file whose name matches the file skeleton.

If you omit the file name, an editing window opens for a file initially named UNTITLED. When you close the editing window you can save the file with a different name.

If you include ?, the Open dialog appears, displaying a list of existing files with no extension and .TXT files to choose from.

NOEDIT

If you include NOEDIT, the program or text file opened with MODIFY COMMAND or MODIFY FILE can't be changed but can be viewed and copied to the clipboard.

NOWAIT

When you issue MODIFY COMMAND or MODIFY FILE from within a program, a text editing window is opened and program execution pauses until the text editing window is closed. Include NOWAIT to continue program execution after the text editing window is opened. The program continues executing on the program line immediately following MODIFY COMMAND NOWAIT or MODIFY FILE NOWAIT.

An implicit NOWAIT occurs if you open more than one text editing window with a single MODIFY COMMAND or MODIFY FILE command. For example:

```
MODIFY COMMAND *.PRG.
```

RANGE <expN1>, <expN2>

RANGE lets you open a text editing window with a range of characters selected. Characters are selected starting at the position specified with <expN1> up to (but not including) the <expN2>th character. If <expN1> is equal to <expN2>, no characters are selected, and the cursor is placed at the position specified with <expN1>.

WINDOW <window name1>

If you include WINDOW <window name1>, the text editing window takes on the characteristics of the window you specify. For example, if the window is defined with the FLOAT clause of DEFINE WINDOW, the text editing window can be moved. The window you specify with <window name1> doesn't have to be active or visible, but it must be defined.

IN [WINDOW] <window name2>

If you include IN WINDOW <window name2>, the text editing window is opened within a parent window without assuming the characteristics of the parent window. A text editing window activated inside a parent window can't be moved outside the parent window. If the parent window is moved, the text editing window moves with it.

To access the text editing window, the parent window must first be defined with DEFINE WINDOW and must be visible.

IN SCREEN

By including IN SCREEN, you can explicitly place a text editor window on the desktop in FoxPro for MS-DOS or in the main window in FoxPro for Windows when a user-defined window is active. Text editor windows are placed on the desktop or main FoxPro window by default.

SAME — FoxPro for Windows only

Include SAME to prevent the text editing window from coming forward as the active window. If the text editing window is hidden, it is displayed but doesn't become the active window.

SAME is ignored in FoxPro for MS-DOS.

SAVE

Include SAVE to keep an image of a text editing window on the desktop, the main FoxPro window or a user-defined window after the text editing window is closed. Text editing windows are normally removed from the desktop, the main FoxPro window or a user-defined window after you close them. Use CLEAR to erase a text editor window from the desktop, the main FoxPro window or a user-defined window.

SAVE is available only from within a program. Including SAVE has no effect when it is issued in the Command window.

MODIFY GENERAL

Purpose	Opens an editing window for a general field containing an OLE object
Syntax	MODIFY GENERAL <general field1> [, <general field2> ...] [NOMODIFY] [NOWAIT] [WINDOW <window name1>] [IN [WINDOW] <window name2> SCREEN]
Remarks	FoxPro for Windows only
See Also	@ ... SAY, APPEND GENERAL
Description	MODIFY GENERAL opens an editing window for a general field for the current table/DBF record. When the window is open, you can insert, modify or delete linked or embedded OLE objects. For additional information about OLE objects in FoxPro for Windows, see the Object Linking and Embedding (OLE) chapter in the FoxPro for Windows <i>Developer's Guide</i>.
Clauses	<general field1> [, <general field2> ...] Specify the names of the general fields to open with <general field1>, <general field2>, and so on. You can open an editing window for a general field in a table/DBF open in a noncurrent work area by including the table/DBF alias with the field name. You can open multiple general fields by including a list of general fields separated by commas. NOEDIT If you include NOEDIT, the OLE object contained in the general field can't be changed, but it can be viewed and copied to the clipboard.

NOWAIT

When you issue **MODIFY GENERAL** from within a program, the general field editing window is opened and program execution pauses until the editing window is closed. Include **NOWAIT** to continue program execution after the general field editing window is opened. The program continues executing on the program line immediately following **MODIFY GENERAL NOWAIT**.

NOWAIT is available only in a program. Including **NOWAIT** has no effect on **MODIFY GENERAL** when it is issued in the Command window.

WINDOW <window name1>

If you include **WINDOW <window name1>**, the general field editing window takes on the characteristics of the window you specify with **<window name1>**. For example, if the window is defined with the **FLOAT** clause of **DEFINE WINDOW**, the general field window can be moved. The window you specify doesn't have to be active or visible, but it must be defined.

IN [WINDOW] <window name2>

If you include **IN WINDOW <window name2>**, the general field editing window is opened within a parent window without assuming the characteristics of the parent window. A general field editing window activated inside a parent window can't be moved outside the parent window. If the parent window is moved, the editing window moves with it.

To access the general field editing window, the parent window must first be defined with **DEFINE WINDOW** and must be visible.

IN SCREEN

By including **IN SCREEN**, you can explicitly place a general field editing window in the main window in FoxPro for Windows when a user-defined window is active. A general field editing window is placed in the main FoxPro window by default.

MODIFY LABEL

Purpose	Modifies or creates labels and saves them to a label definition file
Syntax	MODIFY LABEL [<file> ?] [[WINDOW <window name1>] [IN [WINDOW] <window name2> IN SCREEN]] [NOENVIRONMENT] [NOWAIT] [SAVE]
See Also	CREATE LABEL, LABEL

Description MODIFY LABEL lets you interactively modify existing labels or create new labels in the Label Design window in FoxPro for MS-DOS or the Report Layout window in FoxPro for Windows.

Issuing MODIFY LABEL without any additional arguments displays the Open dialog. If you choose **New** from this dialog, the name Untitled is assigned to the Label Design or Report Layout window. You can save the label file with a different name when you close the Label Design or Report Layout window.

Clauses <file>
You can modify existing labels by specifying the file name of the label definition table/.DBF with <file>. If the file you specify doesn't exist or can't be found, a new label file is created.

?

If you include the ? clause or omit a label file name, the Open dialog appears and displays a list of existing label files to choose from. You can also choose **New** to create a new label file.

WINDOW <window name1>
If you include WINDOW <window name1>, the Label Design or Report Layout window takes on the characteristics of the window you specify with <window name1>. For example, if the window is defined with the FLOAT clause of DEFINE WINDOW, the Label Design or Report Layout window can be moved. The window you specify doesn't have to be active or visible, but it must be defined.

IN [WINDOW] <window name2>

If you include IN WINDOW <window name2>, the Label Design or Report Layout window is opened within a parent window without assuming the characteristics of the parent window. A Label Design or Report Layout window activated inside a parent window can't be moved outside the parent window. If the parent window is moved, the Label Design or Report Layout window moves with it.

To access the Label Design or Report Layout window, the parent window must first be defined with DEFINE WINDOW and must be visible.

IN SCREEN

By including IN SCREEN, you can explicitly place the Label Design window on the desktop in FoxPro for MS-DOS or the Report Layout window in the main window in FoxPro for Windows when a user-defined window is active. The Label Design window is placed on the desktop and the Report Layout window is placed in main FoxPro window by default.

NOENVIRONMENT

When you close the Label Design or Report Layout window, you are prompted to save the current FoxPro environment with the label definition file. Saving the FoxPro environment places an additional record in the label definition table/.DBF. This record contains the names of all open table/.DBF and index files, the index order, and any relationships between the tables/.DBFs.

MODIFY LABEL automatically restores the environment saved with the label definition file if you saved the FoxPro environment. If you include NOENVIRONMENT, the environment isn't restored. Include NOENVIRONMENT to prevent changes to the current environment.

NOWAIT

When you issue MODIFY LABEL from within a program, the Label Design or Report Layout window is opened and program execution pauses until the Label Design or Report Layout window is closed. Include NOWAIT to continue program execution after the Label Design or Report Layout window is opened. The program continues executing on the program line immediately following MODIFY LABEL NOWAIT.

NOWAIT is available only in a program. Including NOWAIT has no effect on MODIFY LABEL when it is issued in the Command window.

SAVE

In FoxPro for MS-DOS, include SAVE to keep an image of the Label Design window on the desktop or in a user-defined window after the Label Design window is closed.

In FoxPro for Windows, include SAVE to keep an image of the Report Layout window in the main FoxPro window or in a user-defined window after the Report Layout window is closed.

The Label Design or Report Layout window is normally removed from the desktop, the main FoxPro window or a user-defined window after you close the editor window. Use CLEAR to erase the Label Design or Report Layout window from the desktop, the main FoxPro window or a user-defined window.

SAVE is available only from within a program. Including SAVE has no effect when it is issued in the Command window.

MODIFY MEMO

Purpose	Opens an editing window for a memo field
Syntax	MODIFY MEMO <memo field1> [, <memo field2> ...] [NOEDIT] [NOWAIT] [RANGE <expN1>, <expN2>] [[WINDOW <window name1>] [IN [WINDOW] <window name2> SCREEN]] [SAME] [SAVE]
See Also	CLOSE MEMO

Description MODIFY MEMO opens an editing window for a memo field in the current table/.DBF. In this window, you can view or change the contents of the memo field. You can open an editing window for each memo field if a record has more than one.

To close the memo editing window and save your changes in FoxPro for MS-DOS, press Ctrl+W, click on the close box or choose **Close** from the **File** menu. Press Ctrl+Q or Escape to close the memo editing window without saving your changes.

To close the memo editing window and save your changes in FoxPro for Windows, press Ctrl+W or choose **Close** from the **Control** or **File** menu. Press Ctrl+Q or Escape to close the memo editing window without saving your changes.

You can also close the editing memo window and save your changes by issuing CLOSE MEMO. Similarly, closing a table/.DBF saves changes made to a memo field or fields.

In a table/.DBF opened for shared access on a network, the current record is automatically locked when editing begins on a memo field.

Clauses <memo field1> [, <memo field2> ...]

Specify the names of one or more memo fields to edit with <memo field1>, <memo field2>, and so on. If you include the table/.DBF alias, you can open editing windows for memo fields in the noncurrent work area.

NOEDIT

NOEDIT prevents modification to a memo field opened with MODIFY MEMO. The memo field can't be changed but can be viewed and copied to the clipboard.

NOWAIT

When you issue MODIFY MEMO from within a program, a memo editing window is opened and program execution pauses until the editing window is closed. Include NOWAIT to continue program execution after the memo editing window is opened. The program continues executing on the program line immediately following MODIFY MEMO NOWAIT.

RANGE <expN1>, <expN2>

RANGE lets you open a memo editing window with a range of characters selected. Characters are selected starting at the position specified with <expN1> up to (but not including) the <expN2>th character. If <expN1> is equal to <expN2>, no characters are selected, and the cursor is placed at the position specified with <expN1>.

WINDOW <window name1>

If you include WINDOW <window name1>, the memo editing window takes on the characteristics of the window you specify with <window name1>. For example, if the window is defined with the FLOAT clause of DEFINE WINDOW, the editing window can be moved. The window you specify doesn't have to be active or visible, but it must be defined.

IN [WINDOW] <window name2>

If you include IN WINDOW <window name2>, the memo editing window is opened within a parent window without assuming the characteristics of the parent window. An editing window activated inside a parent window can't be moved outside the parent window. If the parent window is moved, the editing window moves with it.

To access the memo editing window, the parent window must first be defined with DEFINE WINDOW and must be visible.

IN SCREEN

By including IN SCREEN, you can explicitly place a memo editing window on the desktop in FoxPro for MS-DOS or in the main window in FoxPro for Windows when a user-defined window is active. A memo editing window is placed on the desktop or main FoxPro window by default.

SAME

Include SAME to prevent a memo editing window from coming forward as the active window. If the memo editing window is hidden, it is displayed but doesn't become the active window.

SAVE

Include the SAVE option to keep an image of a memo editing window on the desktop, the main FoxPro window or a user-defined window after the editing window is closed. After you close a memo editing window, it is normally removed from the desktop, the main FoxPro window, or a user-defined window. Use CLEAR to erase a memo editing window from the desktop, the main FoxPro window or a user-defined window.

SAVE is available only from within a program. Including SAVE has no effect when it is issued in the Command window.

Example

The following example opens the NOTES memo field for the first record in SALESMAN.DBF in an editing window with a highlighted range.

```
CLOSE DATABASES
USE salesman
WAIT WINDOW "Note that Janet's name is highlighted" NOWAIT
MODIFY MEMO notes NOEDIT RANGE 1,6
USE
```

MODIFY MENU

Purpose	Displays the Menu Design window and enables you to modify or create a menu system
Syntax	MODIFY MENU [<file> ?] [[WINDOW <window name1>] [IN [WINDOW] <window name2> IN SCREEN]] [NOWAIT] [SAVE]
See Also	DEFINE BAR, DEFINE MENU, DEFINE PAD, DEFINE POPUP, SET SYSMENU
Description	MODIFY MENU displays the Menu Design window in which you can modify existing menu files or create new ones. For information about the Menu Builder, see the Menu Builder chapter in the <i>FoxPro User's Guide</i>.
Clauses	<file> You can modify existing menus by specifying the file name of the menu definition table with <file>. If the file you specify doesn't exist or can't be found, a new menu file is created. ? If you include the ? clause or omit a menu file name, the Open dialog appears and displays a list of existing menu files to choose from. You can also choose New to create a new menu file. WINDOW <window name1> If you include WINDOW <window name1>, the Menu Design window takes on the characteristics of the window you specify with <window name1>. For example, if the window is defined with the FLOAT clause of DEFINE WINDOW, the Menu Design window can be moved. The window you specify doesn't have to be active or visible, but it must be defined.

IN [WINDOW] <window name2>

If you include IN WINDOW <window name2>, the Menu Design window is opened within a parent window without assuming the characteristics of the parent window. A Menu Design window activated inside a parent window can't be moved outside the parent window. If the parent window is moved, the Menu Design window moves with it.

To access the Menu Design window, the parent window must first be defined with DEFINE WINDOW and must be visible.

IN SCREEN

By including IN SCREEN, you can explicitly place a Menu Design window on the desktop in FoxPro for MS-DOS or in the main window in FoxPro for Windows when a user-defined window is active. The Menu Design window is placed on the desktop or main FoxPro window by default.

NOWAIT

When you issue MODIFY MENU from within a program, the Menu Design window is displayed and program execution pauses until the Menu Design window is closed. Include NOWAIT to continue program execution after the Menu Design window is opened. The program continues executing on the program line immediately following MODIFY MENU NOWAIT.

NOWAIT is available only in a program. Including NOWAIT has no effect on MODIFY MENU when it is issued in the Command window.

SAVE

Include SAVE to keep an image of the Menu Design window on the desktop, the main FoxPro window or a user-defined window after the Menu Design window is closed. After you close the Menu Design window, it is normally removed from the desktop, the main FoxPro window, or a user-defined window. Use CLEAR to erase the Menu Design window from the desktop, the main FoxPro window or a user-defined window.

SAVE is available only from within a program. Including SAVE has no effect when it is issued in the Command window.

MODIFY PROJECT

Purpose	Opens the Project window and enables you to create or modify a project file
Syntax	MODIFY PROJECT [<file> ?] [[WINDOW <window name1>] [IN [WINDOW] <window name2> SCREEN]] [SAVE]
See Also	BUILD APP, BUILD EXE, BUILD PROJECT, COMPILE, CREATE PROJECT

Description MODIFY PROJECT opens the Project window so you can interactively modify or create a project. A *project* is a special table that keeps track of all programs, screens, menus, libraries, reports, labels, .DBFs, indexes, format files, and so on required to create an application, as well as all the dependencies, references and connections among the files. A project file has a .PJX file name extension and an associated memo file with a .PJT extension. In a project, you specify all the pieces that are required for the final application, and then FoxPro makes sure the generated application is based on the latest source files.

For information on the Project Manager, refer to the Project Manager chapter in the *FoxPro User's Guide*.

Clauses <file>

You can modify existing projects by specifying the file name of the project definition table/.DBF with <file>. If the file you specify doesn't exist or can't be found, a new project can be created.

Issuing MODIFY PROJECT without any additional arguments causes the Open dialog to appear. The name Untitled is assigned to the project file. When you close the Project window, you can save the project with a different name.

?

If you include the ? clause or omit the file name, the Open dialog appears, displaying a list of existing project definition files to choose from. You can also choose the **New** push button to create a new project.

WINDOW <window name1>

If you include WINDOW <window name1>, the Project window takes on the characteristics of the window you specify with <window name1>. For example, if the window is defined with the FLOAT clause of DEFINE WINDOW, the Project window can be moved. The window you specify doesn't have to be active or visible, but it must be defined.

IN [WINDOW] <window name2>

If you include IN WINDOW <window name2>, the Project window is opened within a parent window without assuming the characteristics of the parent window. A Project window activated inside a parent window can't be moved outside the parent window. If the parent window is moved, the Project window moves with it.

To access the Project window, the parent window must first be defined with DEFINE WINDOW and must be visible.

IN SCREEN

By including IN SCREEN, you can explicitly place the Project window on the desktop in FoxPro for MS-DOS or the main window in FoxPro for Windows when a user-defined window is active. The Project window is placed on the desktop or main FoxPro window by default.

NOWAIT

When you issue MODIFY MENU from within a program, the Project window is opened and program execution pauses until the Project window is closed. Include NOWAIT to continue program execution after the Project window is opened. The program continues executing on the program line immediately following MODIFY PROJECT NOWAIT.

NOWAIT is available only from within a program. Including NOWAIT has no effect on MODIFY PROJECT when it is issued in the Command window.

SAVE

Include SAVE to keep an image of the Project window on the desktop, the main FoxPro window or a user-defined window after the Project window is closed. After you close the Project window, it is normally removed from the desktop, the main FoxPro window or a user-defined window. Use CLEAR to erase the Project window from the desktop, the main FoxPro window or a user-defined window.

SAVE is available only from within a program. Including SAVE has no effect when it is issued in the Command window.

MODIFY QUERY

Purpose	Opens the RQBE window, in which you can create or modify a query
Syntax	MODIFY QUERY [<file> ?] [NOWAIT]
See Also	CREATE QUERY, DO
Description	MODIFY QUERY lets you interactively modify an existing query or create a new query in the RQBE window. For more information about creating queries, see the RQBE chapter in the <i>FoxPro User's Guide</i> or the Retrieving Your Data chapter in the <i>FoxPro Getting Started</i> manual.
Clauses	<file> You can modify an existing query by specifying the name of the query file. If the file you specify doesn't exist or can't be found, a new query can be created. Issuing MODIFY QUERY without any additional arguments causes the Open dialog to appear. The name Untitled is assigned to the query. When you exit the RQBE window, you can save the query with a different name. After you have created a query, the query is stored as a FoxPro program file with a .QPR extension. You can execute a query program with DO, but you must include the .QPR extension with the query file name. ? If you include the ? clause or omit the file name, the Open dialog appears, displaying a list of existing query files to choose from. You can also choose the New push button to create a new query. NOWAIT When you issue MODIFY QUERY from within a program, the RQBE window is opened and program execution is suspended until the RQBE window is closed. Including NOWAIT continues program execution after the RQBE window has been opened. The program doesn't wait for the RQBE window to be closed but continues executing from the program line immediately following MODIFY QUERY NOWAIT. NOWAIT is available only from within a program. Including NOWAIT has no effect on the MODIFY QUERY command when it is issued from the Command window.

MODIFY REPORT

Purpose	Opens the Report Design window
Syntax	MODIFY REPORT [<file> ?] [[WINDOW <window name1>] [IN [WINDOW] <window name2> SCREEN]] [NOENVIRONMENT] [NOWAIT] [SAVE]
See Also	CREATE REPORT, REPORT

Description MODIFY REPORT lets you interactively modify an existing report or create a new report in the Report Design window.

For more information on creating and modifying reports, see the reports chapter in the *FoxPro User's Guide* or the Reporting on Your Data chapter in the *FoxPro Getting Started* manual.

Clauses <file>

You can modify an existing report by specifying the name of the report definition file with <file>. If the file you specify doesn't exist or can't be found, a new report is created.

To modify a report created in FoxBASE+, you must include the .FRM file name extension in the name you specify with <file>. When you modify a report created in FoxBASE+, the report is saved in a FoxPro format, and the report can't be opened again in FoxBASE+. A FoxPro table and memo file are created with the same name as the FoxBASE+ report. The table name has an .FRM extension; the memo file name has an .FPT extension. If you change the table name's .FRM extension to .FRX and the memo file name's .FPT extension to FRT, you can then open the report without specifying an extension.

?

If you include the ? clause or omit the report file name, the Open dialog appears and displays a list of existing report files to choose from. You can also choose **New** to create a new report file. If you choose **New**, the name Untitled is assigned to the Report Design window. You can save the report file with a different name when you close the Report Design window.

WINDOW <window name1>

If you include WINDOW <window name1>, the Report Design window takes on the characteristics of the window you specify with <window name1>. For example, if the window is defined with the FLOAT clause of DEFINE WINDOW, the Report Design window can be moved. The window you specify doesn't have to be active or visible, but it must be defined.

IN [WINDOW] <window name2>

If you include IN WINDOW <window name2>, the Report Design window is opened within a parent window without assuming the characteristics of the parent window. A Report Design window activated inside a parent window can't be moved outside the parent window. If the parent window is moved, the Report Design window moves with it.

To access the Report Design window, the parent window must first be defined with DEFINE WINDOW and must be visible.

IN SCREEN

By including IN SCREEN, you can explicitly place a Report Design window on the desktop in FoxPro for MS-DOS or the main window in FoxPro for Windows when a user-defined window is active. The Report Design window is placed on the desktop or main FoxPro window by default.

NOENVIRONMENT

When you close the Report Design window, you are prompted to save the current FoxPro environment with the report definition file. Saving the FoxPro environment places an additional record in the report definition table. This record contains the names of all open table and index files, the index order, and any relationships between the tables.

MODIFY REPORT automatically restores the environment saved with the report definition file if you saved the FoxPro environment. If you include NOENVIRONMENT, the environment isn't restored. Include NOENVIRONMENT to prevent changes to the current environment.

NOWAIT

When you issue MODIFY REPORT from within a program, the Report Design window is opened and program execution pauses until the Report Design window is closed. Include NOWAIT to continue program execution after the Report Design window is opened. The program continues executing on the program line immediately following MODIFY REPORT NOWAIT.

NOWAIT is available only from within a program. Including NOWAIT has no effect on MODIFY REPORT when it is issued in the Command window.

SAVE

Include SAVE to keep an image of the Report Design window on the desktop, the main FoxPro window or a user-defined window after the Report Design window closed. When you close the Report Design window, it is normally removed from the desktop, the main FoxPro window or a user-defined window. Use CLEAR to erase the Report Design window from the desktop, the main FoxPro window or a user-defined window.

SAVE is available only from within a program. Including SAVE has no effect when it is issued in the Command window.

MODIFY SCREEN

Purpose	Opens the Screen Design window and enables you to create or modify a screen
Syntax	<pre>MODIFY SCREEN [<file> ?] [[WINDOW <window name1>] [IN [WINDOW] <window name2> IN SCREEN]] [NOENVIRONMENT] [NOWAIT] [SAVE]</pre>
See Also	CREATE SCREEN
Description	MODIFY SCREEN lets you interactively modify existing screens or create a new screen in the Screen Design window. For more information on creating and modifying screens, see the Screen Builder chapter in the <i>FoxPro User's Guide</i> or the Designing a Custom Input Screen chapter in the <i>FoxPro Getting Started</i> manual.
Clauses	<file> You can modify an existing screen by specifying the name of the screen definition file. If the file you specify doesn't exist or can't be found, a new screen is created. ? If you include the ? clause or omit the screen file name, the Open dialog appears and displays a list of existing screen files to choose from. You can also choose New to create a new screen file. If you choose New, the name Untitled is assigned to the Screen Design window. You can save the screen file with a different name when you close the Screen Design window. WINDOW <window name1> If you include WINDOW <window name1>, the Screen Design window takes on the characteristics of the window you specify. For example, if the window is defined with the FLOAT clause of DEFINE WINDOW, the Screen Design window can be moved. The window you specify doesn't have to be active or visible, but it must be defined.

IN [WINDOW] <window name2>

If you include IN WINDOW <window name2>, the Screen Design window is opened within a parent window without assuming the characteristics of the parent window. A Screen Design window activated inside a parent window can't be moved outside the parent window. If the parent window is moved, the Screen Design window moves with it.

To access the Screen Design window, the parent window must first be defined with DEFINE WINDOW and must be visible.

IN SCREEN

By including IN SCREEN, you can explicitly place a Screen Design window on the desktop in FoxPro for MS-DOS or the main window in FoxPro for Windows when a user-defined window is active. The Screen Design window is placed on the desktop or the main FoxPro window by default.

NOENVIRONMENT

When you close the Screen Design window, you are prompted to save the current FoxPro environment with the screen definition file. Saving the FoxPro environment places an additional record in the screen definition table. This record contains the names of all open table and index files, the index order, and any relationships between the tables.

MODIFY SCREEN automatically restores the environment saved with the screen definition file if you saved the FoxPro environment. If you include NOENVIRONMENT, the environment isn't restored. Include NOENVIRONMENT to prevent changes to the current environment.

NOWAIT

When you issue MODIFY SCREEN from within a program, the Screen Design window is opened and program execution pauses until the Screen Design window is closed. Include NOWAIT to continue program execution after the Screen Design window is opened. The program continues executing on the program line immediately following MODIFY SCREEN NOWAIT.

NOWAIT is available only from within a program. Including NOWAIT has no effect on MODIFY SCREEN when it is issued in the Command window.

SAVE

Include **SAVE** to keep an image of the Screen Design window on the desktop, the main FoxPro window or a user-defined window after the Screen Design window closed. After you close the Screen Design window, it is normally removed from the desktop, the main FoxPro window, or a user-defined window. Use **CLEAR** to erase the Screen Design window from the desktop, the main FoxPro window or a user-defined window.

SAVE is available only from within a program. Including **SAVE** has no effect when it is issued in the Command window.

MODIFY STRUCTURE

Purpose	Allows you to modify the current table/.DBF structure
Syntax	MODIFY STRUCTURE
See Also	CREATE

Description Use MODIFY STRUCTURE to change the structure of the table open in the current work area. If no file is open in the current work area, you are prompted to select a table to modify.

Changes you can make to a table's structure include the addition and deletion of fields and the modification of field names, sizes and types. You can also add indexes and/or tags to a table.

You can also modify the structure of tables by choosing **Setup...** from the **Database** menu or by choosing the **Setup** push button in the View window. For more information on modifying the structure of tables through the FoxPro interface, see the Database Menu chapter in the *FoxPro User's Guide*.

FoxPro automatically makes a backup copy of the current table/.DBF before you change the table's structure. When the modifications are complete, the data contained in the backup copy of the table is appended to the newly modified record structure. A memo field backup file is also created. The table backup file has a .BAK extension, and the memo file backup file has a .TBK extension.

When you modify the structure of a table that has a memo field, the blocksize of the memo file is set to the current blocksize setting. You can specify the memo file blocksize with SET BLOCKSIZE.

Keep in mind when changing fields from one data type to another that the field contents will not always transfer or transfer cleanly. For example, if you wish to convert a field of date type to numeric type, the field contents will not transfer.

Also, if you accept the structure changes and interrupt the data copying process, the new file will not contain all the records in the original table.

In either case, remember that FoxPro creates a .BAK copy of the original table file and a .TBK copy of the original memo field file, if the table has a memo field. If you have any problems with MODIFY STRUCTURE, you can delete the new file(s) and rename the .BAK file and .TBK file, if any, to the original file extensions (.DBF and .FPT).

MODIFY WINDOW

Purpose	Modifies a defined window
Syntax	MODIFY WINDOW <window name> SCREEN [FROM <row1, column1> TO <row2, column2> AT <row3, column3> SIZE <row4, column4>] [FONT <expC1> [, <expN1>]] [STYLE <expC2>] [TITLE <expC3>] [HALFHEIGHT] [DOUBLE PANEL NONE SYSTEM] [CLOSE NOCLOSE] [FLOAT NOFLOAT] [GROW NOGROW] [MINIMIZE] [ZOOM NOZOOM] [ICON FILE <expC4>] [FILL FILE <bmp file>] [COLOR SCHEME <expN2> COLOR <color pair list>]
Remarks	FoxPro for Windows only
See Also	DEFINE WINDOW

Description MODIFY WINDOW changes the attributes of an existing user-defined window or the main FoxPro window. User-defined windows are created with DEFINE WINDOW. MODIFY WINDOW can't be used to change the attributes of FoxPro system windows (the Command window, Browse windows, the Filer, and so on).

Use MODIFY WINDOW to change the location, default font, title, border, controls, icon, wallpaper and color of a user-defined window or the main FoxPro window. Any of these attributes can be changed by including the clauses listed above.

For example, include the TO and FROM or AT and SIZE clauses to specify a new location or size for a user-defined window or the main FoxPro window. To prevent a user-defined window or the main FoxPro window from being moved, include the NOFLOAT key word.

Clauses <window name>

Specify the user-defined window to modify with <window name>. The window you specify must first be created with DEFINE WINDOW.

SCREEN

You can modify the attributes of the main FoxPro window by including SCREEN instead of a window name. To return the main FoxPro window to its startup configuration, issue the following command without any additional clauses:

MODIFY WINDOW SCREEN



Issuing MODIFY WINDOW SCREEN NOCLOSE is an excellent way to prevent yourself from accidentally terminating FoxPro for Windows prematurely.

Additional Clauses

For additional information about the MODIFY WINDOW clauses, see DEFINE WINDOW in this chapter.

MONTH()

Purpose	Returns the numeric month for a given date
Syntax	MONTH(<expD>)
Returns	Numeric
See Also	CMONTH(), DAY(), SYS(), YEAR()

Description MONTH() returns a number from 1 through 12 indicating the month of the year. January is month 1, and December is month 12.

Parameter <expD>
<expD> is the date expression for which you want MONTH() to return the month number.

Examples

```
? DATE( )
08/12/92

? MONTH( DATE( ) )
8

STORE {05/23/92} TO buy
STORE MONTH(buy + 30) TO month
? month
6
```

MOVE POPUP

Purpose	Moves a popup defined with DEFINE POPUP or a FoxPro system menu popup to a new location
Syntax	MOVE POPUP <popup name> TO <row, column> BY <expN1>, <expN2>
See Also	ACTIVATE POPUP, DEFINE POPUP, SIZE POPUP

Description MOVE POPUP moves a popup to a different location on the desktop in FoxPro for MS-DOS, the main FoxPro window in FoxPro for Windows or a user-defined window. The popup can be moved to a specific position or relative to its current position. If a popup has been defined, it can be moved; it doesn't have to be active or visible.

Clauses **<popup name>**

Specify the popup to move with <popup name>, which can be the name of a system menu popup (such as _MFILE, MEDIT, OR _MDATA) or a user-defined popup.

TO <row, column>

Include this clause to move the popup to a specific location on the desktop, main FoxPro window or a user-defined window.

BY <expN1>, <expN2>

Include this clause to move the popup to a location relative to its current position. <expN1> specifies the number of rows to move the popup (down if <expN1> is positive, up if negative). <expN2> specifies the number of columns to move the popup (to the right if <expN2> is positive, to the left if negative).

Example The following example defines and activates a popup, then moves and enlarges it to demonstrate popup versatility.

```
CLOSE DATABASE
CLEAR
DEFINE POPUP movinpop FROM 2,2 TO 7, 14 PROMPT FILES LIKE *.PRG ;
    TITLE 'Programs'
ACTIVATE POPUP movinpop NOWAIT
=CHRSAW(2)
MOVE POPUP movinpop BY 5,5 && Move popup down
=CHRSAW(2)
SIZE POPUP movinpop BY 5,5 && Enlarge the popup
=CHRSAW(2)
SIZE POPUP movinpop BY -5,-5 && Shrink the popup
=CHRSAW(2)
MOVE POPUP movinpop BY -5,-5 && Move popup up
=CHRSAW(2)
DEACTIVATE POPUP movinpop
RELEASE POPUP  movinpop
```

MOVE WINDOW

Purpose	Moves a window to a specified screen location
Syntax	MOVE WINDOW <window name> TO <row, column> BY <expN1>, <expN2> [CENTER]
See Also	ACTIVATE WINDOW

Description Use **MOVE WINDOW** to move a user-defined or system window to a specified location on the desktop in FoxPro for MS-DOS, the main FoxPro window in FoxPro for Windows or a user-defined window. The window can be moved to a specific position or relative to its current position. If a window is defined, it can be moved; it doesn't have to be active or visible.

Clauses <window name>

Specify the name of the window to move with <window name>.

TO <row, column>

Include this clause to specify the row and column coordinates to move a window to on the desktop, the main FoxPro window or a user-defined window.

BY <expN1>, <expN2>

Include this clause to move a window to a location relative to its current position. The first numeric expression <expN1> specifies the number of rows to move the window (down if <expN1> is positive, up if negative). The second numeric expression <expN2> specifies the number of columns to move the window (to the right if <expN2> is positive, to the left if negative).

CENTER

You can center a window on the desktop, the main FoxPro window, or in a parent window by including **CENTER**.

Example In this example, after the window menter is defined and activated, the window is moved.

```
DEFINE WINDOW menter FROM 10,4 TO 15,54 SYSTEM ;  
    TITLE "Nomadic Window"  
ACTIVATE WINDOW menter  
WAIT WINDOW 'Press any key to move window'  
MOVE WINDOW menter TO 20,15  
WAIT WINDOW 'Press any key to release window'  
RELEASE WINDOW menter
```

MRKBAR()

Purpose	Determines whether or not a bar on a popup defined with DEFINE POPUP or on a FoxPro menu system popup is marked with SET MARK OF
Syntax	MRKBAR(<expC>, <expN>)
Parameters	<expC> The name of the popup <expN> The number of the bar to check
Returns	Logical
See Also	MRKPAD(), SET MARK OF

Description MRKBAR() returns a logical value that indicates whether a popup bar (option) is marked. A popup bar can be marked or unmarked with SET MARK OF.

If the specified popup bar is marked, true (.T.) is returned. If the popup bar isn't marked, false (.F.) is returned.

Parameters **<expC>**

Specify the name of the popup containing the bar with <expC>.

<expN>

Specify the number of the bar with <expN>.

For an example using MRKBAR(), see CNTBAR() in this manual.

MRKPAD()

Purpose	Determines whether or not a pad on a user-defined menu bar or on the FoxPro system menu bar is marked with SET MARK OF
Syntax	MRKPAD(<expC1>, <expC2>)
Parameters	<expC1> The name of the menu bar <expC2> The name of the menu pad to check
Returns	Logical
See Also	MRKBAR(), SET MARK OF

Description MRKPAD() returns a logical value that indicates whether a menu pad is marked. A menu pad can be marked or unmarked with SET MARK OF.

If the specified menu pad is marked, true (.T.) is returned. If the menu pad isn't marked, false (.F.) is returned.

Parameters <expC1>

Specify the name of the menu bar containing the menu pad with <expC1>.

<expC2>

Specify the menu pad name with <expC2>.

Example The following example uses MRKPAD() to toggle the mark character of a menu pad when you choose it.

The current system menu bar is first saved to memory with SET SYSMENU SAVE, and all the system menu pads are removed with SET SYSMENU TO.

Several system menu pads are created with DEFINE PAD. When you choose a menu pad, the CHOICE procedure is executed. CHOICE displays the name of the menu pad you choose and the name of the menu bar. SET MARK OF is used with MRKPAD() to display or hide the menu pad's mark character. If you choose **Exit**, the original FoxPro system menu is restored.

```

*** Name this program MARKPAD.PRG ***

CLEAR

SET SYSMENU SAVE
SET SYSMENU TO

SET MARK OF MENU _MSYSMENU TO CHR(4)
PUBLIC markpad
markpad = .T.

DEFINE PAD syspad    OF _MSYSMENU PROMPT '\<System'  COLOR SCHEME 3 ;
    KEY ALT+S, ''
DEFINE PAD editpad    OF _MSYSMENU PROMPT '\<Edit'    COLOR SCHEME 3 ;
    KEY ALT+E, ''
DEFINE PAD recordpad  OF _MSYSMENU PROMPT '\<Record'   COLOR SCHEME 3 ;
    KEY ALT+R, ''
DEFINE PAD windowpad  OF _MSYSMENU PROMPT '\<Window'   COLOR SCHEME 3 ;
    KEY ALT+W, ''
DEFINE PAD reportpad  OF _MSYSMENU PROMPT 'Re\<ports'  COLOR SCHEME 3 ;
    KEY ALT+P, ''
DEFINE PAD exitpad    OF _MSYSMENU PROMPT 'E\<xit'    COLOR SCHEME 3 ;
    KEY ALT+X, ''

ON SELECTION MENU _MSYSMENU ;
    DO choice IN markpad WITH PAD( ), MENU( )

PROCEDURE choice
PARAMETER mpad, rmenu

WAIT WINDOW 'You chose ' + mpad + ;
    ' from menu ' + rmenu NOWAIT
SET MARK OF PAD (mpad) OF _MSYSMENU TO ;
    ! MRKPAD('_MSYSMENU', mpad)
markpad = ! markpad

IF mpad = 'EXITPAD'
    SET SYSMENU TO DEFAULT
ENDIF

```

MROW()

Purpose	Returns the screen or window row position of the mouse pointer
Syntax	MROW([<expC>])
Returns	Numeric
See Also	COL(), MCOL(), ROW(), WCOLS(), WROWS()

Description MROW() returns to row position of the mouse pointer on the desktop in FoxPro for MS-DOS, the main FoxPro window in FoxPro for Windows or user-defined window.

Parameter <expC>

If <expC> is the name of a window, MROW() returns the mouse pointer row coordinate relative to the specified window.

MROW() returns the desktop or main FoxPro window row position of the mouse pointer if <expC> is omitted and there is no active user-defined window.

MROW() returns the mouse pointer row coordinate relative to the active user-defined window if there is an active user-defined window and <expC> is omitted. MROW() returns a value of -1 if the mouse pointer is positioned outside the user-defined window.

In FoxPro for MS-DOS, MROW() returns 0 if <expC> is the name of a Browse window or the Debug window and the mouse pointer is positioned on the top window border. MROW() returns -1 if the mouse pointer is positioned on the top border of any other windows.



MROW() returns -1 if no mouse driver is loaded and there is no output window.

Example

See the second example for INKEY() in this chapter.

MWINDOW()

Purpose	Indicates over which window the mouse pointer is positioned
Syntax	MWINDOW([<window name>])
Returns	Character or Logical
See Also	MCOL(), MDOWN(), MROW(), WONTOP(), WOUTPUT()

Description If you don't specify a window name, MWINDOW() returns the name of the window over which the mouse pointer is positioned. The null string is returned if the mouse pointer is positioned over the desktop in FoxPro for MS-DOS or the main FoxPro window in FoxPro for Windows.

Parameter <window name>

If you specify a window name in MWINDOW() with <window name>, true (.T.) is returned if the mouse pointer is positioned over the specified window. False (.F.) is returned if the mouse pointer isn't positioned over the window.

NDX()

Purpose	Returns the name of an open .IDX index file for the current or specified table/.DBF
Syntax	NDX(<expN1> [, <expN2> <expC>])
Parameters	<expN1> The position in the file list of the .IDX file to be returned <expN2> The work number for a table/.DBF <expC> The table/.DBF alias
Returns	Character
See Also	CDX(), INDEX, MDX(), SET FULLPATH, SET INDEX, SYS(14), TAG(), USE

Description NDX() returns the name of an open single-entry .IDX index file. The CDX() and MDX() functions can be used to return the names of open .CDX compound index files.

Parameters <expN1>

USE and SET INDEX both support an index file list that lets you open index files for a table/.DBF. The index file name returned by this function is determined by the order of the file names in this index file list. For example, if <expN1> is 1, the name of the first index file in the index file list is returned; if <expN1> is 2, the second index name is returned; and so on. Names of any compound index files in the index file list are ignored.

The null string is returned if <expN1> is greater than the number of .IDX index files in the index file list.

<expN2> | <expC>

If you don't specify a work area or table/.DBF alias, NDX() returns the names of .IDX index files open with the table/.DBF in the current work area.

You can return names of .IDX index files open in other work areas by specifying a work area number with <expN2> or a table/.DBF alias with <expC>.

NDX() returns the null string if no table/.DBF is open in the work area you specify. If no table/.DBF has the alias you include, the message "Alias not found" is displayed.

NOTE | * | &&

Purpose	Indicate the beginning of a non-executing comment line in a program file
Syntax	NOTE [<comments>] * [<comments>] && [<comments>]
See Also	MODIFY COMMAND

Description Use the NOTE, * and && commands to place comments in program files. To continue a comment on the following line, place a semi-colon at the end of the comment.

Commands **NOTE [<comments>]**

The word NOTE at the beginning of a program line indicates that the line is a comment. For example:

```
NOTE      This is a comment
```

*** [<comments>]**

An asterisk (*) at the beginning of a program line indicates that the line is a comment. For example:

```
*      This is a comment
```

&& [<comments>]

Two ampersands (&&) at the end of a program line indicate that in-line comments follow. For example:

```
STORE (20*12) TO payments    && 20 years of monthly payments
```

Inserting in-line comments to denote the end of IF ... ENDIF, DO and FOR ... ENDFOR constructs greatly improves the readability of programs.

Example

```
NOTE  Initialize the page number;
      variable.
STORE 1 to MFGNMBR
* Set up the loop
DO WHILE MFGNMBR <= 25    && loop 25 times
    MFGNMBR = MFGNMBR + 1
ENDDO                    && DO WHILE MFGNMBR <= 25
```

NUMLOCK()

Purpose	Returns the current mode of the NumLock key or sets the mode of the NumLock key on or off
Syntax	NUMLOCK([<expL>])
Returns	Logical
See Also	CAPSLOCK(), INSMODE()

Description NUMLOCK() returns a true (.T.) or false (.F.) value, depending on the current mode of the NumLock key. True (.T.) is returned if NumLock is on (pressing a key on the numeric keypad returns a number). False (.F.) is returned if NumLock is off (pressing a key on the numeric keypad moves the cursor).

Parameter <expL>

To turn the NumLock key on or off, include <expL>. If <expL> is .T., the NumLock key is turned on; if <expL> is .F., the NumLock key is turned off. NUMLOCK() returns a logical value corresponding to the NumLock key setting *before* NUMLOCK(.T.) or NUMLOCK(.F.) is issued.

Examples In these examples, the equal sign (=) is used to execute NUMLOCK() without returning a value.

```
oldlock = NUMLOCK( )      && Save original setting
= NUMLOCK(.T.)            && Turn numlock on
= NUMLOCK(!NUMLOCK( ))    && Toggle NumLock to the opposite value
= NUMLOCK(oldlock)        && Return to original setting
```

OBJNUM()

Purpose	Returns the object number of an @ ... GET control
Syntax	OBJNUM(<var> [, <expN>])
Parameters	<var> The name of the field, memory available, or array element specified with @ ... GET <expN> The read-level number
Returns	Numeric
See Also	@ ... GET, CLEAR READ, _CUROBJ, RDLEVEL(), READ, SHOW GET, SHOW OBJECT

Description You can use @ ... GET and @ ... EDIT to create controls, sometimes called *objects*. These controls are fields, check boxes, lists, popups; invisible, push, and radio buttons; spinners and text editing regions. OBJNUM() returns a number that corresponds to a control's creation order in a set of controls.

Parameters **<var>**
<var> is the name of the memory variable, array element or field specified when you create the control.

<expN>
Nested READs are created by issuing @ ... GETs and a READ in a routine called during a READ. READs can be nested to five levels. To return a control number for a control at a read level other than the current read level, include the optional read level number <expN>. If <expN> is omitted, OBJNUM() returns the @ ... GET control number for the current read level.

Example The following example creates a window and places radio buttons in the window that let you move the record pointer through an open table. If a table isn't open, the Open dialog is displayed so you can select a table to open.


```

SET TALK OFF
DEFINE WINDOW gotodialog FROM 9, 17 TO 19,61 ;
    FLOAT NOCLOSE SHADOW DOUBLE COLOR SCHEME 5
PRIVATE file,lastobj,enter,tab,shifttab,up,down,left,right

```

```

***   Assign lastkey values   ***

```

```

enter    = 13
tab      = 9
shifttab = 15
up       = 5
down     = 24
right    = 4
left     = 19
lastobj  = 1

```

```

***   Open a table/.DBF   ***

```

```

IF EMPTY(DBF( ))
    file = GETFILE('DBF','Pick a table/.DBF')
    IF EMPTY(FILE)
        WAIT WINDOW 'Cancelled' NOWAIT
        RETURN
    ENDIF
    USE (file)
ENDIF

```

```

***   Draw the fields   ***

```

```

ACTIVATE WINDOW gotodialog
@ 0,1 TO 8,25
@ 1,3 GET radio PICTURE '@*RVN \<Top;\<Bottom;\<Record;\<Skip' ;
    SIZE 1,10,1 DEFAULT 1 WHEN radwhen( ) VALID radvalid( )
@ 5,15 GET recordnum SIZE 1,8 DEFAULT 0 ;
    FUNCTION 'Z' WHEN recwhen( ) VALID recvalid( ) DISABLE
@ 7,15 GET skipnum SIZE 1,8 DEFAULT 0 ;
    FUNCTION 'Z' WHEN skipwhen( ) DISABLE
@ 3,29 GET okcancel PICTURE '@*VT \!\<Goto;\?\<Cancel' ;
    SIZE 1,10,2 DEFAULT 0 WHEN okwhen( ) VALID okvalid( )
READ CYCLE
RELEASE WINDOW gotodialog

```

```

*** Place cursor in recordnum field if it is enabled ***
*** and the last object selected was the radio button ***
*** record. Do this only if Enter, Tab, Down Arrow or ***
*** Right Arrow is pressed to exit the record radio button. ***

```

```

FUNCTION radwhen

```

```

IF _CUROBJ = OBJNUM(radio)+3 ;
  AND radio = OBJNUM(radio)+2 ;
  AND lastobj = OBJNUM(radio)+2 ;
  AND (LASTKEY( ) = enter OR LASTKEY( ) = tab ;
  OR LASTKEY( ) = down OR LASTKEY( ) = right)
  _CUROBJ = OBJNUM(recordnum)
  lastobj = OBJNUM(recordnum)
  RETURN
ENDIF

```

```

*** Place cursor in recordnum field if it is enabled ***
*** and the last object selected was the radio button ***
*** skip. Do this only if Shift+Tab, Up Arrow or ***
*** Left Arrow is pressed to exit the skip radio button. ***

```

```

IF _CUROBJ = OBJNUM(radio)+2 AND radio = OBJNUM(radio)+2 ;
  AND lastobj = OBJNUM(radio)+3 AND (LASTKEY( ) = shifttab ;
  OR LASTKEY( ) = left OR LASTKEY( ) = up)
  lastobj = _CUROBJ
  _CUROBJ = OBJNUM(recordnum)
  RETURN
ENDIF
lastobj = _CUROBJ

```

```

*** Enable and disable appropriate get fields when ***
*** a radio button is selected. ***

```

```

FUNCTION radvalid
DO CASE
CASE radio = 1 or radio = 2
    STORE 0 TO recordnum,skipnum
    SHOW GET recordnum DISABLED
    SHOW GET skipnum DISABLED
CASE radio = 3
    STORE 0 TO skipnum
    SHOW GET recordnum ENABLED
    SHOW GET skipnum DISABLED
    _CUROBJ = OBJNUM(recordnum)
CASE radio = 4
    STORE 0 TO recordnum
    SHOW GET recordnum DISABLED
    SHOW GET skipnum ENABLED
    _CUROBJ = OBJNUM(skipnum)
ENDCASE

```

```

*** Routine to bypass the recno field and move to the goto button. ***
*** Put cursor on goto push button if the last object selected ***
*** was the radio button skip. Do this only if Tab, Down Arrow ***
*** or Right Arrow is pressed to exit the skip radio button. ***

```

```

FUNCTION recwhen
IF lastobj = OBJNUM(radio)+3 AND (LASTKEY( ) = tab ;
    OR LASTKEY( ) = down OR LASTKEY( ) = right)
    _CUROBJ = OBJNUM(okcancel)
    RETURN
ENDIF
*** Routine to bypass the recno field and move to the skip button. ***
*** Place cursor on skip radio button if the last object selected ***
*** was the goto push button. Do this only if Shift+Tab, Up Arrow ***
*** or Left Arrow is pressed to exit the goto push button. ***

IF lastobj = OBJNUM(okcancel) AND (LASTKEY( ) = shifttab ;
    OR LASTKEY( ) = up OR LASTKEY( ) = left)
    _CUROBJ = OBJNUM(radio)+3
    RETURN
ENDIF
lastobj = _CUROBJ

```

```

FUNCTION recvalid
IF !MDOWN( ) AND (LASTKEY( ) = enter OR LASTKEY( ) = tab ;
    OR LASTKEY( ) = down OR LASTKEY( ) = right)
    _CUROBJ = OBJNUM(radio)+3
    RETURN
ENDIF

IF !MDOWN( ) AND (LASTKEY( ) = shifttab ;
    OR LASTKEY( ) = up OR LASTKEY( ) = left)
    _CUROBJ = OBJNUM(radio)+2
ENDIF

FUNCTION skipwhen
lastobj = _CUROBJ

FUNCTION okwhen
lastobj = _CUROBJ

*** Routine to process the button choice ***

FUNCTION okvalid
IF okcancel = 1
    DO CASE
    CASE radio = 1
        GO TOP
    CASE radio = 2
        GO BOTTOM
    CASE radio = 3
        IF recordnum > RECCOUNT( )
            WAIT WINDOW 'Record out of range' NOWAIT
        ELSE
            GO recordnum
        ENDIF
    CASE radio = 4
        IF skipnum+RECNO( ) > RECCOUNT( ) OR skipnum + RECNO( ) < 0
            WAIT WINDOW 'Record out of range' NOWAIT
        ELSE
            SKIP skipnum
        ENDIF
    ENDCASE
ENDIF

```

OCCURS()

Purpose	Returns the number of times a character expression occurs within another character expression
Syntax	OCCURS(<expC1>, <expC2>)
Parameters	<expC1> The character expression for which to search <expC2> The character expression to search
Returns	Numeric
See Also	\$, INLIST()

Description OCCURS() returns the number of times the value of a character expression is found in the value of another character expression. OCCURS() returns zero if the value of an expression isn't found in the value of another expression.

Parameters <expC1>

Include a character expression <expC1> for which to search.

<expC2>

The second character expression <expC2> is searched for occurrences of the first character expression <expC1>.

Examples

```
STORE 'abracadabra' TO m_string
? OCCURS('a',m_string)
5

? OCCURS('b',m_string)
2

? OCCURS('c',m_string)
1

? OCCURS('e',m_string)
0
```

OEMTOANSI()

Purpose	Converts each character in a character expression to the corresponding character in the ANSI character set
Syntax	OEMTOANSI(<expC>)
Remarks	FoxPro for Windows only
See Also	ANSITOOEM(), CHRTRAN(), SYS(15)

Description Use OEMTOANSI() to move data from FoxPro for MS-DOS to FoxPro for Windows.

Parameter **<expC>**

OEMTOANSI() converts each character in <expC> to the corresponding character in the ANSI character set. The character expression <expC> should contain characters from the MS-DOS (OEM) character set.

If a character in <expC> doesn't have an ANSI equivalent, the character is converted to a similar ANSI character. For example, MS-DOS line drawing characters may not be found in the ANSI character set.

ON BAR

Purpose	Activates a popup or menu bar when a specific popup option is chosen
Syntax	ON BAR <expN> OF <popup name1> [ACTIVATE POPUP <popup name2> ACTIVATE MENU <menu name>]
See Also	ACTIVATE MENU, DEFINE BAR, DEFINE MENU, DEFINE POPUP, ON SELECTION BAR, ON SELECTION POPUP

Description ON BAR specifies a popup or menu bar that is activated when you choose an option from a popup. A popup that displays and activates another popup is called a *hierarchical* popup. Use ON SELECTION BAR or ON SELECTION POPUP to execute a command when an option is chosen from a popup.

The popups and the menu bar are defined with DEFINE POPUP and DEFINE MENU, or are part of the FoxPro menu system.

Clauses <expN> OF <popup name1>

Each option in a popup can have another popup or a menu bar assigned to it. Specify the option's bar number <expN> and the name of the popup <popup name1> containing the option.

A popup option that has a popup or menu bar assigned to it has an arrow placed to the right of the option text. The arrow indicates that choosing that option activates an additional popup or menu bar. An additional space for the hierarchical arrow is placed to the right of each option if you define the popup with DEFINE POPUP ... MARGIN. If the popup is created without the MARGIN clause, the hierarchical arrow may overwrite the last character of the option.

**ACTIVATE POPUP <popup name2>
| ACTIVATE MENU <menu name>**

Specify the name of the popup <popup name2> or menu bar <menu name> to activate when the popup option is chosen. Use ON BAR without the ACTIVATE POPUP or ACTIVATE MENU to release a popup or menu bar from a popup option.

Example

The following example demonstrates a system of hierarchical popups. A menu bar named DINNER is created with two menu bar pads. Each pad uses ON PAD to activate the popup named MAINCOURSE or DESSERT. The popups named MAINCOURSE and DESSERT each have additional popups named BURGER, PIZZA and PIE assigned to their options with three ON BAR commands. The OLIVE and PIE options have additional popups assigned with two more ON BAR commands.

When a selection is made, ON SELECTION POPUP ALL executes a procedure named YOUR CHOICE that activates a window and displays the choice. The choice is determined with POPUP() and PROMPT(), which return the name of the popup and the contents of the option's prompt.

```
DEFINE WINDOW order FROM 10,0 TO 13,39
```

```
DEFINE MENU dinner
```

```
DEFINE PAD one OF dinner PROMPT '\<Main Course' KEY ALT+M, ''
```

```
DEFINE PAD two OF dinner PROMPT '\<Dessert' KEY ALT+D, ''
```

```
ON PAD one OF dinner ACTIVATE POPUP maincourse
```

```
ON PAD two OF dinner ACTIVATE POPUP dessert
```

```
DEFINE POPUP maincourse MARGIN MESSAGE ;
```

```
  'We have burgers and pizza today'
```

```
DEFINE BAR 1 OF maincourse PROMPT '\<Hamburgers'
```

```
DEFINE BAR 2 OF maincourse PROMPT '\<Pizza'
```

```
ON BAR 1 OF maincourse ACTIVATE POPUP burger
```

```
ON BAR 2 OF maincourse ACTIVATE POPUP pizza
```

```
DEFINE POPUP burger MARGIN MESSAGE ;
```

```
  'What would you like on your burger?'
```

```
DEFINE BAR 1 OF burger PROMPT '\<Ketchup'
```

```
DEFINE BAR 2 OF burger PROMPT '\<Mustard'
```

```
DEFINE BAR 3 OF burger PROMPT '\<Onions'
```

```
DEFINE BAR 4 OF burger PROMPT '\<Pickles'
```

```
DEFINE POPUP pizza MARGIN MESSAGE ;
```

```
  'Here are the available toppings'
```

```
DEFINE BAR 1 OF pizza PROMPT '\<Anchovies'
```

```
DEFINE BAR 2 OF pizza PROMPT '\<Green Peppers'
```

```
DEFINE BAR 3 OF pizza PROMPT '\<Olives'
```

```
DEFINE BAR 4 OF pizza PROMPT '\<Pepperoni'
```

```
ON BAR 3 OF pizza ACTIVATE POPUP olives
```



```

DEFINE POPUP olives MARGIN
DEFINE BAR 1 OF olives PROMPT '\<Black' MESSAGE 'Black olives?'
DEFINE BAR 2 OF olives PROMPT '\<Green' MESSAGE 'Green olives?'

DEFINE POPUP dessert MARGIN MESSAGE 'Our dessert offerings'
DEFINE BAR 1 OF dessert PROMPT '\<Brownies'
DEFINE BAR 2 OF dessert PROMPT '\<Cookies'
DEFINE BAR 3 OF dessert PROMPT '\<Ice Cream'
DEFINE BAR 4 OF dessert PROMPT '\<Pie'
ON BAR 4 OF dessert ACTIVATE POPUP pie

DEFINE POPUP pie MARGIN MESSAGE 'What kind of pie?'
DEFINE BAR 1 OF pie PROMPT '\<Blueberry'
DEFINE BAR 2 OF pie PROMPT '\<Cherry'
DEFINE BAR 3 OF pie PROMPT '\<Peach'
DEFINE BAR 4 OF pie PROMPT '\<Rhubarb'

ON SELECTION POPUP ALL DO yourchoice

ACTIVATE MENU dinner

PROCEDURE yourchoice
ACTIVATE WINDOW order
CLEAR
DO CASE
    CASE POPUP( ) = 'BURGER'
        @ 0,0 SAY 'A ' + LOWER(POPUP( )) + ' order:'
        @ 1,0 SAY 'You ordered a burger with ' + LOWER(PROMPT( ))
    CASE POPUP( ) = 'PIZZA'
        @ 0,0 SAY 'A ' + LOWER(POPUP( )) + ' order:'
        @ 1,0 SAY 'You ordered a pizza with ' + LOWER(PROMPT( ))
    CASE POPUP( ) = 'OLIVES'
        @ 0,0 SAY 'A pizza order:'
        @ 1,0 SAY 'You ordered a pizza with ' ;
            + LOWER(PROMPT( )) + ' olives'
    CASE POPUP( ) = 'DESSERT'
        @ 0,0 SAY 'A ' + LOWER(POPUP( )) + ' order:'
        @ 1,0 SAY 'You ordered ' + LOWER(PROMPT( )) + ' for dessert'
    CASE POPUP( ) = 'PIE'
        @ 0,0 SAY 'A pie order:'
        @ 1,0 SAY 'You ordered ' + LOWER(PROMPT( )) + ' pie'
ENDCASE
WAIT WINDOW
DEACTIVATE WINDOW order
RETURN

```

ON ERROR

Purpose	Specifies a command that executes when an error occurs
Syntax	ON ERROR [<command>]
See Also	DO, ERROR(), FUNCTION, LINENO(), MESSAGE(), PROGRAM(), PROCEDURE, RETRY

Description When an error occurs during program execution, FoxPro executes the command you specify with ON ERROR. Typically, ON ERROR uses DO to execute an error-handling procedure.

After the command specified with ON ERROR executes, program execution resumes on the line immediately following the line that caused the error. However, if the error-handling procedure includes RETRY, the program line that caused the error is executed again.

Clause <command>

If you specify with <command> a procedure to execute when an error occurs, you can use ERROR(), MESSAGE(), LINENO() and PROGRAM() to pass the error number, the error message, the program line number and program name to the procedure. This information can be used to correct the cause of the error.

Use ON ERROR without <command> to cause no command to execute when an error occurs (the default).

Example

```
ON ERROR DO errhand WITH ;
    ERROR( ), MESSAGE( ), MESSAGE(1), PROGRAM( ), LINENO( )

*** The next line should cause an error ***

USE nodatabase
ON ERROR && restore system error handler

PROCEDURE errhand
PARAMETER merror, mess, mess1, mprog, mlineno
? 'Error number: ' + LTRIM(STR(merror))
? 'Error message: ' + mess
? 'Line of code with error: ' + mess1
? 'Line number of error: ' + LTRIM(STR(mlinen))
? 'Program with error: ' + mprog
```

ON ESCAPE

Purpose	Specifies a command that executes when you press the Escape key during program or command execution
Syntax	ON ESCAPE [<command>]
See Also	INKEY()

Description When you press the Escape key during program or command execution, FoxPro executes the command you specify with ON ESCAPE. Typically, ON ESCAPE uses DO to execute a procedure.

After the command specified with ON ESCAPE executes, program execution resumes on the line immediately following the program line that was executing when Escape was pressed. However, if a procedure specified with ON ESCAPE includes RETRY, the program line that was executing when Escape was pressed executes again.

Use ON ESCAPE without a command to cause no command to execute when Escape is pressed (the default).

If both ON ESCAPE and ON KEY are in effect and Escape is pressed, the command specified with ON ESCAPE is executed.



An ON ESCAPE routine isn't executed if SET ESCAPE is OFF.

Example

This example sets up an infinite loop but defines an ON ESCAPE routine to exit it.

```
SET ESCAPE ON
ON ESCAPE DO stopit
WAIT WINDOW 'Press Esc to stop loop' NOWAIT
moreloop = .T.
DO WHILE moreloop
ENDDO
RETURN

PROCEDURE stopit
moreloop = .F.
RETURN
```

ON KEY

Purpose	Specifies a command that executes when you press any key during program execution
Syntax	ON KEY [<command>]
See Also	INKEY()

Description When you press any key during program execution, FoxPro executes the command you specify with ON KEY. Typically, ON KEY uses DO to execute a procedure.

After the command you specify with ON KEY executes, program execution resumes on the line immediately following the program line that was executing when a key was pressed. However, if a procedure specified with ON KEY includes RETRY, the program line that was executing when a key was pressed executes again.

Use ON KEY without a command to cause no command to execute when a key is pressed (the default).

If both ON KEY and ON ESCAPE are in effect and Escape is pressed, the command specified with ON ESCAPE is executed.

Example In this example, records from the CUSTOMER table/DBF are displayed. If a key is pressed while the records are being displayed, the procedure named PAUSE is called. PAUSE uses WAIT to suspend display.

```
SET TALK OFF
USE customer
CLEAR
ON KEY DO pause
DO WHILE NOT EOF( )
    ? 'Company: ' + company
    ? 'Address: ' + address
    ?
    SKIP
ENDDO

PROCEDURE pause
STORE INKEY( ) TO HOLD
WAIT
RETURN
```

ON KEY =

Purpose	Executes a command when a specific key or key combination is pressed during a READ
Syntax	ON KEY [= <expN>] [<command>]
Remarks	Included for backward compatibility – use ON KEY LABEL
See Also	INKEY(), ON KEY LABEL

Description When you press a specific key or key combination during the execution of a READ, FoxPro executes the command you specify with the ON KEY =. Pressing the specified key or key combination when a READ isn't in effect doesn't execute the ON KEY = command. ON KEY = typically uses DO to execute a procedure.

Only one ON KEY = command can be active at a time. Use ON KEY LABEL to execute commands for multiple keys or key combinations.

Clauses = <expN>

If the key you specify is a printable character, specify the ASCII value for the key with <expN>. If the key cannot be printed, or if you use a key combination, include a value from the following table in <expN>.

<command>

The command to be executed when the key is pressed is specified with <command>.

Keycodes	Corresponding Keystrokes
272 – 281	Alt+Q, W, E, R, T, Y, U, I, O, P
286 – 294	Alt+A, S, D, F, G, H, J, K, L
300 – 306	Alt+Z, X, C, V, B, N, M
315 – 324	F1 to F10 Function keys
327	Home
328	↑
329	PgUp
331	←
333	→
335	End
336	↓
337	PgDn
338	Ins
339	Del
340 – 349	Shift+F1 to Shift+F10
350 – 359	Ctrl+F1 to Ctrl+F10
360 – 369	Alt+F1 to Alt+F10
370	Ctrl+Print+Scrn
371	Ctrl+←
372	Ctrl+→
373	Ctrl+End
374	Ctrl+PgDn
375	Ctrl+Home
376 – 387	Alt+1, 2, 3, 4, 5, 6, 7, 8, 9, 0, -, =
388	Ctrl+PgUp

Example

In this example, records from CUSTOMER.DBF are displayed for editing. If you press F1, the name of the current field is passed to the procedure named SCRNHLP and a message is displayed in a window.

```

ON KEY = 315 DO scrnhlp WITH VARREAD( )
SET TALK OFF
USE customer
SCATTER TO temp
DEFINE WINDOW input FROM 6,10 TO 18,70 PANEL
ACTIVATE WINDOW input
@ 1,3   SAY 'Customer: '   GET company
@ 3,3   SAY 'Address: '   GET address
@ 5,3   SAY 'City: '      GET city
@ 7,3   SAY 'State: '     GET state
@ 7,18  SAY 'Zip: '       GET zip
@ 9,8   SAY 'Press Escape to cancel or F1 for help'
READ

IF LASTKEY( ) = 27
    GATHER FROM temp
ENDIF

DEACTIVATE WINDOW input
RELEASE WINDOW input

PROCEDURE scrnhlp
PARAMETERS fieldname
DEFINE WINDOW help_me FROM 1,0 TO 4,79
ACTIVATE WINDOW help_me
@ 0,20 SAY 'Sorry, no help available for ' + fieldname
WAIT
RELEASE WINDOW help_me
RETURN

```

ON KEY LABEL

Purpose	Specifies a command that executes when you press a specific key or key combination or click the mouse
Syntax	ON KEY [LABEL <key label>] [<command>]
See Also	INKEY(), ON(), POP KEY, PUSH KEY

Description When you press a specific key or key combination or click the mouse, FoxPro executes the command you specify with ON KEY LABEL for the key(s) or mouse click. ON KEY LABEL typically uses DO to execute a procedure.

ON KEY LABEL executes the command immediately during the execution of READ, BROWSE, EDIT, CHANGE and user-defined menus and popups. If a program is executing when the key is pressed or the mouse is clicked, the current program line is executed and then the ON KEY LABEL command is executed. Any ON KEY LABEL key assignments created in a program remain in effect after the program is run. ON KEY LABEL key assignments can also be created in the Command window.



To prevent recursive calls during the execution of an ON KEY LABEL procedure, include PUSH KEY CLEAR early in the procedure to disable all active ON KEY LABELs. Issue POP KEY at the end of the procedure to enable the ON KEY LABELs.

The ON KEY LABEL key assignments aren't in effect in the FoxPro system menu bar, dialogs, alerts, system popups and so on. ON KEY LABEL key assignments are effective in the FoxPro system windows — the FoxPro text editor, the Command window, the Trace window, and so on.

FoxPro for Windows cannot trap certain events because they are under Windows control. In particular, the ON KEY LABEL MOUSE, ON KEY LABEL LEFTMOUSE and ON KEY LABEL RIGHTMOUSE commands are executed when you click on a Windows control like a **Control** menu, scroll bars, and so on.

Unlike ON KEY, you can have multiple active ON KEY LABEL commands. For example, you can assign a command to each of the arrow keys and both mouse buttons.

Note that executing an ON KEY LABEL resets PARAMETERS() to 0. See PARAMETERS() topic in this chapter for additional information.

Clauses LABEL <key label>

The key label is specified as the letter or digit label on the keyboard key, or as a special name assigned to the key. See the following table for the special key label assignments.

Keycap Identification	<key label>
←	LEFTARROW
→	RIGHTARROW
↑	UPARROW
↓	DNARROW
Home	HOME
End	END
PgUp	PGUP
PgDn	PGDN
Del	DEL
Backspace	BACKSPACE
Spacebar	SPACEBAR
Ins	INS
Tab	TAB
Shift Tab	BACKTAB
Enter	ENTER
F1 to F12	F1, F2, F3...
Ctrl+F1 to Ctrl+F12	Ctrl+F1, Ctrl+F2 ...
Shift+F1 to Shift+F12	Shift+F1, Shift+F2 ...
Alt+F1 to Alt+F12	Alt+F1, Alt+F2, Alt+F3 ...
Alt+0 to Alt-9	Alt+0, Alt+1, Alt+2 ...
Alt+A to Alt+Z	Alt+A, Alt+B, Alt+C ...
Ctrl+←	Ctrl+LEFTARROW
Ctrl+→	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, Ctrl+C ...
Right Mouse	RIGHTMOUSE
Left Mouse	LEFTMOUSE
Mouse	MOUSE
Escape	ESC

<command>

<command> is the command that executes when the specified key is pressed or the mouse is clicked.

The following example displays a message when an arrow key is pressed.

Example

```

SET TALK OFF
CLEAR
ON KEY LABEL RIGHTARROW ? 'Right Arrow'
ON KEY LABEL LEFTARROW ? 'Left Arrow'
ON KEY LABEL UPARROW ? 'Up Arrow'
ON KEY LABEL DNARROW ? 'Down Arrow'
SET ESCAPE ON
STORE .F. TO mexit
ON ESCAPE STORE .T. TO mexit
STORE 0 TO TEMP
@ 2,0 SAY 'Press an arrow key to test ON KEY LABEL, <Escape> to exit'
DEFINE WINDOW menter FROM 10,20 TO 14,60 DOUBLE
ACTIVATE WINDOW menter
DO WHILE NOT mexit
  ENDDO
DEACTIVATE WINDOW menter
ON KEY

```

ON PAD

Purpose	Specifies the menu popup or menu bar that is activated when you choose a specific menu pad
Syntax	ON PAD <pad name> OF <menu name1> [ACTIVATE POPUP <popup name> ACTIVATE MENU <menu name2>]
See Also	ACTIVATE MENU, DEFINE MENU, ON SELECTION MENU, ON SELECTION PAD
Description	ON PAD specifies a menu popup or menu bar that FoxPro displays and activates when a specific menu pad is chosen. Use ON SELECTION PAD or ON SELECTION MENU to execute a command when a menu pad is chosen.
Clauses	<pad name> OF <menu name1> The menu popup or menu bar is assigned to the menu pad specified with <pad name> and <menu name1>. ACTIVATE POPUP <popup name> ACTIVATE MENU <menu name2> Specify the name of the menu popup <popup name> or menu bar <menu name2> to activate when the menu pad is chosen. Use ON PAD <pad name> OF <menu name1> without an ACTIVATE POPUP or ACTIVATE MENU clause to release a menu popup or menu bar from a menu pad.
Example	ON PAD is used in this program example to activate a menu popup when you choose a menu pad in the menu bar. The current FoxPro system menu bar is saved to memory with SET SYSMENU SAVE and all system menu pads are removed with SET SYSMENU TO. Two new system menu pads are created with DEFINE PAD, and DEFINE POPUP creates menu popups for each menu pad. DEFINE BAR creates the options in each of the menu popups. When a menu pad is chosen, ON PAD uses ACTIVATE POPUP to activate the corresponding menu popup.

When an option is chosen from a menu popup, ON SELECTION POPUP uses PROMPT() and POPUP() to pass the option number and menu popup name to the procedure named CHOICE. CHOICE displays the prompt of the option you choose and the name of the menu popup containing the option. The original FoxPro system menu is restored when you choose **Exit** from the menu popup named CARDINFO.

```
*** Name this program ONPAD.PRG ***

CLEAR

SET SYSMENU SAVE
SET SYSMENU TO

DEFINE PAD convpad OF _MSYSMENU PROMPT '\<Conversions' COLOR SCHEME 3 ;
    KEY ALT+C, ''
DEFINE PAD cardpad OF _MSYSMENU PROMPT 'Card \<Info' COLOR SCHEME 3 ;
    KEY ALT+I, ''

ON PAD convpad OF _MSYSMENU ACTIVATE POPUP conversion
ON PAD cardpad OF _MSYSMENU ACTIVATE POPUP cardinfo

DEFINE POPUP conversion MARGIN RELATIVE SHADOW COLOR SCHEME 4
DEFINE BAR 1 OF conversion PROMPT 'Ar\<ea' ;
    KEY CTRL+E, '^E'
DEFINE BAR 2 OF conversion PROMPT '\<Length' ;
    KEY CTRL+L, '^L'
DEFINE BAR 3 OF conversion PROMPT 'Ma\<ss' ;
    KEY CTRL+S, '^S'
DEFINE BAR 4 OF conversion PROMPT 'Spee\<d' ;
    KEY CTRL+D, '^D'
DEFINE BAR 5 OF conversion PROMPT '\<Temperature' ;
    KEY CTRL+T, '^T'
DEFINE BAR 6 OF conversion PROMPT 'T\<ime' ;
    KEY CTRL+I, '^I'
DEFINE BAR 7 OF conversion PROMPT 'Volu\<me' ;
    KEY CTRL+M, '^M'

ON SELECTION POPUP conversion;
    DO choice IN onpad WITH PROMPT( ), POPUP( )
```

```

DEFINE POPUP cardinfo MARGIN RELATIVE SHADOW COLOR SCHEME 4
DEFINE BAR 1 OF cardinfo PROMPT '\<View Charges' ;
    KEY ALT+V, ''
DEFINE BAR 2 OF cardinfo PROMPT 'View \<Payments' ;
    KEY ALT+P, ''
DEFINE BAR 3 OF cardinfo PROMPT 'Vie\<w Users' ;
    KEY ALT+W, ''
DEFINE BAR 4 OF cardinfo PROMPT '\-'
DEFINE BAR 5 OF cardinfo PROMPT '\<Charges '
DEFINE BAR 6 OF cardinfo PROMPT '\-'
DEFINE BAR 7 OF cardinfo PROMPT 'E\<xit '

ON SELECTION POPUP cardinfo;
    DO choice IN onpad WITH PROMPT( ), POPUP( )

PROCEDURE choice
PARAMETERS mprompt, mpopup

WAIT WINDOW 'You chose ' + mprompt + ;
    ' from popup ' + mpopup NOWAIT

IF mprompt = 'Exit'
    SET SYSMENU TO DEFAULT
ENDIF

```

ON PAGE

Purpose	Executes a command when printed output reaches a specified line number
Syntax	ON PAGE [AT LINE <expN> <command>]
See Also	EJECT PAGE

Description When a specified line number is reached in a report, or when EJECT PAGE is issued, FoxPro executes the command you specify with ON PAGE.

Clause **AT LINE <expN> <command>**

The system variable _PLINENO keeps track of the current line number in a report. The specified command executes when _PLINENO becomes greater than the line number <expN> specified in ON PAGE. ON PAGE typically uses DO to execute a procedure for handling page breaks, headers and footers.

The command specified with ON PAGE also executes when EJECT PAGE is issued. See EJECT PAGE in this chapter for more information.

Executing ON PAGE without the AT LINE <expN> clause clears the previous ON PAGE.

ON READERROR

Purpose	Specifies a command that executes in response to a data input error
Syntax	ON READERROR [<command>]
See Also	@ ... GET, READ

Description When an input error occurs during a READ, FoxPro executes the command you specify with ON READERROR. Input errors that ON READERROR traps for include:

- Invalid dates
- Input that falls outside of a range defined with
@ ... GET ... RANGE
- Input that doesn't meet a condition defined with
@ ... GET ... VALID

Clause <command>

The command specified with ON READERROR executes when one of the above input errors occur. ON READERROR typically uses DO to execute a procedure that prompts the user for correct data.

Use ON READERROR without <command> to clear the previous ON READERROR.

Example In this example, a message is displayed if a price is entered that is lower than cost or greater than 3 times the cost.

```
CLOSE DATABASES
SET STATUS OFF
SET TALK OFF
USE parts
ON READERROR DO errhand
@ 10,13 SAY 'Part Number: ' GET pno
@ 12,13 SAY 'Cost: ' GET cost
@ 14,13 SAY 'Selling Price: '
@ 14,28 GET price PICTURE '999.99' RANGE cost,(cost + (cost * 3))
READ
```

ON READERROR

```
PROCEDURE errhand
IF price < cost
    WAIT WINDOW "Price can't be lower than cost"

ELSE
    WAIT WINDOW "Markup is too high"
ENDIF
RETURN
```


ON SELECTION BAR

Purpose	Specifies a command that executes when you choose a specific popup option
Syntax	ON SELECTION BAR <expN> OF <popup name> [<command>]
See Also	DEFINE BAR, DEFINE POPUP, ON BAR, ON SELECTION POPUP

Description When a specific popup option is chosen, FoxPro executes the command you specify with ON SELECTION BAR. Typically, ON SELECTION BAR uses DO to execute a procedure or program. Place ON SELECTION BAR between the DEFINE POPUP and the ACTIVATE POPUP commands when you create and activate the popup.

You can also use ON SELECTION BAR with a FoxPro menu system popup. Use ON SELECTION POPUP to execute a command when you choose any option in a popup. Use ON BAR to activate a menu bar or a popup when you chose a specific option in a popup.

Clauses **<expN> OF <popup name> [<command>]**

Include the option's bar number <expN>, the popup name <popup name>, and the command to execute for the specified popup option.

Use ON SELECTION BAR <expN> OF <popup name> without a command to release a command assigned to a popup option.

ON SELECTION MENU

Purpose	Specifies a command that executes when you choose any pad on a menu bar
Syntax	ON SELECTION MENU <menu name> ALL [<command>]
See Also	ACTIVATE MENU, DEFINE MENU, ON PAD, ON SELECTION PAD

Description When you choose any pad on a menu bar, FoxPro executes the command you specify with ON SELECTION MENU. Place ON SELECTION MENU between the DEFINE MENU and the ACTIVATE MENU commands when you create the menu.

Use ON SELECTION PAD to execute a command when a specific menu pad is chosen. An ON SELECTION PAD takes precedence over an ON SELECTION MENU. Use ON PAD to activate a menu bar or a popup when a specific menu pad is chosen.

Clauses **<menu name> | ALL**

The command executes when you choose a menu pad from the menu bar whose name you specify with <menu name>. If you include ALL instead of a menu name, the command executes when you choose any menu pad from any menu bar.

<menu name> is the name of a menu defined with DEFINE MENU or the FoxPro system menu.

<command>

Specify with <command> a command to execute when you choose a menu pad. Use ON SELECTION MENU without a command to release a command assigned to a menu bar.

Example ON SELECTION MENU is used in this program example to execute a procedure when a menu pad is chosen from the FoxPro system menu.

The current system menu bar is saved to memory with SET SYSMENU SAVE and all system menu pads are removed with SET SYSMENU TO.

DEFINE PAD creates several system menu pads. When you choose a menu pad the CHOICE procedure assigned to the menu bar with ON SELECTION MENU executes. CHOICE displays the name of the menu pad you choose and the name of the menu bar. If you choose the **Exit** menu pad, the original FoxPro system menu is restored.

```
*** Name this program ONMENU.PRG ***
```

```
CLEAR
```

```
SET SYSMENU SAVE
```

```
SET SYSMENU TO
```

```
DEFINE PAD syspad    OF _MSYSMENU PROMPT '\<System'  COLOR SCHEME 3 ;
    KEY ALT+S, ''
```

```
DEFINE PAD editpad   OF _MSYSMENU PROMPT '\<Edit'     COLOR SCHEME 3 ;
    KEY ALT+E, ''
```

```
DEFINE PAD recordpad OF _MSYSMENU PROMPT '\<Record'   COLOR SCHEME 3 ;
    KEY ALT+R, ''
```

```
DEFINE PAD windowpad OF _MSYSMENU PROMPT '\<window'   COLOR SCHEME 3 ;
    KEY ALT+W, ''
```

```
DEFINE PAD reportpad OF _MSYSMENU PROMPT 'Re\<ports'  COLOR SCHEME 3 ;
    KEY ALT+P, ''
```

```
DEFINE PAD exitpad   OF _MSYSMENU PROMPT 'E\<xit'     COLOR SCHEME 3 ;
    KEY ALT+X, ''
```

```
ON SELECTION MENU _MSYSMENU ;
```

```
    DO choice IN ormenu WITH PAD( ), MENU( )
```

```
PROCEDURE choice
```

```
    PARAMETER mpad, mmenu
```

```
    WAIT WINDOW 'You chose ' + mpad + ;
```

```
        ' from menu ' + mmenu NOWAIT
```

```
    IF mpad = 'EXITPAD'
```

```
        SET SYSMENU TO DEFAULT
```

```
    ENDIF
```

ON SELECTION PAD

Purpose	Specifies a command that executes when you choose a specific menu bar pad
Syntax	ON SELECTION PAD <pad name> OF <menu name> [<command>]
See Also	ACTIVATE MENU, DEFINE MENU, DEFINE PAD, ON PAD, ON SELECTION MENU
Description	When the specified menu bar pad is chosen, FoxPro executes the command you specify with ON SELECTION PAD. Typically, ON SELECTION PAD uses DO to execute a procedure or program when a specific menu pad is chosen from the menu bar. Place ON SELECTION PAD between the DEFINE MENU and the ACTIVATE MENU commands when you create and activate the menu bar. Use ON SELECTION MENU to execute a command when you choose any menu pad in a menu bar. Use ON PAD to activate a menu bar or a popup when you chose a menu pad in a menu bar.
Clauses	<pad name> OF <menu name> [<command>] Include the pad name and menu bar name to execute the command for the specified menu pad. Use ON SELECTION PAD without a command to release a command assigned to the menu pad.

ON SELECTION POPUP

Purpose	Specifies a command that executes when you choose any option from a specified popup or from all popups
Syntax	ON SELECTION POPUP <popup name> ALL [<command>]
See Also	ACTIVATE POPUP, DEFINE BAR, DEFINE POPUP, ON BAR

Description When you choose any option from a popup, FoxPro executes the command you specify with ON SELECTION POPUP. Place ON SELECTION POPUP between the DEFINE POPUP and the ACTIVATE POPUP commands when you create the popup.

Use ON SELECTION BAR to execute a command when a specific popup option is chosen. An ON SELECTION BAR takes precedence over an ON SELECTION POPUP. Use ON BAR to activate a menu bar or a popup when a specific popup option is chosen.

Clauses <popup name> | **ALL**

The command is executed when you choose an option from the popup you specify with <popup name>. If you include ALL instead of a popup name, the command is executed when you choose an option from any popup.

<command>

Specify a command to execute when you choose a popup option. Use ON SELECTION POPUP without a command to release a command assigned to a popup option with a previous ON SELECTION POPUP.

Example The following program example, named SYSTIME.PRG, places a time and date pad in the system menu bar. The display of clock, date and day of the week can be toggled on and off from within the popup named TIMEPOP.

ON SELECTION POPUP is used in this example to execute the procedure named TOGGLE. When an option is chosen from the TIMEPOP popup, the TOGGLE routine executes and BAR() passes a bar number to the routine.

SET MARK OF and MRKBAR() display or clear a mark character for each option as its associated display is toggled on or off.

In FoxPro for Windows, the options are placed on the **Help** menu.

ON SELECTION POPUP

```
*** Name the program systime.prg ***

SET TALK OFF
SET CLOCK OFF

DEFINE BAR 1 BEFORE _MST_FILER OF _MSYSTEM ;
    PROMPT '\<Time & Date' KEY SHIFT+F2, 'F2 '

DEFINE POPUP timepop MARGIN && Menu popup for System menu pad
DEFINE BAR 1 OF timepop PROMPT '\<Clock '
DEFINE BAR 2 OF timepop PROMPT '\<Date '
DEFINE BAR 3 OF timepop PROMPT 'Da\<y '
DEFINE BAR 4 OF timepop PROMPT '\-' && Draw a separator
DEFINE BAR 5 OF timepop PROMPT '\<Show All '
DEFINE BAR 6 OF timepop PROMPT 'Clear \<All '
ON BAR 1 OF _MSYSTEM ACTIVATE POPUP timepop
ON SELECTION POPUP timepop DO toggle IN systime WITH BAR( )

PROCEDURE toggle
PARAMETER barnumber
clockcolor = SCHEME(1,7)
DO CASE && CASE for each option
CASE barnumber = 1    && Clock option
    *** Set clock on or off, toggle mark character ***
    SET MARK OF BAR 1 OF timepop TO NOT MRKBAR('timepop', 1)
    IF MRKBAR('timepop', 1) = .T.
        SET CLOCK ON
    ELSE
        SET CLOCK OFF
    ENDIF
CASE barnumber = 2    && Date option
    *** Show or clear date, toggle mark character ***
    SET MARK OF BAR 2 OF timepop TO NOT MRKBAR('timepop', 2)
    IF MRKBAR('timepop', 2) = .T.
        @ 1,70 SAY DATE( ) COLOR (clockcolor)
    ELSE
        @ 1,68 CLEAR TO 1,79
    ENDIF
CASE barnumber = 3    && Day option
    *** Show or clear day, toggle mark character ***
    SET MARK OF BAR 3 OF timepop TO NOT MRKBAR('timepop', 3)
    IF MRKBAR('timepop', 3) = .T.
        @ 2,70 SAY CDOW(DATE( )) COLOR (clockcolor)
    ELSE
        @ 2,70 CLEAR TO 2,79
    ENDIF
```

```
CASE barnumber = 5    && Show all option
  *** Set all marks to .T., turn on clock, show date and day ***
  SET MARK OF BAR 1 OF timepop TO .T.
  SET MARK OF BAR 2 OF timepop TO .T.
  SET MARK OF BAR 3 OF timepop TO .T.
  SET CLOCK ON
  @ 1,70 SAY DATE( ) COLOR (clockcolor)
  @ 2,70 SAY CDOW( DATE( )) COLOR (clockcolor)
CASE barnumber = 6    && Clear all option
  *** Set all marks to .F., turn off clock, clear date and day ***
  SET MARK OF BAR 1 OF timepop TO .F.
  SET MARK OF BAR 2 OF timepop TO .F.
  SET MARK OF BAR 3 OF timepop TO .F.
  SET CLOCK OFF
  @ 1,70 CLEAR TO 2,79
ENDCASE
RETURN
```

ON()

Purpose	Returns the command assigned to ON ERROR, ON ESCAPE, ON KEY, ON KEY LABEL, ON PAGE or ON READERROR
Syntax	ON(<expC1> [, <expC2>])
Parameters	<expC1> <expC2>
Returns	Character
See Also	INKEY(), LASTKEY(), ON ERROR, ON ESCAPE, ON KEY, ON KEY LABEL, ON PAGE, ON READERROR, READKEY()

Description A command can be assigned to each of these event handling commands:

- ON ERROR
- ON ESCAPE
- ON KEY
- ON KEY LABEL
- ON PAGE
- ON READERROR

When an event occurs that is trapped by one of the above event-handling commands, the command assigned by the event-handling command executes. ON() returns the command you assigned to an event-handling command. The null string is returned if a command isn't currently assigned to the event-handling command you specify.

Parameters <expC1>

The character expression <expC1> specifies one of the above event-handling commands.

Here are the commands and the corresponding character expressions to use with ON():

Command	<expC1>
ON ERROR	ERROR
ON ESCAPE	ESCAPE
ON KEY	KEY
ON KEY LABEL	KEY
ON PAGE	PAGE
ON READERROR	READERROR

For example, to return the command currently assigned to ON ERROR, use

```
? ON('ERROR')
```

<expC2>

The second character expression <expC2> is included to return the command assigned to a specific key or key combination in ON KEY LABEL. Specify "KEY" in <expC1> and the key label name of the key or key combination in <expC2>. A complete listing of key label names is available in ON KEY LABEL in this chapter.

For example, to return the command currently assigned to the F7 function key with ON KEY LABEL, use

```
? ON('KEY', 'F7')
```

Example

The following example uses ON() to display ON ERROR and ON KEY LABEL settings.

```
ON ERROR DO errorhand
ON KEY LABEL CTRL+F2 WAIT WINDOW 'You pressed ^F2'
ON KEY LABEL ALT+Z DISPLAY MEMORY
```

```
? ON('ERROR')
? ON('KEY', 'CTRL+F2')
? ON('KEY', 'ALT+Z')
```

```
ON ERROR
ON KEY LABEL CTRL+F2
ON KEY LABEL ALT+Z
```

ORDER()

Purpose	Returns the name of the controlling index file or tag for the current or specified table/.DBF
Syntax	ORDER([<expN1> <expC> [, <expN2>]])
Parameters	<expN1> Work area number <expC> Work area alias <expN2> Include to return the drive and directory
Returns	Character
See Also	INDEX, SET INDEX, SET ORDER, USE

Description A table/.DBF can have several index files open simultaneously. However, only one single-entry index file (the controlling index file) or tag from a compound index file (the controlling tag) controls the order in which the table/.DBF is displayed or accessed. Certain commands, such as SEEK, use the controlling index file or controlling tag to search for records. This function returns the name of the controlling index file or controlling tag.

USE and SET INDEX both support opening more than one index, with an index file list. A controlling index file or controlling tag can be designated in this index file list. SET ORDER can also be used to designate a controlling index or controlling tag.

Parameters <expN1> | <expC>

ORDER() by default returns the controlling index file name or controlling tag name for the current work area.

You can return the controlling index file name or controlling tag name from unselected work areas by specifying the work area number <expN1> or the table/.DBF or work area alias <expC>.

<expN2>

If a table/.DBF has a controlling index file or tag, and you include <expN2>, the drive and directory is returned along with the single-entry or compound index file name. The numeric expression <expN2> can have any value.

Example The following examples display index tags and files.

```
CLOSE DATABASES
SELECT 0
USE customer ORDER on
? ORDER( )
ON

? ORDER('customer', 1)
C:\FOXPROW\CUSTOMER.CDX
```

OS()

Purpose	Returns the name and version number of the operating system under which FoxPro is running
Syntax	OS()
Returns	Character
See Also	DISKSPACE(), GETENV(), VERSION()

Description OS() returns the name and version number of the host operating system.

Example ? OS()
 DOS 05.00

PACK

Purpose	Removes all records marked for deletion, and reduces the size of a memo file
Syntax	PACK [MEMO] [DBF]
See Also	DELETE, DELETED(), RECALL, SET EXCLUSIVE

Description PACK permanently removes all records marked for deletion in the current table/.DBF. PACK also reduces the size of a memo file associated with a table/.DBF.

PACK removes records marked for deletion from the current table/.DBF. Records can be marked for deletion with DELETE, by choosing the **Delete...** option on the **Record** menu, or from within a Browse window. When PACK is issued without the MEMO and DBF clauses, the table/.DBF and the memo file are both packed.

PACK requires exclusive use of the table/.DBF. See SET EXCLUSIVE in this chapter for more information about opening a table/.DBF exclusively on a network.



Exercise caution when using PACK. There is no way to retrieve deleted records after the pack is finished.

When PACK is issued, FoxPro copies all records not marked for deletion to a temporary table/.DBF. When the PACK is completed, FoxPro deletes the original table/.DBF from disk and renames the temporary table/.DBF with the original table/.DBF name. PACK can be interrupted by pressing Escape. Pressing Escape stops the PACK, the temporary table/.DBF is deleted and the original table/.DBF remains unchanged. The original table/.DBF is also recovered if you run out of disk space during a PACK.

If the current table/.DBF has one or more indexes open, PACK rebuilds the index files.

Clauses MEMO

Information in memo fields is stored in an associated memo file. A memo file has the same name as the table/.DBF and an .FPT extension.

Include the MEMO clause to pack the memo file without packing the table/.DBF file. PACK MEMO removes unused space from the memo file, but doesn't remove records marked for deletion from the table/.DBF.

DBF

If you include the DBF clause but not the MEMO clause, the table/.DBF is packed without packing the memo file.

PAD()

Purpose	Returns the name of the menu pad that was most recently chosen from a menu bar
Syntax	PAD()
Returns	Character
See Also	ACTIVATE MENU, BAR(), DEFINE PAD, DEFINE MENU, MENU(), ON PAD, ON SELECTION PAD, PROMPT()
Description	PAD() returns the name of the pad chosen from a menu bar as an upper-case character string. A menu bar must be defined and active for PAD() to return a pad name. Menu bars are defined and activated with DEFINE MENU and ACTIVATE MENU. You can also use PAD() with the FoxPro system menu bar. PAD() returns the null string if a menu isn't active or PAD() is issued from the Command window.
Example	PAD() is used in this program example to pass the name of a menu pad to a procedure. The current system menu bar is saved to memory with SET SYSMENU SAVE, and all system menu pads are removed with SET SYSMENU TO. Several system menu pads are created with DEFINE PAD. When a menu pad is chosen, PAD() is used to pass the name of the menu pad to the procedure named CHOICE. CHOICE displays the name of the menu pad you choose and the name of the menu bar. If you choose the Exit menu pad, the original FoxPro system menu is restored.

```

*** Name this program PADEXAM.PRG ***

CLEAR

SET SYSMENU SAVE
SET SYSMENU TO

DEFINE PAD syspad    OF _MSYSMENU PROMPT '\<System'  COLOR SCHEME 3 ;
    KEY ALT+S, ''
DEFINE PAD editpad    OF _MSYSMENU PROMPT '\<Edit'      COLOR SCHEME 3 ;
    KEY ALT+E, ''
DEFINE PAD recordpad  OF _MSYSMENU PROMPT '\<Record'    COLOR SCHEME 3 ;
    KEY ALT+R, ''
DEFINE PAD windowpad  OF _MSYSMENU PROMPT '\<Window'    COLOR SCHEME 3 ;
    KEY ALT+W, ''
DEFINE PAD reportpad  OF _MSYSMENU PROMPT 'Re\<ports'   COLOR SCHEME 3 ;
    KEY ALT+P, ''
DEFINE PAD exitpad    OF _MSYSMENU PROMPT 'E\<xit'      COLOR SCHEME 3 ;
    KEY ALT+X, ''

ON SELECTION MENU _MSYSMENU ;
    DO choice IN padexam WITH PAD( ), MENU( )

PROCEDURE choice
PARAMETER mpad, mmenu

WAIT WINDOW 'You chose ' + mpad + ;
    ' from menu ' + mmenu NOWAIT

IF mpad = 'EXITPAD'
    SET SYSMENU TO DEFAULT
ENDIF

```


PADC(), PADL(), PADR()

Purpose	Returns the specified expression padded on the left, right, or both left and right sides
Syntax	PADL(<expr>, <expN> [, <expC>]) PADR(<expr>, <expN> [, <expC>]) PADC(<expr>, <expN> [, <expC>])
Parameters	<expr> The expression to pad <expN> The total length of the expression after it is padded <expC> The characters used as padding
Returns	Character
See Also	ALLTRIM(), LEFT(), LTRIM(), RIGHT(), STUFF(), TRIM()

Description These functions return a character string from a character expression, padded to a specified length.

PADL() inserts padding on the left, PADR() inserts padding on the right and PADC() inserts padding on both sides.

Parameters **<expr>**

Include the expression <expr> to be padded. This expression can be a character, numeric or date expression.

<expN>

<expN> specifies the total number of characters in the expression after it is padded.

<expC>

The value of <expC> is used for padding. If <expC> is omitted, spaces (ASC(32)) are used for padding. If <expC> is included, the value of <expC> is repeated as necessary to pad the expression <expr> to a length of <expN> characters.

PADC(), PADL(), PADR()

Examples

```
STORE 'TITLE' TO string
? PADL(string, 40, '=')
=====TITLE
? PADR(string, 40, '=')
TITLE=====
? PADC(string, 40, '=')
=====TITLE=====
```

PARAMETERS

Purpose	Assigns data passed from a calling program to local memory variables
Syntax	PARAMETERS <parameter list>
See Also	DO, PARAMETERS(), SET UDFPARMS
Description	PARAMETERS assigns data passed from a calling program to local memory variables or arrays. PARAMETERS using DO ... WITH must be the first executable statement in the called program, procedure, or user-defined function (UDF). By default, DO ... WITH passes variables and arrays to programs by reference. When a value is changed in the called program, the new value is passed back to the associated variable in the calling program. If you want to pass a variable or array by value, enclose the variable or array in parentheses in the DO ... WITH parameter list. Any changes made to the parameter in the called program will not be passed back to the calling program. Variables and arrays are by default passed by value to a UDF. Use SET UDFPARMS REFERENCE to pass variables and arrays to a UDF by reference.
Clause	<parameter list> The parameter names listed in <parameter list> are memory variable or array names. Parameters within <parameter list> are separated by commas. There must be at least as many parameters in the PARAMETERS statement as in the DO ... WITH statement. If more variables or arrays are listed in the PARAMETERS statement than are passed by DO ... WITH, the remaining variables or arrays are initialized to false (.F.). A maximum of 24 parameters can be passed. PARAMETERS() returns the number of parameters that were passed to the most recently executed routine.

Example The following example passes parameters to an error-handling routine.

```
ON ERROR DO errhand WITH ERROR( ), MESSAGE( ), ;  
    MESSAGE(1), PROGRAM( ), LINENO( )
```

```
USE nodatabase
```

```
ON ERROR && restores system error handling routine
```

```
PROCEDURE errhand
```

```
PARAMETER merror, mess, mess1, mprog, mlineno
```

```
? 'Error number: ' + LTRIM(STR(merror))
```

```
? 'Error message: ' + mess
```

```
? 'Line of code with error: ' + mess1
```

```
? 'Line number of error: ' + LTRIM(STR(mlineno))
```

```
? 'Program with error: ' + mprog
```

PARAMETERS()

Purpose	Returns the number of parameters that were passed to the most recently called program procedure, or user-defined function (UDF)
Syntax	PARAMETERS()
Returns	Numeric
See Also	DO, FUNCTION, PARAMETERS, PROCEDURE, SET UDFPARMS, UDFs

Description PARAMETERS() returns a number indicating the number of parameters that were passed to the program, procedure, or UDF that most recently executed. This function is useful in situations where a variable number of parameters are passed to a procedure or UDF.

Note that the value returned by PARAMETERS() is reset every time a procedure or function is called or ON KEY LABEL is executed.

Example The following example calls a procedure and displays in a wait window the number of parameters passed.

```
DO testpar WITH 1,2,3
PROCEDURE testpar
PARAMETERS s1,s2,s3
msg = 'PARAMETERS( ) =' + ALLTRIM(STR(PARAMETERS( )))
WAIT WINDOW (msg)
RETURN
```

PARAMETERS()

Example The following example uses a procedure to display the average of 4 values.

```
SET TALK OFF
val1 = 10
val2 = 20
val3 = 30
val4 = 15
m_min = getavg(val1, val2, val3, val4)
? 'Average value is '
?? m_min
```

- * This UDF permits up to 9 parameters to be passed.
- * It uses the PARAMETER() function to determine how many
- * were passed and returns the average value.

```
FUNCTION getavg
PARAMETER para1,para2,para3,para4,para5,para6,para7,para8,para9
IF PARAMETERS( ) = 0
    RETURN 0
ENDIF
mresult = 0
FOR mcount = 1 to PARAMETERS( )
    mcompare = 'para' +(STR(mcount,1))
    mresult = mresult + EVAL(mcompare)
ENDFOR
mresult = mresult / (mcount - 1)
RETURN mresult
```

Executing this program returns:

Average value is 18.75

PAYMENT()

Purpose	Returns the amount of each periodic payment on a fixed-interest loan
Syntax	PAYMENT(<expN1>, <expN2>, <expN3>)
Parameters	<expN1> The beginning loan principal <expN2> The fixed interest rate per period <expN3> The total number of payments
Returns	Numeric
See Also	CALCULATE, FV(), PV()

Description	This financial function returns the amount of each payment on a loan. PAYMENT() assumes a constant periodic interest rate and that payments are made at the end of each period.
Parameters	<expN1> <expN1> is the beginning principal of the loan. <expN2> The fixed interest rate per period is <expN2>. If monthly payments are made on the loan but the interest rate is annual, divide the annual interest rate by 12. <expN3> The total number of payments to be made on the loan is <expN3>.
Example	<pre> STORE 100000 TO principal && \$100,000 beginning principal STORE .105/12 TO interest && 10.5% annual interest rate STORE (20*12) TO payments && 20 years of monthly payments ? PAYMENT(principal, interest, payments) 998.38 </pre>

PCOL()

Purpose	Returns the current column position of the printer output
Syntax	PCOL()
Returns	Numeric
See Also	COL(), ROW(), PROW()

Description PCOL() returns the current column position of the printer’s print head.

The value returned by PCOL() is relative to the current setting of the left printer margin. The left margin may be set with SET MARGIN or by storing a value to the system memory variable _PLOFFSET.

PCOL() is especially useful for relative addressing of printed text.

You can use the \$ operator in place of PCOL().

Examples @ PROW(),PCOL()+12 SAY ‘Contact person’
 @ PROW(),\$+12 SAY ‘Contact person’

PI()

Purpose	Returns the numeric constant π
Syntax	PI()
Returns	Numeric
See Also	SET DECIMALS

Description This mathematical function returns the numeric constant π (≈ 3.141592). π is the ratio of the circumference of a circle to its diameter.

The number of decimal places displayed in the value returned by PI() is determined by SET DECIMALS.

Examples

```
? PI( )
3.14

STORE 2.30 TO radius
STORE PI( ) * radius^2 TO area
? area
16.6190
```

PLAY MACRO

Purpose	Executes a keyboard macro
Syntax	PLAY MACRO <macro name> [TIME <expN>]
See Also	CLEAR MACROS, ON KEY LABEL, RESTORE MACROS, SAVE MACROS

Description With FoxPro you can save a series of keystrokes as a keyboard macro by pressing Shift-F10. **PLAY MACRO** plays this series of keystrokes. Playing keyboard macros within programs lets you create self-running demonstration programs.

Once a keyboard macro has been stored in memory with **RESTORE MACRO**, or by creating the macro with Shift-F10, it can also be played back by pressing the key or keys you specified when you created the macro. For more information on creating macros, see the *FoxPro User's Guide*.

If **PLAY MACRO** is issued from the Command window, it executes immediately. If **PLAY MACRO** is issued in a program, execution is delayed until the program executes a command that allows keyboard input. Examples of commands that wait for input are @ ... GET, BROWSE, and EDIT.



If a series of **PLAY MACROS** are pending, FoxPro doesn't execute them in the order in which they are issued. Macros are played within a program in *reverse order* — the first **PLAY MACRO** executes last, the last **PLAY MACRO** executes first. The **PLAY MACROS** execute from bottom to top order as they appear within the program.

Clauses **<macro name>**

The name of the keyboard macro to play is specified with <macro name>.

TIME <expN>

The **TIME** clause specifies the time interval between the delivery of each keystroke in a keyboard macro. Specify a time delay between 0 and 10 seconds in <expN>. <expN> can evaluate to a number with a decimal fraction. For example, if you specify <expN> equal to 1, keystrokes from the macro will be played with a one-second delay between each keystroke.

POP KEY

Purpose	Restores ON KEY LABELs that were placed on the stack with PUSH KEY
Syntax	POP KEY [ALL]
See Also	ON KEY LABEL, ON(), PUSH KEY
Description	ON KEY LABELs are placed on the stack with PUSH KEY. POP KEY, when used with PUSH KEY, lets you save a set of key assignments defined with ON KEY LABELs, make changes to the key assignments with more ON KEY LABELs, and then restore the original key assignments. For additional information on placing ON KEY LABELs on the stack, see the PUSH KEY in this manual.
Clause	ALL Including the ALL clause with POP KEY clears all current key assignments defined with ON KEY LABELs, and clears all key assignments defined with ON KEY LABELs from the stack.

POP MENU

Purpose	Restores the specified menu bar definition that was placed on the stack with PUSH MENU
Syntax	POP MENU <menu name> [TO MASTER]
See Also	ACTIVATE MENU, DEFINE MENU, POP POPUP, PUSH MENU, PUSH POPUP

Description POP MENU, when used with PUSH MENU, lets you save a menu definition, make changes to the menu definition and then restore the menu definition to its original state.

Menus are placed on and removed from the stack in last in, first out order.

Clauses **<menu name>**

<menu name> specifies the name of the menu whose definition is pulled off the stack. The specified menu can be a user-defined menu or the FoxPro system menu.

TO MASTER

Including the TO MASTER clause restores the first menu definition pushed onto the menu stack, and then clears the stack.

Example Type the following commands in the command window. In this example, the system menu bar definition is pushed to the stack and then modified. The original system menu definition is then restored by popping it off the stack.

```
PUSH MENU _MSYSMENU
SET SYSMENU TO _MFILE, _MEDIT
POP MENU _MSYSMENU
```

POP POPUP

Purpose	Restores the specified popup definition that was placed on the stack with PUSH POPUP
Syntax	POP POPUP <popup name>
See Also	ACTIVATE POPUP, DEFINE POPUP, POP MENU, PUSH MENU, PUSH POPUP
Description	POP POPUP, when used with PUSH POPUP, lets you save a popup definition, make changes to the popup definition, and then restore the popup definition to its original state. Popup definitions are placed on and removed from the stack in a last in, first out order.
Clause	<popup name> <popup name> specifies the name of the popup whose definition is pulled off the stack. The specified popup can be a popup defined with DEFINE POPUP or a FoxPro system popup.
Example	In the following example, a popup named POPEXAM is created. The popup definition is pushed to the stack and then one of the popup options is modified. The original popup definition is then restored by popping it off the stack. <pre> DEFINE POPUP popexam FROM 5,5 DEFINE BAR 1 OF popexam PROMPT 'One' DEFINE BAR 2 OF popexam PROMPT 'Two' DEFINE BAR 3 OF popexam PROMPT 'Three' DEFINE BAR 4 OF popexam PROMPT 'Four' ACTIVATE POPUP popexam NOWAIT PUSH POPUP popexam WAIT 'Popup pushed' WINDOW RELEASE BAR 2 OF popexam WAIT 'This is the modified Popup' WINDOW POP POPUP popexam WAIT 'Popup popped, original popup restored' WINDOW DEACTIVATE POPUP popexam RELEASE POPUP popexam </pre>

POPUP()

Purpose	Returns as a character string the name of the active popup
Syntax	POPUP()
Returns	Character
See Also	ACTIVATE POPUP, BAR(), DEFINE BAR, DEFINE POPUP, ON SELECTION POPUP

Description POPUP() returns, as a character string, the name of the active popup.

A popup must be defined and active for POPUP() to return its name. Poppers are defined and activated with DEFINE POPUP and ACTIVATE POPUP. The popup can also be a FoxPro system popup. If a popup isn't active or POPUP() is issued from the Command window, POPUP() returns the null string.

Example POPUP() is used in this program example to pass the name of the popup you choose to a procedure.

The current system menu bar is first saved to memory with SET SYSMENU SAVE and then all system menu pads are removed with SET SYSMENU TO.

Two new system menu pads are created with DEFINE PAD, and DEFINE POPUP is used to create a popup for each menu pad. DEFINE BAR is used to create options in each of the popups. When a menu pad is chosen, ON PAD uses ACTIVATE POPUP to activate the corresponding popup.

When an option is chosen from a popup, ON SELECTION POPUP uses PROMPT() and POPUP() to pass the option number and popup name to the procedure named CHOICE. CHOICE displays the prompt of the option you choose and the name of the popup containing the option. If you choose the **Exit** option in the popup named CARDINFO, the original FoxPro system menu is restored.

```
*** Name this program POPEXAMP.PRG ***
```

```
CLEAR
```

```
SET SYSMENU SAVE
```

```
SET SYSMENU TO
```

```
DEFINE PAD convpad OF _MSYSMENU PROMPT '\<Conversions' COLOR SCHEME 3 ;  
    KEY ALT+C, ''
```

```
DEFINE PAD cardpad OF _MSYSMENU PROMPT 'Card \<Info' COLOR SCHEME 3 ;  
    KEY ALT+I, ''
```

```
ON PAD convpad OF _MSYSMENU ACTIVATE POPUP conversion
```

```
ON PAD cardpad OF _MSYSMENU ACTIVATE POPUP cardinfo
```

```
DEFINE POPUP conversion MARGIN RELATIVE SHADOW COLOR SCHEME 4
```

```
DEFINE BAR 1 OF conversion PROMPT 'Ar\<ea' ;
```

```
    KEY CTRL+E, '^E'
```

```
DEFINE BAR 2 OF conversion PROMPT '\<Length' ;
```

```
    KEY CTRL+L, '^L'
```

```
DEFINE BAR 3 OF conversion PROMPT 'Ma\<ss' ;
```

```
    KEY CTRL+S, '^S'
```

```
DEFINE BAR 4 OF conversion PROMPT 'Spee\<d' ;
```

```
    KEY CTRL+D, '^D'
```

```
DEFINE BAR 5 OF conversion PROMPT '\<Temperature' ;
```

```
    KEY CTRL+T, '^T'
```

```
DEFINE BAR 6 OF conversion PROMPT 'T\<ime' ;
```

```
    KEY CTRL+I, '^I'
```

```
DEFINE BAR 7 OF conversion PROMPT 'Volu\<me' ;
```

```
    KEY CTRL+M, '^M'
```

```
ON SELECTION POPUP conversion;
```

```
    DO choice IN popexamp WITH PROMPT( ), POPUP( )
```

POPUP()

```
DEFINE POPUP cardinfo MARGIN RELATIVE SHADOW COLOR SCHEME 4
DEFINE BAR 1 OF cardinfo PROMPT '\<View Charges' ;
    KEY ALT+V, ''
DEFINE BAR 2 OF cardinfo PROMPT 'View \<Payments' ;
    KEY ALT+P, ''
DEFINE BAR 3 OF cardinfo PROMPT 'Vie\<w Users' ;
    KEY ALT+W, ''
DEFINE BAR 4 OF cardinfo PROMPT '\-'
DEFINE BAR 5 OF cardinfo PROMPT '\<Charges '
DEFINE BAR 6 OF cardinfo PROMPT '\-'
DEFINE BAR 7 OF cardinfo PROMPT 'E\<xit '

ON SELECTION POPUP cardinfo;
    DO choice IN popexamp WITH PROMPT( ), POPUP( )

PROCEDURE choice
PARAMETERS mprompt, mpopup

WAIT WINDOW 'You chose ' + mprompt + ;
    ' from popup ' + mpopup NOWAIT

IF mprompt = 'Exit'
    SET SYSMENU TO DEFAULT
ENDIF
```


PRINTJOB ... ENDPRINTJOB

Purpose	Activates print job system memory variable settings
Syntax	PRINTJOB <statements> ENDPRINTJOB
See Also	ON PAGE

Description PRINTJOB initializes the printer and certain system memory variables that affect printed output. PRINTJOB can send control codes to the printer, eject a page from the printer before and/or after a print job, initialize the printer column number and control the number of copies printed.

Issuing PRINTJOB performs these tasks:

- Sends starting printer control codes that are stored in the system memory variable `_PCODE` to the printer. See the System Memory Variables chapter in this manual and check your printer manual for more information on printer control codes.
- Ejects a page if the system memory variable `_PEJECT` is set to `BEFORE` or to `BOTH`.
- Sets the system memory variable `_PCOLNO` to 0. `_PCOLNO` stores the printer column number.

Issuing ENDPRINTJOB performs these tasks:

- Sends ending printer control codes to the printer. You can reset the printer to the configuration it had before PRINTJOB was issued. The ending printer control codes are stored in the system memory variable `_PECODE`.
- Ejects a page if the system memory variable `_PEJECT` is set to `AFTER` or to `BOTH`.
- Loops back to PRINTJOB to print another copy of the report, if the `_PCOPIES` system memory variable is set to a value higher than 1 (the default). The number of copies is determined by the value of the system memory variable `_PCOPIES`. When the number of copies printed equals the value of `_PCOPIES`, FoxPro exits the loop. Program control then begins with the command immediately following ENDPRINTJOB.

PRINTJOB and ENDPRINTJOB may be executed only from within a program. You may not nest PRINTJOBS.

PRINTSTATUS()

Purpose	Returns true (.T.) if the printer or print device is on line
Syntax	PRINTSTATUS()
Returns	Logical
See Also	SET DEVICE, SET PRINTER, SYS(13)

Description This function returns true (.T.) if the printer or print device is ready, and false (.F.) if it isn't.

PRINTSTATUS() is similar to SYS(13), except SYS(13) returns READY or OFFLINE instead to true or false.

Examples

```
? PRINTSTATUS( )
.F.

*** Program Example ***

STORE PRINTSTATUS( ) TO ready
IF NOT ready
    WAIT 'Make sure printer is attached and turned on!' WINDOW
ELSE
    WAIT 'Printer is ready!' WINDOW
ENDIF
```

PRIVATE

Purpose	Hides specified memory variables or arrays that were defined in a calling program from the current program
Syntax	PRIVATE <memvar list> PRIVATE ALL [LIKE <skel> EXCEPT <skel>]
See Also	DIMENSION, PUBLIC

Description PRIVATE hides memory variables or arrays defined in a previously executed program from the current program. The hiding of variables created in higher-level programs enables variables of the same name as the private variables to be manipulated in the current program without affecting the values of the hidden variables. Once the program containing PRIVATE has completed execution, all memory variables and arrays that were declared private are again available.



PRIVATE doesn't create variables; it simply hides variables declared in higher-level programs from the current program.

- Clauses**
- <memvar list>**
<memvar list> specifies the memory variables or arrays to be declared private.
- ALL LIKE <skel>**
The ALL LIKE clause causes PRIVATE to hide all memory variables and arrays whose names match the skeleton <skel>. The skeleton can contain the wildcard characters ? and *.
- ALL EXCEPT <skel>**
The ALL EXCEPT clause causes PRIVATE to hide all memory variables or arrays unless their names match the skeleton <skel>. The skeleton can contain the wildcard characters ? and *.

PRIVATE

Example

*** Program example demonstrating PRIVATE ***

SET TALK OFF

val1 = 10

val2 = 15

DO down

? val1, ? val2

PROCEDURE down

PRIVATE val1

val1 = 50

val2 = 100

? 'Val1 Val2'

? val1, val2

RETURN

Val1	Val2
------	------

50	100
----	-----

10	100
----	-----

PRMBAR()

Purpose	Returns the prompt text from a bar on a popup
Syntax	PRMBAR(<expC>, <expN>)
Parameters	<expC> The popup name <expN> The bar number
Returns	Character
See Also	CNTBAR(), GETBAR(), DEFINE BAR, DEFINE POPUP, MRKBAR(), PRMPAD()

Description Popups are created with DEFINE POPUP and DEFINE BARS. DEFINE POPUP creates the popup, while DEFINE BARS create the popup bars. PRMBAR() also works with a FoxPro system popup. PRMBAR() returns the text that appears on the bar.

A popup doesn't have to be active for PRMBAR() to return a bar's prompt text.

If a popup bar was created with the hot key (\<) or disabled option (\) special characters aren't included, only the prompt is returned. The special characters aren't included. PRMBAR() returns the null string when a popup bar is a separator created with the \- special characters. The null string is also returned if a bar is created without a prompt.

Parameters <expC>

Specify the popup name with <expC>.

<expN>

Include the bar number <expN>. The prompt text of the bar with that number is returned. For example, if <expN> is 1 the first bar's prompt text is returned, if <expN> is 2 the second bar's prompt text is returned and so on. <expN> must be at least 1 and no greater than the number of bars in the popup.

Example

For an example using PRMBAR(), see CNTBAR() in this manual.

PRMPAD()

Purpose	Returns the text from a menu pad
Syntax	PRMPAD(<expC1>, <expC2>)
Parameters	<expC1> The menu bar name <expC2> The menu pad name
Returns	Character
See Also	CNTBAR(), GETBAR(), DEFINE BAR, DEFINE POPUP, MRKPAD(), PRMBAR()

Description Menu pads are created with DEFINE MENU and DEFINE PAD. DEFINE MENU creates the menu bar, and DEFINE PAD creates the menu pads. PRMPAD() also works with the FoxPro menu system. PRMPAD() returns the text that appears on the pad.

A menu bar doesn't have to be active for PRMPAD() to return a pad's prompt.

If a menu pad is created with the hot key (\<) or disabled (\) special characters, only the prompt is returned. The special characters aren't included. The null string is returned if a pad is created without a prompt.

Parameters **<expC1>**
Specify the menu bar name with <expC1>.

<expC2>
Specify the menu pad name in <expC2>.

Example

In the following example a menu bar named PRMEXAMPLE is created with three menu pads. The hot key and disabled option designators aren't returned from the pads named TWO and THREE. The menu is activated to show the pads and is cleared from the screen and from memory when a pad is chosen.

```
CLEAR
SET TALK OFF
STORE 'prmexample' TO mpopname

DEFINE MENU prmexample BAR AT LINE 1
DEFINE PAD one OF prmexample PROMPT 'This will be returned'
DEFINE PAD two OF prmexample PROMPT '\<As will this'
DEFINE PAD three OF prmexample PROMPT '\And this too'

@ 8,1 SAY PRMPAD('prmexample', 'one')
@ 9,1 SAY PRMPAD('prmexample', 'two')
@ 10,1 SAY PRMPAD(mpopname, 'three')

ACTIVATE MENU prmexample
DEACTIVATE MENU prmexample
RELEASE MENU prmexample
```

PROCEDURE

Purpose	Identifies the beginning of a procedure in a program file
Syntax	PROCEDURE <procedure name>
See Also	FUNCTION, PARAMETERS, PARAMETERS(), RETURN, SET PROCEDURE, SET UDFPARMS, User-defined Functions (UDFs)

Description In many programs, certain routines are frequently repeated. Define these commonly-used routines as separate procedures to reduce program size and complexity and to ease program maintenance.

PROCEDURE <procedure name> is a statement within a program file. It specifies the beginning of each procedure in a program file and defines the procedure name. Procedure names may be up to 10 characters long. They must begin with a letter or underscore and may contain any combination of letters, numbers and underscores.

The PROCEDURE <procedure name> command line is followed by a series of commands that make up the procedure. You may optionally include RETURN as the last line of a procedure, although an implicit RETURN is automatically executed following the last statement of a procedure.

When you execute a procedure with DO <procedure name>, FoxPro searches for the procedure in a specific order.

- First, the file containing DO <procedure name> is searched.
- Second, the file opened with SET PROCEDURE (if one is set) is searched.
- Third, FoxPro looks through the programs in the execution chain. Program files are searched beginning with the most recently executed program and continuing back to the first executed program.
- Fourth, FoxPro searches for a stand-alone program file. If a program file with the same name as the file name specified with DO is found, the program is executed. If a matching program file name isn't found, an error message "File doesn't exist" is returned.

Example

```
SET CENTURY ON
? longdate({08/19/92})
Wednesday, August 19, 1992
```

```
PROCEDURE longdate
PARAMETER mdate
RETURN CDOW(mdate) + ', ' + MDY(mdate)
```

PROGRAM()

Purpose	Returns the name of the program currently being executed, or the name of the program that was executing when an error occurred
Syntax	PROGRAM([<expN>])
Returns	Character
See Also	DO, LINENO(), MESSAGE, SYS(16)

Description PROGRAM() returns the name of the program currently being executed or the program that was executing when an error occurred. This function is useful when recovering from errors. It is similar to SYS(16).

Parameter <expN>

<expN> specifies how many program levels back the program name is to be fetched. This parameter can range from 0 to the program nesting depth. A program can execute another program which can execute another program and so on. Programs may be nested to 32 levels.

Specifying 0 or 1 with <expN> returns the name of the master program (the highest level program). If <expN> is omitted, the name of the currently executing program is returned. If <expN> exceeds the program nesting depth, the null string is returned.

Example

```
ON ERROR DO errhand WITH PROGRAM( )

*** The next line should generate an error

USE nodatabase
ON ERROR &&returns to system default error handling routine
PROCEDURE errhand
PARAMETERS mprogram

WAIT 'An error occurred in the program ' + mprogram WINDOW
```

PROMPT()

Purpose	Returns the prompt text of a pad chosen from a menu bar or an option chosen from a popup
Syntax	PROMPT()
Returns	Character
See Also	BAR(), DEFINE BAR, DEFINE MENU, DEFINE PAD, DEFINE POPUP, POPUP(), PRMBAR()

Description PROMPT() returns the prompt text of the pad last chosen from a menu bar or the option last chosen from a popup. If a menu bar or popup isn't active or Escape is pressed to exit a menu bar or popup, PROMPT() returns the null string.

A menu bar is created with DEFINE MENU, and each menu pad in the menu bar is created with DEFINE PAD.

A popup is created with DEFINE POPUP, and the options (bars) in the popup are created with the PROMPT clause of DEFINE POPUP or DEFINE BAR.

The menu bar or popup can also be the menu bar or a popup in the FoxPro menu system. MENU() and POPUP() may be used to return the name of the menu bar or popup from which the pad or option is chosen.

Example PROMPT() is used in this program example to pass the prompt of the popup option you choose to a procedure.

The current system menu bar is first saved to memory with SET SYSMENU SAVE and then all system menu pads are removed with SET SYSMENU TO.

Two new system menu pads are created with DEFINE PAD, and DEFINE POPUP is used to create a popup for each menu pad. DEFINE BAR is used to create options in each of the popups. When a menu pad is chosen, ON PAD uses ACTIVATE POPUP to activate the corresponding popup.

When an option is chosen from a popup, ON SELECTION POPUP uses PROMPT() and POPUP() to pass the option number and popup name to the procedure named CHOICE. CHOICE displays the prompt of the option you choose and the name of the popup containing the option. If you choose the **Exit** option in the popup named CARDINFO, the original FoxPro system menu is restored.

PROMPT()

```
*** Name this program PROMPTEX.PRG ***

CLEAR

SET SYSMENU SAVE
SET SYSMENU TO

DEFINE PAD convpad OF _MSYSMENU PROMPT '\<Conversions' COLOR SCHEME 3 ;
    KEY ALT+C, ''
DEFINE PAD cardpad OF _MSYSMENU PROMPT 'Card \<Info' COLOR SCHEME 3 ;
    KEY ALT+I, ''

ON PAD convpad OF _MSYSMENU ACTIVATE POPUP conversion
ON PAD cardpad OF _MSYSMENU ACTIVATE POPUP cardinfo

DEFINE POPUP conversion MARGIN RELATIVE SHADOW COLOR SCHEME 4
DEFINE BAR 1 OF conversion PROMPT 'Ar\<ea' ;
    KEY CTRL+E, '^E'
DEFINE BAR 2 OF conversion PROMPT '\<Length' ;
    KEY CTRL+L, '^L'
DEFINE BAR 3 OF conversion PROMPT 'Ma\<ss' ;
    KEY CTRL+S, '^S'
DEFINE BAR 4 OF conversion PROMPT 'Spee\<d' ;
    KEY CTRL+D, '^D'
DEFINE BAR 5 OF conversion PROMPT '\<Temperature' ;
    KEY CTRL+T, '^T'
DEFINE BAR 6 OF conversion PROMPT 'T\<ime' ;
    KEY CTRL+I, '^I'
DEFINE BAR 7 OF conversion PROMPT 'Volu\<me' ;
    KEY CTRL+M, '^M'

ON SELECTION POPUP conversion;
    DO choice IN promptex WITH PROMPT( ), POPUP( )
```

```

DEFINE POPUP cardinfo MARGIN RELATIVE SHADOW COLOR SCHEME 4
DEFINE BAR 1 OF cardinfo PROMPT '\<View Charges' ;
    KEY ALT+V, ''
DEFINE BAR 2 OF cardinfo PROMPT 'View \<Payments' ;
    KEY ALT+P, ''
DEFINE BAR 3 OF cardinfo PROMPT 'Vie\<w Users' ;
    KEY ALT+W, ''
DEFINE BAR 4 OF cardinfo PROMPT '\-'
DEFINE BAR 5 OF cardinfo PROMPT '\<Charges '
DEFINE BAR 6 OF cardinfo PROMPT '\-'
DEFINE BAR 7 OF cardinfo PROMPT 'E\<xit '

ON SELECTION POPUP cardinfo;
    DO choice IN promptex WITH PROMPT( ), POPUP( )

PROCEDURE choice
PARAMETERS mprompt, mpopup

WAIT WINDOW 'You chose ' + mprompt + ;
    ' from popup ' + mpopup NOWAIT

IF mprompt = 'Exit'
    SET SYSMENU TO DEFAULT
ENDIF

```

PROPER()

Purpose	Returns the specified character expression with each word having the initial letter capitalized and the remaining characters lowercase
Syntax	PROPER(<expC>)
Returns	Character
See Also	LOWER(), UPPER()

Description PROPER() returns from a character expression a character string capitalized as appropriate for proper names. Each word in the returned character string has its initial character capitalized and the remaining characters in lower-case.

Parameter **<expC>**
A properly capitalized character string is returned from the character expression <expC> you specify.

Examples STORE 'foxpro' TO expr1
? PROPER(expr1)
Foxpro

STORE 'FOXPRO' TO expr2
? PROPER(expr2)
Foxpro

To make sure that the names of all contacts in CUSTOMER.DBF have proper capitalization:

```
CLOSE DATABASES
USE customer
REPLACE ALL contact WITH PROPER(contact)
```

PROW()

Purpose	Returns the current row position of the printer output
Syntax	PROW()
Returns	Numeric
See Also	@ ... SAY, COL(), PCOL(), ROW()

Description PROW() returns the current row position of the printer's print head. Issuing EJECT resets PROW() to 0.

PROW() is especially useful for relative addressing of printed text.

Example @ PROW(), PCOL() + 12 SAY 'Contact person'
@ PROW(), \$+12 SAY 'Contact person'

The two commands above return the same result. You can use the \$ operator in place of PCOL(). Both \$ and PCOL() return the current column position of the printer.

PUBLIC

Purpose	Defines global memory variables or arrays
Syntax	PUBLIC <memvar list> PUBLIC [ARRAY] <array1> (<expN1> [, <expN2>]) [, <array2> (<expN3> [, <expN4>])] ...
See Also	DIMENSION, PARAMETERS, PARAMETERS(), PRIVATE, RELEASE

Description PUBLIC creates memory variables or arrays as global variables or global arrays. Global variables and arrays can be used and modified from any program that you execute during the current FoxPro session.

Memory variables and arrays that are created with PUBLIC are initialized false (.F.) except for the public variables FOX and FOXPRO, which are initialized true (.T.). The public variables FOX and FOXPRO can be used to conditionally execute code based on the product you are running.

Any memory variable or array you create in the Command window is automatically public.

Any memory variables or arrays you wish to declare as public must be declared public prior to assigning them a value.



If within a program you assign a value to a memory variable or array and later declare it public with PUBLIC, a syntax error will be generated.

Clauses **<memvar list>**
<memvar list> specifies the names of one or more memory variables to be initialized and designated as global.

**[ARRAY] <array1> (<expN1> [, <expN2>)
[, <array2> (<expN3> [, <expN4>])] ...**
The array clause specifies one or more arrays to be initialized and designated as global.

Example

```
SET TALK OFF
PUBLIC val1, val2
val1 = 10
val2 = 15

DO down

? val1
? val2

RELEASE ALL          && Releases private variables only
DISPLAY MEMORY LIKE val?
RELEASE val1, val2    && Public variables must be released explicitly
DISPLAY MEMORY LIKE val?

PROCEDURE down
PRIVATE val1
val1 = 50
val2 = 100
? val1
? val2
RETURN
```

PUSH KEY

Purpose	Places all current ON KEY LABELs on a stack in memory
Syntax	PUSH KEY [CLEAR]
See Also	ON KEY LABEL, ON(), POP KEY

Description PUSH KEY places all current ON KEY LABEL command settings on a stack in memory. PUSH KEY, when used with POP KEY, lets you save key assignments defined with ON KEY LABELs, change these assignments and then restore the previous assignments.

For example, you could use a new set of ON KEY LABELs when a Browse window is open. Before you open the Browse window, use PUSH KEY to save the current ON KEY LABEL key assignments in memory. You can then add or change ON KEY LABELs specifically for the Browse window. After you close the Browse window, your previous ON KEY LABEL key assignments can be restored from memory with POP KEY.

ON KEY LABELs are placed on and removed from the stack in a “last in, first out” order.

DISPLAY STATUS and LIST STATUS show the current key assignments defined with ON KEY LABELs.

CLEAR

If the CLEAR clause is included, all current ON KEY LABEL key assignments are saved on the stack, and the key assignments are then cleared.

PUSH MENU

Purpose	Places a menu bar definition on the stack
Syntax	PUSH MENU <menu name>
See Also	ACTIVATE MENU, DEFINE MENU
Description	PUSH MENU places a menu bar definition in a stack of menu definitions in memory. PUSH MENU, when used with POP MENU, lets you save a menu bar definition, make changes to the menu bar definition and then restore the menu bar definition to its original state. Menu bar definitions are placed on and removed from the stack in last in, first out order.
Clause	<menu name> <menu name> specifies the name of the menu bar whose definition is placed on the stack. The menu bar can be a user-defined menu or the FoxPro system menu bar.
Example	In the following example, the FoxPro system menu bar is pushed to the stack and then modified. The modified system menu bar is then restored by popping it off the stack. <pre>PUSH MENU _MSYSMENU SET SYSMENU TO _MFILE, _MEDIT POP MENU _MSYSMENU</pre>

PUSH POPUP

Purpose	Places a popup definition on the stack
Syntax	PUSH POPUP <popup name>
See Also	ACTIVATE POPUP, DEFINE POPUP

Description PUSH POPUP places a popup definition in a stack of popups. PUSH POPUP, when used with POP POPUP, lets you save a popup definition, make changes to the popup definition and then restore the popup definition to its original state.

Popup definitions are placed on and removed from the stack in last in, first out order.

Clause <popup name>

<popup name> specifies the name of the popup whose definition is placed on the stack. The popup can be a popup defined with DEFINE POPUP or a FoxPro system menu popup.

Example In the following example, a popup named POPEXAM is created. The popup definition is pushed to the stack and then modified. The original popup is then restored by popping it off the stack.

```
DEFINE POPUP popexam FROM 5,5

DEFINE BAR 1 OF popexam PROMPT 'One'
DEFINE BAR 2 OF popexam PROMPT 'Two'
DEFINE BAR 3 OF popexam PROMPT 'Three'
DEFINE BAR 4 OF popexam PROMPT 'Four'

ACTIVATE POPUP popexam NOWAIT
PUSH POPUP popexam

RELEASE BAR 2 OF popexam

POP POPUP popexam

DEACTIVATE POPUP popexam
RELEASE POPUP popexam
```

PUTFILE()

Purpose	Invokes the Save As dialog and returns the file name you specify
Syntax	PUTFILE([<expC1>] [, <expC2>] [, <expC3>])
Parameters	<expC1> The prompt text to be displayed above the list of existing file names <expC2> The default file name to be displayed in the text box <expC3> The extensions of the existing file names to be displayed in the list.
Returns	Character
See Also	FILE(), GETEXPR(), GETFILE(), LOCFILE()

Description Use PUTFILE() to display the FoxPro Save As dialog to choose an existing file name or specify a new file name. PUTFILE() returns the file name with its path. If a file name isn't entered, the default file name (specified by <expC2>) and extension (specified by <expC3>) are returned. If the **Cancel** push button is chosen or Escape pressed, PUTFILE() returns the null string. You can use the file name PUTFILE() returns to name a file and save it to disk.

Parameters **<expC1>**

<expC1> is the prompt text displayed at the top of the Save As dialog.

<expC2>

<expC2> is the default file name displayed in the text box.

<expC3>

<expC3> specifies file name extensions. Only file names with the specified extension are displayed in the Save As dialog's scrollable list when the **All Files** check box isn't checked. For a complete list of the file extensions used by FoxPro, see the file extension table in the FoxPro for Windows *Developer's Guide* or in the FoxPro for MS-DOS *Update* manual. The first extension in <expC3> is automatically appended to the file name entered if an extension isn't included with the file name.

The character expression <expC3> can take one of the following forms:

- <expC3> can contain a single extension, for example ".PRG", and only file names with that extension are displayed.
- <expC3> can contain a list of file name extensions separated by semicolons. For example, including "PRG;FXP" displays all file names with the extensions .PRG and .FXP.

If file names have the same root name but different extensions (for example, CUSTOMER.PRG and CUSTOMER.FXP), only the file name with the first extension in <expC3> is displayed.

- <expC3> can contain a list of file name extensions separated by vertical bars (for example, "PRG|FXP"). In this case, all file names with listed extensions are displayed, even if the files have the same name root.
- If <expC3> contains just a semicolon (";"), all file names without an extension are displayed.
- If <expC3> is the null string, the names of all files in the current directory are displayed.
- If <expC3> contains MS-DOS wildcard characters (? and *), all file names with extensions that meet the wildcard criteria are displayed. For example, if <expC3> is "?X?", all file names with the extensions .FXP, .EXE, .TXT and so on are displayed.

Example

This example creates a delimited data file from any existing table/.DBF the user chooses. GETFILE() is used to find and open a table/.DBF and PUTFILE() is used to return the name of the target file.

```
dbf_file = GETFILE('DBF', 'Open Table/.DBF:')
USE (dbf_file)

file_name = ALIAS( ) + '.DLM'
out_file = PUTFILE('Delimited file:', file_name, 'DLM')

IF EMPTY(out_file)  && Escape pressed
    CANCEL
ENDIF

COPY TO (out_file) DELIMITED  && Create delimited file
MODIFY FILE (out_file) NOEDIT
```

PV()

Purpose	Returns the present value of an investment
Syntax	PV(<expN1>, <expN2>, <expN3>)
Parameters	<expN1> The periodic payment <expN2> The periodic interest rate <expN3> The total number of payments
Returns	Numeric
See Also	CALCULATE, FV()

Description The financial function PV() returns the present value of an investment. The present value is computed as the value of an investment based on a series of equal periodic payments at a constant periodic interest rate.

Parameters **<expN1>**
The periodic payment is specified by <expN1>. It can evaluate to a positive or negative number. PV() assumes that the payments are made at the end of each period.

<expN2>
The periodic interest rate is specified by <expN2>. If the interest rate of an investment is annual and the payments are made monthly, divide the annual interest rate by 12.

<expN3>
The total number of payments is specified by <expN3>.

Example

```

STORE 500 TO payment      && Periodic payments made monthly
STORE .075/12 TO interest  && 7.5% annual interest rate
STORE 48 TO periodss      && Four years (48 months)
? PV(payment, interest, periods)
20679.19
  
```

QUIT

Purpose	Exits FoxPro
Syntax	QUIT
See Also	CANCEL, RESUME, RUN, SUSPEND

Description Use QUIT to terminate the current FoxPro session and return control to the operating system. In FoxPro for MS-DOS you are returned to the DOS prompt; in FoxPro for Windows you are returned to Windows.

You can also exit FoxPro by choosing **Quit** from the **File** menu.



QUIT should always be used to terminate a FoxPro session. Turning the computer off without issuing QUIT may result in damage to open files and the loss of data. Exiting FoxPro without issuing QUIT may also leave temporary work files on disk that would normally be deleted.

RAND()

Purpose	Returns a random number between 0 and 1
Syntax	RAND([<expN>])
Returns	Numeric

Description RAND() returns a random number between 0 and 1.

Parameter <expN>

<expN> lets you to specify the seed value to use with the function. Using the same positive value for <expN> in the first RAND() function you issue, and no <expN> in subsequent RAND() functions, always returns the same sequence of random numbers.

If <expN> is negative in the first RAND() function you issue, a seed value is created from the system clock. To achieve the most random series of numbers, use RAND() initially with a negative argument and subsequent RAND() functions without an argument.

A seed value of 100001 is automatically used if <expN> is omitted in the first RAND() function you issue. You can restore the default at any time with RAND(100001).

Examples

```
*** Seed random number generator (do this only once per program)
```

```
= RAND(-1)
```

```
*** Print uniformly distributed random number between 0 and 1
```

```
? RAND( )
```

```
*** Returns a uniformly distributed real number between a and b
```

```
FUNCTION urand
```

```
PARAMETER a, b
```

```
RETURN (b-a)*rand( )+a
```

```
*** Returns a uniformly distributed integer between i and j
```

```
FUNCTION irand
```

```
PARAMETER i, j
```

```
RETURN int((j-i+1)*rand( )+i)
```

```

*** Returns a value which simulates throwing a pair of dice.
*** (Uses 'FUNCTION irand' above.)
*** Note: This is NOT the same as 'irand(12)' which has a quite
*** different probability distribution.

```

```

FUNCTION dice
RETURN irand(1,6)+irand(1,6)

```

```

*** Returns a random n-digit number (Uses 'FUNCTION irand' above)

```

```

FUNCTION digits
PARAMETER n
x = 10^n
y = irand(0, x-1) + x
RETURN substr(str(y,n+1),2)

```

```

*** Function to return exponentially distributed values of mean 'a'

```

```

FUNCTION edist
PARAMETER a
RETURN -a*log(rand( ))

```

```

*** Function to return a normally distributed value with mean 'm'
*** and standard deviation 'std'
*** This function requires that the subroutine 'x_p' also be present.
*** The algorithm was taken from the "Handbook of Mathematical
*** Functions," National Bureau of Standards, June 1964.
*** Equation 26.2.23, page 933. Its error is < (4.5E-4 * std)

```

```

FUNCTION ndist
PARAMETER m, std
x = x_p(RAND( ))
RETURN x * std + m

```

```

FUNCTION x_p
PARAMETER p
IF p < 0 or p > 1
    RETURN 0
ENDIF
IF p < .5
    adjust = .T.
ELSE
    p = 1 - p
    adjust = .F.
ENDIF

```

```

t = SQRT(LOG(1/(p*p)))
x = t -((t * 0.010328 + 0.802853) ;
    * t + 2.515517) ;
    / (((t * 0.001308 + 0.189269) ;
    * t + 1.432788) * t + 1)
RETURN IIF(adjust, -x, x)

```

```

*** Returns area under normal curve with mean 'm' and standard
*** deviation 'std' from -infinity to x
*** The algorithm was taken from the "Handbook of Mathematical
*** Functions," National Bureau of Standards, June 1964.
*** Equation 26.2.17, page 932. Its error is < (7.5E-8 * std)
*** This doesn't use the RAND( ) function; it's just included
*** here for your convenience. It requires the function 'p'
*** to also be present

```

```

FUNCTION normal
PARAMETERS x, m, std
RETURN p((x-m)/std)

```

```

FUNCTION p
PARAMETER x
IF x < 0
    adjust = .T.
    x = -x
ELSE
    adjust = .F.
ENDIF
t = 1 / (1 + 0.2316419 * x)
poly = (((1.330274429 ;
    * t - 1.821255978) ;
    * t + 1.781477937) ;
    * t - 0.356563782) ;
    * t + 0.319381530) * t
res = 1 - 0.398942280401432678 ;
    * EXP(-x*x/2) * poly
RETURN IIF(adjust, 1-res, res)

```

RAT()

Purpose	Returns the beginning numeric position of the first occurrence of one character expression or memo field within another character expression or memo field, counting from the rightmost character
Syntax	RAT(<expC1>, <expC2> [, <expN>])
Parameters	<expC1> <expC2> <expN> The expression to search for The expression to be searched The <expN>th occurrence of <expC1> in <expC2>
Returns	Numeric
See Also	AT(), LEFT(), OCCURS(), RATLINE(), RIGHT(), SUBSTR()

Description RAT(), the reverse AT() function, searches a character expression, starting with the last character in the expression, for the occurrence of another character expression. It then returns an integer corresponding to the position where the first character in the character expression is found. 0 is returned if the character expression isn't found.

The search performed by RAT() is case sensitive; upper-case and lower-case are respected.

Parameters <expC1>, <expC2>

RAT() looks for <expC1> in <expC2>. <expC1> and <expC2> can be memo fields.

<expN>

Include <expN> to specify which occurrence of <expC1> in <expC2> to look for. By default, RAT() searches for the first occurrence of <expC1> (<expN> = 1). If <expN> is included, RAT() searches for the <expN>th occurrence of <expC1> in <expC2>. 0 is returned if <expN> is higher than the number of times <expC1> occurs in <expC2>.

Examples

```
STORE 'abracadabra' TO string
STORE 'a' TO find_str
? RAT(find_str,string)
11
? RAT(find_str,string,3)
6
```

RATLINE()

Purpose	Returns the line number of the last occurrence of a character expression within another character expression or memo field, counting from the last line
Syntax	RATLINE(<expC1>, <expC2>)
Parameters	<expC1> The character expression to search for <expC2> The character expression to be searched
Returns	Numeric
See Also	AT(), ATLINE(), LEFT(), MLINE(), OCCURS(), RAT(), RIGHT(), SUBSTR()

Description RATLINE(), the reverse ATLINE() function, searches a character expression, starting with the last character in the expression, for the occurrence of another character expression. If the search is successful, the line number where the match occurs is returned. 0 is returned if the search is unsuccessful.

The search performed by RATLINE() is case sensitive.



The line number returned by RATLINE() is affected by the value assigned to the memo field width by SET MEMOWIDTH. The memo field width determines the value returned by RATLINE() for *character expressions* (character string literals, memory variables or array elements and fields) as well as memo fields.

Parameters <expC1>, <expC2>

RATLINE() searches <expC2> for <expC1>. The search begins at the last character of <expC2>. RATLINE() returns, as an integer, the line number where <expC1> is found. <expC1> and <expC2> can be memo fields. The memo field line containing the matching character expression can be returned as a character string with MLINE().

Example In this example, RATLINE() returns the line number for the last line in the NOTES memo field containing the word “salesperson”. MLINE() uses this value to return the contents of the line.

```
CLOSE ALL
USE salesman
STORE 'salesperson' TO string
STORE MLINE(notes, RATLINE(string, notes)) TO mfileline
? mfileline
```

RDLEVEL()

Purpose	Returns the current READ level
Syntax	RDLEVEL()
Returns	Numeric
See Also	@ ... GET, CLEAR READ, _CUROBJ, OBJNUM()

Description Nested READs are created by issuing @ ... GETs and a READ in a routine that's executed during an active READ. READs can be nested to five levels.

Use RDLEVEL() to return the level of the current READ command. RDLEVEL() returns a value of 0, 1, 2, 3, 4 or 5 depending how deep the current READ is nested. RDLEVEL() returns 0 if no READ is executing.

Example In the following example, pressing F2 at any time displays the current READ level using RDLEVEL(). You can advance to the next READ level by pressing Enter or you can move back a READ level by pressing Escape. Note the use of the VALID clause with @ ... GET and a UDF to create nested READs.

```
ON KEY LABEL F2 DO saylevel
CLEAR
DO saylevel
@ 4,2 GET level1 VALID(proc1( )) DEFAULT 1
READ      && READ Level 1
ON KEY LABEL F2
RETURN

PROCEDURE proc1
@ 6,2 GET level2 VALID(proc2( )) DEFAULT 2
READ      && READ Level 2
RETURN ''
```

RDLEVEL()

```
PROCEDURE proc2
@ 8,2 GET level3 VALID(proc3( )) DEFAULT 3
READ      && READ Level 3
RETURN ''
```

```
PROCEDURE proc3
@ 10,2 GET level4 DEFAULT 4
READ      && READ Level 4
RETURN ''
```

```
PROCEDURE saylevel
@ 2,2 SAY 'Read level: ' + STR(RDLEVEL( ))
RETURN ''
```


READ

Purpose	Activates objects created with @ ... GET and @ ... EDIT commands
Syntax	READ [CYCLE] [ACTIVATE <expL1>] [DEACTIVATE <expL2>] [MODAL] [WITH window title list] [SHOW <expL3>] [VALID <expL4 expN1>] [WHEN <expL5>] [OBJECT <expN2>] [TIMEOUT <expN3>] [SAVE] [NOMOUSE] [LOCK NOLOCK] [COLOR <color pair list> COLOR SCHEME <expN4>]
See Also	@ ... GET, CLEAR READ, CLEAR GETS, RDLEVEL(), READKEY(), SET CONFIRM, VARREAD(), User-Defined Functions (UDFs), SYS(2016)

Description READ activates objects. Objects are GET fields, check boxes, lists, popups, push buttons, radio buttons, spinners and text editing regions defined with @ ... GETs. READ activates all objects created after the last READ or CLEAR GETS was issued.

Exiting the READ

A user can exit a READ in several ways. Moving forward past the last object or backward past the first object exits READ (if the CYCLE clause isn't included with READ). Pressing Escape, Ctrl+W or choosing a control that was defined to terminate the READ also exits the READ.

Multi-Window READ

A READ can span multiple windows. You can place objects can be placed in different windows and activate all of them with a single READ. When you move the cursor from object to object, the objects are traversed in the order they are created, regardless of the window they are placed in.

When you move the cursor to an object, the window that contains the object is activated and becomes the current output window when the current object is in the window. When you press Tab, Enter or Down Arrow when the cursor is positioned on the last object in a window, the cursor is positioned on the first object in the next window. When you press Shift+Tab or Up Arrow when the cursor is positioned on the first object in a window, the cursor is positioned on the last object in the previous window. SHOW GETS contains an example of multi-window reads.

Nested READS

Nested READs are created by defining objects and issuing READ in a routine called while a READ is active. READ commands can be nested to a depth of five levels. RDLEVEL() returns the level of the current READ. The RDLEVEL() topic in this chapter contains an example with nested READ commands.

Event Execution Order

The following list shows the order in which READ events and clauses are executed when the READ is first issued:

1. READ level WHEN clause is executed
2. First GET window is activated
3. READ level ACTIVATE clause is executed
4. READ level SHOW clause is executed
5. GET level WHEN clause is executed for first GET

The following list shows the order in which READ clauses are called when a new window is activated:

1. VALID clause for the field being exited is executed
2. Window of field being exited is deactivated
3. Window of new GET field is activated
4. READ level DEACTIVATE clause is executed
5. READ level ACTIVATE clause is executed
6. WHEN clause for the new field is executed

Clauses**CYCLE**

If the CYCLE clause is included, the READ isn't terminated when you move forward past the last object or backward past the first object. If the cursor is positioned on the last object and you press Tab, Enter or the Down Arrow, the cursor is repositioned on the first object. If you are positioned on the first object and press Shift+Tab or the Up Arrow, you will be repositioned on the last object. If CYCLE is included, pressing a terminating button, Escape, Ctrl+W, or issuing CLEAR READ or a TIMEOUT clause is required to terminate the READ.

ACTIVATE <expl1>

The ACTIVATE clause is executed when READ is issued and whenever the current READ window changes. ACTIVATE can be viewed as a window level WHEN clause.

The logical expression <expl1> is typically a user-defined function. WOUTPUT() can be used in the user-defined function to determine which window is activated. The user-defined function can disable @ ... GET objects in other windows, hide windows, display a message and so on.

DEACTIVATE <expl2>

The DEACTIVATE clause is executed whenever you attempt to bring any other window forward (that is, whenever WONTOP() changes). DEACTIVATE can be viewed as a window level VALID clause.

The logical expression <expl2> is typically a user-defined function. The user-defined function can be used to validate the contents of fields in a window before allowing you to bring forward another window. The READ isn't terminated if the user-defined function returns false (.F.). The READ is terminated if the user-defined function returns true (.T.).

MODAL

Including the MODAL key word in READ prevents *all* windows from being activated with the exception of windows that are involved in the READ. If windows that are not involved in the READ are defined when a MODAL READ is issued, they cannot be activated or accessed.

WITH <window title list>

By default, all interactive windows (Browse windows, desk accessories and windows opened with MODIFY FILE, MODIFY REPORT, and so on) can participate in a READ. Include the WITH clause to restrict the windows that participate in the READ. The windows included in the WITH clause can include FoxPro system windows (Browse windows, desk accessories and text and memo editing windows).

Including the WITH clause automatically creates a MODAL READ.

- Include the Browse window title (by default, the table/.DBF alias) in the window list to make a Browse window accessible.
- Include the table/.DBF alias in the window list to make a memo window accessible.

If a window title contains a character that isn't alphabetic, numeric or an underscore, include the part of the title up to the first character that isn't alphabetic, numeric or an underscore in the window list. For example, if the window title is Client List, include Client in the window list (the space between Client and List isn't alphabetic, numeric or an underscore).

If a window title contains a character that isn't alphabetic, numeric or an underscore, you can place the title in quotation marks in the window list. For example, if the window title is Client List, you can include "Client List" in the window list.

SHOW <expl3>

The SHOW clause is executed whenever SHOW GETS is issued. The value returned by a SHOW routine is ignored. A SHOW routine can be used to refresh @ ... SAYs or to enable or disable objects.

VALID <expl4> | <expN1>

The VALID clause is evaluated when you attempt to exit the current READ or when a READ issued without a previous @ ... GET clause times out. The VALID clause can specify a logical or numeric expression or a user-defined function (UDF) that returns a logical or numeric value.

The READ is terminated if <expl4> evaluates to true (.T.). If VALID <expl4> evaluates to a logical false (.F.), the current object remains active if possible.

If the object cannot remain active (perhaps the VALID routine deactivated the object), you are moved to the first object.

If the VALID clause returns a number, you are moved to the corresponding object. The READ is terminated if an object corresponding to the number doesn't exist. A VALID clause returning a value that isn't of logical or numeric type has the same effect as one returning true (.T.).

WHEN <expL5>

The WHEN clause determines if the READ is executed. <expL5> must evaluate to true (.T.) when the READ is issued for the READ to be executed. The READ is ignored and program execution continues with the command immediately following the READ if <expL5> evaluates to false (.F.).

OBJECT <expN2>

Include the OBJECT clause to specify which object is initially selected when READ is issued. <expN2> determines which object is initially selected. Object numbers are determined by the order in which the objects are created.

Each individual push, radio and invisible button is considered one object. In the following program example, a field and three radio buttons are created. The middle radio button is initially selected by including the OBJECT 3 clause with the READ. The NAME field is object number 1, the first radio button is object number 2, the second radio button is object number 3 and the last radio button is object number 4.

Example

```
STORE 1 TO radio
STORE SPACE(10) TO name
CLEAR
@ 2,2 SAY 'Enter a name: ' GET name
@ 4,2 GET radio PICTURE '@*R Apples;Oranges;Lemons'
READ CYCLE OBJECT 3
```

TIMEOUT <expN3>

A TIMEOUT clause determines how long the READ is in effect. <expN3> specifies the number of seconds that can elapse without user input before the READ is terminated. If the READ is terminated by a TIMEOUT clause, READKEY() returns 20 if changes have not been made to any object. If changes have been made, READKEY() returns 276.

When READ is terminated by a TIMEOUT clause, any changes made to the field being edited when the READ ends are discarded. However, changes made to other fields are saved.

SAVE

All objects definitions are automatically cleared when the READ ends unless the SAVE key word is included. You can reissue READ without recreating the objects if you include SAVE with the previous READ.

NOMOUSE

Include the NOMOUSE key word to prevent objects from being selected with the mouse. You must use the keyboard to move from object to object. You can still use the mouse within fields to cut, copy, paste and position the cursor.

LOCK | NOLOCK

The LOCK and NOLOCK key words let you specify if a record containing fields specified in objects is automatically locked during a READ when the table/.DBF is opened for shared use on a network.

If LOCK is included, record locks are attempted for every record used by the objects. If the locks are successfully placed, you can edit the records. SET REPROCESS determines how many times or for how long locking attempts are made. READ defaults to READ LOCK if LOCK or NOLOCK isn't included.

If NOLOCK is included, the records used by objects are not locked. Instead, all objects that use fields in the records are made read-only and the control cannot be selected and is displayed in the disabled colors.

If you issue READ NOLOCK and modify a memo field, the record is automatically locked.

COLOR <color pair list>
COLOR SCHEME <expN4>

The color of the current @ ... GET editing region can be specified by including the number of an existing color scheme in the COLOR SCHEME clause or a set of color pairs in the COLOR clause.

When an object is selected, it assumes the colors of the second color pair in the color pair list. All other colors in the color pair list are ignored. If you omit the first color pair you must include a comma for the color pair.

If you include a COLOR SCHEME clause, color pair 2 of the specified color scheme determines the color of the current @ ... GET editing region. By default, the color of the current @ ... GET editing region is determined by color pair 2 of color scheme 1, color pair 2 of the window the @ ... GET editing region is placed in or color pair 2 of the color clause specified in @ ... GET. The color of other objects (check boxes, popups, push buttons, etc.) isn't affected by this color clause.

A color scheme is a set of 10 predefined color pairs. The color pairs in a color scheme can be changed with SET COLOR OF SCHEME. In FoxPro for MS-DOS the color pairs in a color scheme can also be changed in the Color Picker.

A color pair is a set of two letters separated by a forward slash. The first color letter specifies the foreground color and the second letter specifies the background color.

For example, this color pair specifies a red foreground on a white background:

R/W

For a list of colors and their corresponding color letters, see the SET COLOR commands section in this chapter or the Color Table by Color Pair topic in the online help facility.

In FoxPro for Windows, a color pair can also be specified with a set of 6 RGB (Red Green Blue) color values separated by commas. The first 3 color values specify the foreground color and the second 3 color values specify the background color. The color values can range from 0 through 255.

In FoxPro for Windows, the R/W color pair in the example above can also be specified with this RGB color pair:

RGB(255,0,0,255,255,255)

Foundation READ

FoxPro lets you easily create event-driven interfaces (like the FoxPro interface) for your applications. This section suggests a method for creating an event-driven interface in FoxPro.

In previous versions of FoxPro, some sophisticated users utilized an event loop to handle menu choices and window selections when creating an event-driven interface. The event loop would constantly poll the system status and then take action based on which window was on top or which menu option the user chose.

DO WHILE is used to create the event loop (often a large loop). Event loops should be avoided for the following reasons:

- Event loops are constantly being executed, making them intensive CPU users.
- Event loops are complex and difficult to program.
- Event loops are unfriendly in multi-program environments like DESQview, Windows and the Macintosh MultiFinder.

A Single READ with Multiple Windows

Event loop programming is rarely required in FoxPro. A single READ supporting multiple windows can be used to eliminate most event loops. A single READ with multiple windows provides versatility and flexibility without complex programming.

We suggest that you encompass all the windows that you intend to interact with. FoxPro can coordinate windows containing objects (@ ... GETs, check boxes, radio buttons, popups, and so on) with other windows not usually associated with a READ. By default, all interactive windows (Browse windows, desk accessories, memo editing windows and windows opened with MODIFY FILE, MODIFY REPORT) can participate in a READ.

Coordinating Windows with a Single READ

To programmatically coordinate windows containing @ ... GET objects with a single READ:

1. Create the windows that will contain the @ ... GET objects.
2. Activate one of the windows and issue the @ ... GETs that create the objects for the window.
3. Activate another window and issue the @ ... GETs that create the objects for that window.
4. Continue in this fashion for the remaining windows.
5. Issue READ.

Screen set program code generated by GENSCRN provides a good example of this sequence.

The windows created and activated are now coordinated with a single READ. Each of the GET objects appears in its respective window, and the user can move between the @ ... GET objects in each of the windows. When the user moves from object to object with the keyboard, the objects are traversed in the order the @ ... GETs have been issued, regardless of which window they are in. A window is activated and becomes the current output window when the current object is in the window.

When the user presses Tab, Enter or Down Arrow when the cursor is positioned on the last object in a window, the cursor is positioned on the first @ ... GET object in the next window. When the user presses Shift+Tab or Up Arrow on the first object in a window, the cursor is positioned on the last @ ... GET object in the previous window. The user can also move between different @ ... GET objects in different windows by clicking on the objects.

Access to other windows during a single READ can be controlled with the READ MODAL clause. Including MODAL in READ prevents interaction with windows that are not encompassed in the READ's coordinated set of windows. When MODAL is included with READ, the user cannot close, zoom, minimize, or move windows not involved in the READ. Clicking on a window not involved in the READ sounds a beep if BELL is SET ON.

By default, all interactive windows (Browse windows, desk accessories and windows opened with MODIFY FILE, MODIFY REPORT, and so on) can participate in a READ.

You can explicitly indicate the windows that participate in the READ by specifying their names in the WITH clause. The READ command is automatically made MODAL if you include the WITH clause to specify other windows involved in the READ. The windows included in the WITH clause can include FoxPro system windows (Browse windows, desk accessories and text and memo editing windows).

ACTIVATE and DEACTIVATE

When a window involved in the READ is brought forward, the action taken can be controlled by the ACTIVATE and DEACTIVATE clauses. The user can bring a window forward by clicking on the window, by cycling to the window with Ctrl+F1 or by choosing a control or menu option that activates the window.

When a window containing objects is foremost and you attempt to bring another window forward, the DEACTIVATE clause is executed after the new window is brought forward.

The value returned by the `DEACTIVATE` clause determines whether the `READ` is terminated. If the `DEACTIVATE` clause returns true (.T.), the `READ` is terminated. If the clause returns false (.F.), the `READ` isn't terminated.

When the `DEACTIVATE` clause is executed, `WONTOP( )` returns the name of the new window brought forward and `WLAST( )` returns the name of the window that was formerly frontmost. The window functions (`WONTOP( )`, `WOUTPUT( ) ...`) return false (.F.) when you include the name of a window that doesn't exist.

If your `DEACTIVATE` clause lets the new window remain forward and the new window contains objects, the `ACTIVATE` clause is executed.

When a window is brought forward, you can return the window to its former position by issuing the following command in the `DEACTIVATE` clause:

```
ACTIVATE WINDOW (WLAST( ))
```

The function `WREAD( )` facilitates manipulation of windows participating in a `READ`. `WREAD( )` lets you determine if a specific window is participating in the current `READ`.

Coordinating Multiple READs / Foundation READs

The single `READ` with multiple windows section above recommends an approach to handling multiple windows with a single `READ` command. The single `READ` approach works for most applications.

However, a foundation `READ` may be necessary in some applications. A foundation `READ` is a `READ` without objects. A foundation `READ` is used to coordinate multiple `READs` into a single interactive session.

A foundation `READ` typically incorporates code in its `VALID` clause routine that activates and controls the child `READs`. Child `READs` are sometimes called nested `READs`.



A foundation READ normally has a VALID clause. The VALID clause is executed when an event occurs that would otherwise terminate the READ. Here are the events that terminate a foundation READ:

- Any mouse click or keystroke that isn't a menu selection or doesn't execute an ON KEY LABEL. During a foundation READ, the FoxPro system menu is accessible.

Selecting an item from a menu or executing an ON KEY LABEL command doesn't terminate a foundation READ. If a menu is activated but a selection isn't made from the menu, the foundation READ isn't terminated.

- A child READ is terminated. However, the foundation READ isn't terminated if a grandchild READ is terminated.

The value returned by the READ VALID clause determines whether or not the foundation READ is terminated. If the READ VALID clause returns true (.T.), the foundation READ is terminated; if it returns false (.F.), the foundation READ command remains active.

READKEY() facilitates the management of child READ commands. By specifying a numeric argument in READKEY(), you can determine how the last READ was terminated. For example, the value can indicate that the last READ was terminated by closing a window, by a CLEAR READ, by a READ VALID or DEACTIVATE clause returning a logical true (.T.).

Here are the values returned by READKEY(<expN>) with the corresponding reason the READ was terminated:

Value	Reason
1	None of the following
2	CLEAR READ issued
3	Terminating control chosen
4	READ window closed
5	DEACTIVATE clause returned .T.
6	READ timed out

Sample Applications

Included with FoxPro are three sample applications, INV.APP, EX1.APP and EX2.APP. These sample applications illustrate the features available with READs and how these features can be used to create an event-driven interface.

The applications are built from three projects, INV.PJX, EX1.PJX and EX2.PJX. The projects contain all the files (programs, screen sets, menus, tables/.DBFs, and so on) used by the applications. To examine the files included in a project, open the project with MODIFY PROJECT, or through the Open dialog.

INV: A Single READ with Multiple Windows

The sample application INV.APP demonstrates the use of a single READ with multiple windows. A user-defined window containing @ ... GET objects, two Browse windows and the CONTROL2 screen set control panel are coordinated with a single READ command.

EX1: A Foundation Read with Multiple Applications

EX1.APP, a more complex application than INV.APP, demonstrates coordination of four READs, each of which use the control panel built from the CONTROL3.SCX screen set. A foundation READ coordinates the four READs. EX1.APP has been designed so you can utilize any FoxPro desk accessories or windows (Help, Filer, Calculator, Calendar/Diary, editing windows, and so on) in conjunction with the application windows.

Applications are launched from the **Application** menu. You can switch from one application to another by clicking in the respective application window. Execution of the entire system can be terminated by choosing the **Quit** option in the **File** menu.

EX1 Quiz

Before you attempt to write your own multi-READ event driven code, we recommend you carefully study EX1.APP and be certain you can answer the following questions about its operation. The answers are provided at the end of this section.

1. What would happen if the menu handling routine always immediately launched the requested application, regardless of the current READ level (returned by RDLEVEL())?
2. Why is it necessary to specify the IN EX1.MPR clause in the menu command on the **Application** menu?

3. Where is the code that handles the situation when an application window is closed manually, either through the window's **Control** menu (FoxPro for Windows) or by clicking on its close box (FoxPro for MS-DOS), or by choosing **Close** from the **File** menu?
4. What is the largest number of READs that are active at any one time while EX1.APP is executing?
5. What insures that the control panel vanishes when the last remaining application window is closed?
6. If you bring, say, the Calculator on top of an application window, is the application READ still active? The foundation READ?
7. When you are executing EX1.APP and neither an application window nor the control panel is visible, and you're using, say, the Filer, is any READ active?
8. When you're running in one application window and you click in another application window, how is control transferred from the first application to the new one? What event triggers the transfer?

If you can answer all these questions, you're ready to proceed to the next example.

EX2: A Foundation Read with Multiple Applications

This application is similar to EX1.APP, but replaces the single invoice editing screen with the INV.APP application that, besides editing the invoice file, includes the CONTROL2 control panel and two Browse windows coordinated as part of the READ.

Study EX2.APP carefully and you'll see that the changes required to handle this more complex situation are rather simple. If you can answer the eight questions about EX1.APP, you should have no difficulty with EX2.APP. And by now, you should have a general idea how to proceed in your own applications.

EX1 Quiz Answers

1. You would add one READ level each time you selected an application from the menu and, therefore, would get “Too many READs in effect” in short order. FoxPro supports a maximum of five READ levels.
2. Because the procedure MENUHIT is contained within EX1.MPR, EX1.MPR isn’t a part of EX1.PRG; the IN EX1.MPR clause must be included to tell EX1.PRG where to look for the MENUHIT procedure.
3. It’s found in the STOPREAD procedure which is evaluated each time an application window is deactivated. The window name is passed to STOPREAD. If the window isn’t visible, (because it was closed either manually or by clicking on its close box) the control panel is displayed. STOPREAD returns .T. if the window is involved in the READ, otherwise .F. is returned (a FoxPro system window was closed).
4. There are at most three READs active: the foundation READ, one of the application READs, and a READ MODAL associated with either BROWSER.SPR or FINDER.SPR.
5. The procedure named EFFACE, which is called in the cleanup code for each application screen, ensures that when the last application window is closed the control panel is released.

EFFACE is called when either the **Quit** button is pressed or a READ window has been manually closed.

EFFACE looks through all the windows that are open, from back to front. If it finds any of the user application windows (which require the control panel) it simply exits. Otherwise, it concludes you’re finished with the control panel and releases it.

6. The application READ is terminated because its DEACTIVATE clause returned .T., but the foundation READ remains active.
7. The first application’s DEACTIVATE clause returns .T., thereby terminating that application’s READ. Termination of the application’s READ triggers the foundation READ’s VALID clause routine.
8. This routine, called MYHANDLER, uses WONTOP() to return the name of the application window where the mouse was clicked, and the screen program (.SPR) for that window is executed.

Examples For examples using READs, see the sections on @ ... GET, CLEAR READ and SHOW GETS.

READ MENU

Purpose	Activates a popup
Syntax	READ MENU TO <memvar> [SAVE]
Remarks	Included for backward compatibility – use the FoxPro Menu Builder to create menus
See Also	@ ... MENU, CREATE MENU

Description This command activates the menu popup created with @ ... MENU.

The value initially contained in the memory variable or array element <memvar> specifies which popup option is selected when the popup is activated. When you choose a popup option, a value is stored to <memvar> corresponding to option you choose. If you press Escape to exit the popup, <memvar> contains 0.

Clause **SAVE**

The SAVE clause keeps an image of the menu popup on the desktop (FoxPro for MS-DOS), the main FoxPro window (FoxPro for Windows) or in a user-defined window after a choice is made from the menu or Escape is pressed. Normally, the menu is cleared from the desktop, main FoxPro window or a user-defined window after you make a choice.

READKEY()

Purpose	Returns a value corresponding to the key pressed to exit certain editing commands, or a value indicating how the last READ was terminated
Syntax	READKEY ([<expN>])
Returns	Numeric
See Also	INKEY(), ON KEY, READ, VARREAD()

Description READKEY() returns a value that represents the key pressed to exit from certain editing commands or it returns a value that indicates how the last READ was terminated.

If READKEY() is issued without the optional numeric expression <expN>, the value returned represents the key pressed to exit from these editing commands: APPEND, BROWSE, CHANGE, CREATE, EDIT, INSERT, MODIFY and READ.

An integer between 0 and 36, or between 256 and 292 is returned. The value returned is between 0 and 36 if the data was not modified. The value returned is between 256 and 292 if the data is modified.

Values Returned by READKEY					
Key(s)	No Update Code	Update Code	Key(s)	No Update Code	Update Code
Backspace Left Arrow Ctrl+H Ctrl+S	0	256	F1	36	292
Right Arrow Ctrl+D* Ctrl+L	1	257	Ctrl+Q Escape	12	268
Home Ctrl+A*	2	258	Ctrl+End Ctrl+W	—	270
End Ctrl+F	3	259	Enter Ctrl+Right Arrow Ctrl+M	15	271
Up Arrow Ctrl+E* Shift+Tab Ctrl+K*	4	260	Timeout	20	276
Down Arrow Ctrl+X* Tab Ctrl+Enter Ctrl+I Ctrl+J	5	261	Ctrl+Home Ctrl+]	33	289
PgUp Ctrl+R*	6	262	Ctrl+PgUp	34	290
PgDn Ctrl+C	7	263	Ctrl+PgDn Ctrl+^	35	291
Ctrl+ Left Arrow	8	264			

* If SET SYSMENU is OFF or FOXPLUS.FKY has been loaded, these values are returned. FOXPLUS.FKY can be found in the GOODIES directory.

Parameter <expN>

READKEY() returns a value that indicates how the last READ was terminated if the optional numeric expression <expN> is included. The numeric expression <expN> can be any value.

The following table lists the values returned by READKEY() with the reason the READ was terminated.

Value	Reason
1	None of the following
2	CLEAR READ issued
3	Terminating control chosen
4	READ window closed
5	DEACTIVATE clause returned .T.
6	READ timed out

Example

In the following example, the data from the current record in CUSTOMER.DBF is placed in an array with SCATTER. The field data for the current record is displayed for editing. Press Escape to cancel your changes. READKEY() tests to see if the READ was exited by pressing Escape. If Escape is pressed the original values are restored to the fields with GATHER.

```

SET TALK OFF
CLOSE ALL
CLEAR
USE customer
COPY TO temp.dbf
USE temp
SCATTER TO temp
DEFINE WINDOW menter FROM 6,10 to 18,70 PANEL
ACTIVATE WINDOW menter
@ 1,3 SAY 'Customer: ' GET company
@ 3,3 SAY 'Address: ' GET address
@ 5,3 SAY 'City: ' GET city
@ 9,18 SAY 'Press Escape to cancel'
READ
IF READKEY( ) = 268  && Escape was pressed
    GATHER FROM temp
ENDIF
DEACTIVATE WINDOW menter
RELEASE WINDOW menter
DISPLAY NEXT 5 FIELDS company, city
USE
ERASE temp.dbf

```

RECALL

Purpose	Unmarks records marked for deletion
Syntax	RECALL [<scope>] [FOR <expL1>] [WHILE <expL2>] [NOOPTIMIZE]
See Also	DELETE, PACK, SET DELETED, ZAP

Description RECALL unmarks records that are marked for deletion in the selected table/.DBF. RECALL can be used to recover records, provided that a PACK or ZAP has not been performed on the table/.DBF.



Once a file has been packed, all records marked for deletion are gone *forever*.

Records can be marked for deletion by issuing DELETE, from within a Browse, Change or Edit window or by choosing **Delete...** from the **Record** menu.

Records can be recalled by issuing RECALL, from within a Browse, Change or Edit window or by choosing **Recall...** from the **Record** menu.

Clauses **<scope>**

The scope clauses are: ALL, NEXT <expN>, RECORD <expN>, and REST. These are explained in the Overview of the FoxPro Language chapter of this manual. Commands which include <scope> operate only on the table/.DBF in the active work area.

You can specify a scope of records to recall. Only the records that fall within the range of records specified by the scope are recalled.

The default scope for RECALL is the current record (NEXT 1).

FOR <expL1>

If the FOR clause is included, only the records for which <expL1> evaluates to true (.T.) are recalled. Including a FOR clause lets you conditionally recall records, filtering out undesired records.

Rushmore optimizes a RECALL FOR if <expL1> is an optimizable expression. For best performance, use an optimizable expression in the FOR clause. A discussion of Rushmore optimizable expressions appears in the Optimizing Your Application chapter in the *FoxPro Developer's Guide*.

WHILE <expL2>

If the WHILE clause is included, records are recalled for as long as <expL2> evaluates to true (.T.).

NOOPTIMIZE

To prevent Rushmore optimization of RECALL, include NOOPTIMIZE. For further information on NOOPTIMIZE, see SET OPTIMIZE in this manual.

RECCOUNT()

Purpose	Returns the number of records in the current or specified table/.DBF
Syntax	RECCOUNT ([<expN> <expC>])
Returns	Numeric
See Also	RECNO(), RECSIZE()

Description RECCOUNT() returns the number of records in a table/.DBF. All records in the table/.DBF are counted — the value returned by RECCOUNT() isn't affected by SET DELETED and SET FILTER.

Parameter <expN> | <expC>

RECCOUNT() with no parameter returns the number of records in the table/.DBF in the selected work area. To return the number of records in a table/.DBF open in another work area, include the work area number <expN> or the table/.DBF alias <expC>.

RECCOUNT() returns zero if a table/.DBF isn't open in the work area you specify.

Example In this example, FoxPro compares the available disk space with the amount needed to sort CUSTOMER.DBF.

```
*** Check DISKSPACE before a SORT ***

USE customer

*** Get size of table/.DBF header ***

dbf_top = HEADER( )

*** Calculate size of table/.DBF ***

file_size = dbf_top + (RECSIZE( ) * RECCOUNT( ) + 1)
IF DISKSPACE( ) > (file_size * 3)
    @ 5,0 SAY 'Sufficient disk space to Sort.'
ELSE
    @ 5,0 SAY 'Insufficient disk space. Sort cannot be done.'
ENDIF
```

RECNO()

Purpose	Returns the current record number
Syntax	RECNO([<expN> <expC>])
Returns	Numeric
See Also	GO, RECCOUNT(), RECSIZE(), SEEK, SKIP

Description RECNO() returns the current record number in the current or specified table/.DBF. The record pointer is positioned on the current record.

RECNO() returns a value that is one greater than the number of records in the file if the record pointer is positioned beyond the last record in the file. RECNO() returns 1 if the record pointer is positioned before the first record in the file.

If the table/.DBF has no records, RECNO() returns 1 and EOF() returns true (.T.).

Parameter <expN> | <expC>

RECNO() returns the current record number for the table/.DBF in the selected work area.

To return the current record number for a table/.DBF open in another work area, include the work area number <expN> or the table/.DBF alias <expC>.

If a table/.DBF isn't open in the work area you specify, RECNO() returns zero.

RECNO(0) Soft Seek

After an unsuccessful SEEK in an indexed table/.DBF, RECNO(0) uses “soft seek” logic to determine the record number to return. RECNO(0) returns the record number of the closest match if the SEEK is unsuccessful and no matching record is found. RECNO(0) returns 0 if a close match cannot be found. The message “Record out of range” is generated if you issue GO RECNO(0) and a close match hasn't been found.

Example The following example searches CUSTOMER.DBF for a name, and, if the name isn't found, uses RECNO(0) to return the closest matching company.

```
CLOSE DATABASES
USE customer
SET ORDER TO company
SEEK 'John Doe'
IF FOUND( )
    DISPLAY
ELSE
    GOTO RECNO(0)
    ? 'Closest matching company is ' + company
ENDIF
```

The result displayed is:

Closest matching company is Jordan Books

RECSIZE()

Purpose	Returns the size of a table/.DBF record
Syntax	RECSIZE ([<expN> <expC>])
Returns	Numeric
See Also	RECCOUNT(), FSIZE()

Description RECSIZE() returns the size (width) of a table/.DBF record.

Parameter <expN> | <expC>

RECSIZE() with no parameter returns the record size for the table/.DBF in the selected work area.

To return the record size for a table/.DBF open in another work area, include the work area number <expN> or the table/.DBF alias <expC>.

RECSIZE() returns zero if a table/.DBF isn't open in the work area you specify.

Example See RECCOUNT() for an example using RECSIZE().

REGIONAL

Purpose	Creates regional memory variables and arrays
Syntax	#REGION <number> REGIONAL <memvar list>
See Also	PRIVATE, PUBLIC, STORE

Description REGIONAL creates regional memory variables and arrays. Memory variables or arrays with identical names can be created without interfering with each other when their values are protected within a “region”. Regional variables are similar to private variables.

Declaring screen variables as regional in the screen setup code ensures that a variable from one screen isn’t affected by a variable with the same name used in another screen. If the variables in each of the screens in the screen set are declared as regional variables in their setup code, FoxPro automatically resolves the conflicts.

The GENSCRN screen generating program used by the Screen Builder automatically inserts the necessary #REGION compiler directives needed to resolve memory variable name conflicts.

Creating Regional Variables

#REGION <number>

Regional variables should be declared before they are used in a program. A region is created with the #REGION compiler directive (#REGION isn’t a command). Include a region number 0 through 31.

REGIONAL <memvar list>

After a region is created with the #REGION directive, declare the variables for that region with REGIONAL. <memvar list> is a list of memory variables and arrays separated by commas.

During program compilation, if another identically-named regional variable has already been compiled when a regional declaration is encountered, the second occurrence of the variable name is made unique to ensure there is no conflict with the previously-declared regional variable.

A variable's name is made unique by padding the regional variable's name out to 10 characters with underscores and the current region number. This substitution takes place entirely during program compilation and has no effect on execution speed.

When a variable name is modified, the modified variable name can be displayed with DISPLAY MEMORY. To monitor the variable in the Debug window, use its modified variable name. Since the Trace window uses the original program source code, the original variable name (not the modified name created by the compiler) appears in the Trace window.

Example

Two sets of regional variables are created. In region 1, the variables A, B, C and D are created, and the character string "One" is stored to each. In region 2, the variables C, D, E and F are created, and the character string "Two" is stored to each. The variables C and D are common to both regions.

The screen output from DISPLAY MEMORY is then shown. The names of memory variables C and D are modified in the second region. C becomes C____2, and D becomes D____2. All the variables are private and can be accessed by lower-level programs.

```
#REGION 1
REGIONAL A,B,C,D
STORE 'One' to A,B,C,D
```

```
#REGION 2
REGIONAL C,D,E,F      && C and D are common to both regions
STORE 'Two' to C,D,E,F
DO showmemory
```

```
PROCEDURE showmemory
DISPLAY MEMORY LIKE *
```

*** Output from DISPLAY MEMORY ***

```
A          Priv  C  "One"
B          Priv  C  "One"
C          Priv  C  "One"
D          Priv  C  "One"
C____2     Priv  C  "Two"
D____2     Priv  C  "Two"
E          Priv  C  "Two"
F          Priv  C  "Two"
```

REINDEX

Purpose	Rebuilds open index files
Syntax	REINDEX [COMPACT]
See Also	INDEX, SET INDEX, SET UNIQUE, SYS(), USE

Description REINDEX updates index files. Index files become outdated when a table/.DBF is opened without opening its corresponding index files and changes are made to the index files' key fields. When index files become outdated, they can be updated by reindexing.

All index files open in the selected work area are updated by REINDEX. All tags in .CDX compound index files are also updated. Structural .CDX compound indexes automatically opened with the table/.DBF are also updated.

Any index files created with the UNIQUE key word of the INDEX command or with UNIQUE SET ON retain their UNIQUE status when they are reindexed.

To REINDEX outdated index files, issue these commands:

```
USE <table> INDEX <outdated index or indexes>
REINDEX
```

COMPACT

FoxPro recognizes each index file type (.CDX, .IDX and structural) and reindexes accordingly. You can convert regular .IDX index files to compact .IDX index files by including COMPACT.

RELATION()

Purpose	Returns relational expressions for tables/.DBFs
Syntax	RELATION(<expN1> [, <expN2> <expC>])
Parameters	<expN1> Specifies which relation is to be returned <expN2> The work area number for a table/.DBF <expC> The table/.DBF alias
Returns	Character
See Also	DISPLAY STATUS, LIST STATUS, SET RELATION, SET RELATION OFF, TARGET()

Description This function returns a specified relational expression for a table/.DBF open in a specific work area. The null string is returned if no relations exist. See SET RELATION for additional information about creating relations between tables/.DBFs.

DISPLAY STATUS and LIST STATUS also display relational expressions.

Parameters **<expN1>**

<expN1> specifies which relation is returned. For example, if <expN1> is 3, RELATION returns the relational expression for the third relation created.

<expN2> | <expC>

RELATION() by default returns relational expressions for the table/.DBF in the selected work area. To return relational expressions for a table/.DBF open in another work area, include the work area number <expN> or the table/.DBF alias <expC>.

If a table/.DBF isn't open in the work area you specify, RELATION() returns the null string.

Example In the following example, tables/.DBFs are opened and orders set, then the relations are displayed using RELATION().

```
CLOSE DATABASES
USE salesman IN 0 ORDER ono
USE customer IN 0 ORDER ono
USE offices IN 0 ORDER ono
SELECT salesman
SET RELATION TO ono INTO customer
SET RELATION TO ono INTO offices ADDITIVE
? RELATION(1)      && ONO
? RELATION(2)      && ONO
? RELATION(3)      && null string
```

RELEASE

Purpose	Releases memory variables, libraries, menus, menu pads, popups, popup options or windows
Syntax	RELEASE <memvar list> RELEASE ALL [LIKE <skel> EXCEPT <skel>] RELEASE LIBRARY <library name> RELEASE MENUS [<menu list> [EXTENDED]] RELEASE PAD <pad name> OF <menu name> RELEASE PAD ALL OF <menu name> RELEASE POPUPS [<popup list> [EXTENDED]] RELEASE BAR <expN> OF <popup name> RELEASE BAR ALL OF <popup name> RELEASE WINDOWS [<window list>]
See Also	CLEAR MEMORY, DEACTIVATE POPUP, DEFINE MENU, DEFINE PAD, DEFINE POPUP, DEFINE BAR, DEFINE WINDOW, SET LIBRARY, SYS(2013)

Description RELEASE removes memory variables and arrays, menus, menu pads, popups, popup options, user-defined windows and system windows from memory. This makes the previously allocated memory available. System memory variables cannot be released from memory.

You can also use RELEASE to remove menu pads from the FoxPro system menu bar and options from the FoxPro system menus.

Commands **RELEASE <memvar list>**

RELEASE <memvar list> releases memory variables and arrays whose names are contained in the <memvar list> from memory. Separate the memory variables and arrays with commas.

RELEASE ALL [LIKE <skel> | EXCEPT <skel>]

Use RELEASE ALL to release all memory variables and arrays, release all memory variables and arrays that match the specified skeleton or release all memory variables and arrays EXCEPT those that match the specified skeleton. The skeleton <skel> can include the wildcard characters ? and *.



RELEASE ALL, RELEASE ALL LIKE or RELEASE ALL EXCEPT do not release public variables when issued in a program. Use RELEASE <memvar list> to release public variables.

RELEASE LIBRARY <library name>

RELEASE LIBRARY removes a single external API routine library (written in C or assembly language) from memory. Issue SET LIBRARY TO to remove all API libraries from memory. An external API routine library is loaded into memory with SET LIBRARY. See SET LIBRARY for additional information on API routine libraries.

RELEASE MENUS [<menu list> [EXTENDED]]

RELEASE MENUS releases specified user-defined menu bars from memory. An active menu must first be deactivated with DEACTIVATE MENU before it can be released from memory. The menus removed from memory are specified in the <menu list>. If RELEASE MENUS is issued without a menu list, all user-defined menus are removed from memory.

Include the EXTENDED key word to release a menu bar and all its subordinate pads, popups, bars and all associated ON SELECTION BAR, ON SELECTION MENU, ON SELECTION PAD, and ON SELECTION POPUPS.

**RELEASE PAD <pad name> OF <menu name>
| RELEASE PAD ALL OF <menu name>**

RELEASE PAD removes a specified pad or all pads from a menu bar. You can selectively remove a pad from a user-defined menu bar or the FoxPro system menu bar by specifying its name in <pad name>. For example, the command RELEASE PAD _MEDIT OF _MSYSMENU removes the **Edit** pad and its menu from the system menu bar.

Every pad in a user-defined menu bar is removed if you specify ALL. ALL cannot be used to remove pads from the system menu bar.

RELEASE POPUPS [<popup list> [EXTENDED]]

RELEASE POPUPS releases specified or all user-defined popups from memory. An active popup must be deactivated with DEACTIVATE POPUP before it can be released from memory.

Include the names of popups in <popup list> to specify the popups to release. If RELEASE POPUPS is issued without a <popup list>, all user-defined popups are removed from memory.

System menu popups that appear under the system menu bar can also be released. To release a system menu popup, include the name of the system menu popup (_MFILE, _MEDIT, _MDATA, and so on). Use SET SYSMENU TO DEFAULT to restore the default system menu bar and system menu popups.

Include EXTENDED to release a popup and its subordinate popups, bars and all associated ON SELECTION POPUP and ON SELECTION BAR commands.

**RELEASE BAR <expN> OF <popup name>
 | RELEASE BAR ALL OF <popup name>**

RELEASE BAR removes a specified option or all options from a popup. You can selectively remove an option from a user-defined popup by specifying its option number <expN>. To remove an option from a system menu (**Edit**, **File**, **Database**, and so on), include the system option name in <expN>.

Every option in a user-defined popup is removed if you include ALL. ALL cannot be used to remove options from a FoxPro system menu popup.

RELEASE WINDOWS [<window list>]

RELEASE WINDOWS removes the windows whose names are contained in the <window list> from memory. <window list> can include both user-defined windows and FoxPro system windows.

The active user-defined window is released if a window list isn't included.

FoxPro desk accessories and system windows can be removed from the desktop, main FoxPro window or a user-defined window with RELEASE WINDOW.

The following is a list of desk accessories and system windows that can be released from the screen or a parent window with RELEASE WINDOW:

Desk Accessories	System Windows
Filer	Command
Calculator	Debug
CALENDAR/Diary	Trace
SPECIAL Characters (FoxPro for MS-DOS)	View
ASCII Chart (FoxPro for MS-DOS)	
Puzzle	

When releasing desk accessories that have two parts to their name (CALENDAR/Diary, SPECIAL Characters and the ASCII Chart), use just the first part of the name (shown here in upper case). For example, use RELEASE WINDOW CALENDAR to release the CALENDAR/Diary. Use ACTIVATE WINDOW to place a desk accessory or a system window on the desktop, main FoxPro window or in a user-defined window.

Example

These commands remove the **Window** pad from the system menu bar and the **Sum** option of the **Database** menu.

```
PUSH MENU _MSYSMENU
*** Remove a pad from the system menu bar ***
RELEASE PAD _MSM_WINDO OF _MSYSMENU
*** Remove an option from System menus ***
RELEASE BAR _MDA_SUM OF _MDATA
POP MENU _MSYSMENU
```

RELEASE MODULE

Purpose	Releases a binary file from memory
Syntax	RELEASE MODULE <file>
See Also	CALL, LOAD
Description	RELEASE MODULE removes a binary file from memory. Binary files are placed in memory with LOAD and are executed with CALL. Up to 16 binary files can be placed in memory at one time.
Clause	<file> <file> specifies the name of the binary file to clear.

RENAME

Purpose	Renames a file
Syntax	RENAME <file1> TO <file2>
See Also	COPY FILE, COPY TO, FILER

Description Use RENAME to change the name of a file, <file1>, to a new name, <file2>.

Include the file extensions for both files. If the file extensions are not included, the default extension .DBF is assumed. Include a period (.) after the file name to rename a file that doesn't have an extension.

Include paths with the file names if the files are not on the default drive and directory.

RENAME requires that <file2> doesn't already exist and <file1> must exist but cannot be open.

REPLACE

Purpose	Updates table/.DBF records
Syntax	REPLACE <field1> WITH <expr1> [ADDITIVE] [, <field2> WITH <expr2> [ADDITIVE]] ... [<scope>] [FOR <expL1>] [WHILE <expL2>] [NOOPTIMIZE]
See Also	GATHER, INSERT – SQL, SCATTER

Description REPLACE replaces data in a field with the value in an expression. Fields in unselected work areas must be prefaced with their alias.



No replacement occurs if the record pointer is at the end of the file in the current work area and you specify a field in another work area.

Clauses **<field1> WITH <expr1>**
[, <field2> WITH <expr2> ...

The data in <field1> is replaced with the value of the expression <expr1>, the data in <field2> is replaced with the value of the expression <expr2> and so on.

When the expression value is longer than the width of a numeric field, REPLACE forces the value to fit:

- First, decimal places are truncated and the remaining decimal portion of the field is rounded.
- If the value still doesn't fit, the field contents are stored in scientific notation.
- If the value still doesn't fit, asterisks replace the field contents.

ADDITIVE

The ADDITIVE clause applies to replacements in memo fields only. If ADDITIVE is included, replacements to memo fields are appended to the end of the memo fields. If ADDITIVE is not included, the memo field is overwritten with the value of the expression.

<scope>

The scope clauses are: ALL, NEXT <expN>, RECORD <expN>, and REST. These are explained in the Overview of the FoxPro Language chapter of this manual. Commands which include <scope> operate only on the table/.DBF in the active work area.

You can specify a scope of records to replace. Only the records that fall within the range of records specified by the scope are replaced.

The default scope for REPLACE is the current record (NEXT 1).

FOR <expL1>

If the FOR clause is included, the specified fields are replaced only in records for which <expL1> evaluates to true. Including FOR lets you conditionally replace records, filtering out undesired records.

Rushmore optimizes REPLACE FOR if <expL1> is an optimizable expression. For best performance, use an optimizable expression in the FOR clause. A discussion of Rushmore optimization appears in the Optimizing Your Application chapter in the FoxPro *Developer's Guide*.

WHILE <expL2>

If the WHILE clause is included, fields in records are replaced for as long as the logical expression <expL2> evaluates to true (.T.).

NOOPTIMIZE

To prevent Rushmore optimization of REPLACE, include NOOPTIMIZE. For further information on NOOPTIMIZE, see SET OPTIMIZE in this manual, and the discussion of Rushmore in the Optimizing Your Application chapter in the FoxPro *Developer's Guide*.

Example

The following example creates a temporary table/.DBF, defines and activates a data input window, and replaces the fields of a blank record in the temporary table/.DBF with information entered in the input window.

```

CLOSE DATABASES
USE customer
COPY NEXT 1 TO temp
USE TEMP
SET TALK OFF
SCATTER MEMVAR BLANK
DEFINE WINDOW menter FROM 7,10 to 17,70 PANEL
ACTIVATE WINDOW menter
@ 1,3 SAY 'Company: ' GET m.company
@ 3,3 SAY 'Address: ' GET m.address
@ 5,3 SAY 'City: ' GET m.city
@ 7,3 SAY 'State: ' GET m.state
@ 7,18 SAY 'Zip: ' GET m.zip
READ
DEACTIVATE WINDOW menter
RELEASE WINDOW menter
IF UPDATED( )
    APPEND BLANK
    REPLACE company WITH m.company
    REPLACE address WITH m.address
    REPLACE city WITH m.city
    REPLACE state WITH m.state
    REPLACE zip WITH m.zip
    BROWSE
ENDIF
CLOSE DATABASES
ERASE temp.dbf

```

REPLICATE()

Purpose	Returns a character string that contains a specified character expression repeated a specified number of times
Syntax	REPLICATE(<expC>, <expN>)
Parameters	<expC> The character expression to replicate <expN> The number of times to replicate the expression
Returns	Character
See Also	SPACE()

Description REPLICATE() returns a character string from a character expression repeated a specified number of times.

The maximum length of the resulting character string cannot exceed 64,000 characters in the Standard version of FoxPro for MS-DOS. In the 32-bit Extended version of FoxPro for MS-DOS and in FoxPro for Windows, the maximum length of the resulting character string is restricted by the amount of available memory.

Parameters **<expC>**
The character expression that is replicated is specified in <expC>.

<expN>
The character expression is replicated <expN> times.

Examples

```
? REPLICATE('HELLO ',5)
HELLO HELLO HELLO HELLO HELLO

@ 2,05,12,59 BOX REPLICATE(CHR(177),9)
```

REPORT

Purpose	Produces a report
Syntax	REPORT [FORM <file> ?] [ENVIRONMENT] [<scope>] [FOR <expL1>] [WHILE <expL2>] [HEADING <expC>] [NOEJECT] [NOCONSOLE] [NOOPTIMIZE] [PDSETUP] [PLAIN] [PREVIEW] [TO PRINTER [PROMPT] TO FILE <file>] [SUMMARY]
Remarks	PROMPT clause — FoxPro for Windows only
See Also	CREATE REPORT, MODIFY REPORT

Description REPORT displays or prints reports under the control of a report definition file created with MODIFY REPORT or CREATE REPORT.

The default extension for a report definition file is .FRX. If the report definition file is on a drive other than the default drive or directory, the path must also be included with the file name.

If REPORT is issued without any additional arguments, the Open dialog appears, displaying a list of existing report files to choose from.

Clauses **FORM** <file1>
You can specify which report to print by including the name of a report definition file in <file>.

?
If you include ? the Open dialog appears, displaying a list of existing report files to choose from.

ENVIRONMENT

When you create or modify reports, you can save the current FoxPro environment with the report definition file. Saving the FoxPro environment places an additional record in the report definition table. This record contains the names of all open tables and index files, the index order and any relationships between the tables.

Include `ENVIRONMENT` to restore the environment saved in the report definition file.

<scope>

The scope clauses are: `ALL`, `NEXT <expN>`, `RECORD <expN>`, and `REST`. These are explained in the Overview of the FoxPro Language chapter of this manual. Commands which include <scope> operate only on the table in the active work area.

You can specify a scope of records included in the report. Only the records that fall within the range of records specified by the scope are included in the report.

The default scope for `REPORT` is `ALL` records.

FOR <expL1>

If the `FOR` clause is included, data in only those records for which <expL1> evaluates to true (.T.) are printed. Including `FOR` lets you conditionally print data in records, filtering out undesired records.

Rushmore optimizes `REPORT FOR` if <expL1> is an optimizable expression. For best performance, use an optimizable expression in the `FOR` clause. A discussion of Rushmore optimization appears in the Optimizing Your Application chapter in the FoxPro *Developer's Guide*.

WHILE <expL2>

If the `WHILE` clause is included, record data is printed for as long as the logical expression <expL2> evaluates to true (.T.).

HEADING <expC>

Include `HEADING` to specify text for an additional heading to be placed on each page of the report. <expC> is the text. The `PLAIN` option takes precedence if both `HEADING` and `PLAIN` are included.

NOEJECT

`NOEJECT` suppresses a form feed to the printer prior to printing the report.

NOCONSOLE

Include NOCONSOLE to suppress the echoing of a report to the desktop in FoxPro for MS-DOS, the main FoxPro window in FoxPro for Windows or a user-defined window when a report is being printed or sent to a file.

NOOPTIMIZE

To prevent Rushmore optimization, include NOOPTIMIZE. For further information on NOOPTIMIZE, see SET OPTIMIZE in this chapter, and the discussion of Rushmore in the Optimizing Your Application chapter in the FoxPro *Developer's Guide*.

PDSETUP

In FoxPro for MS-DOS, when you create a report definition with the FoxPro Report Writer, you can specify a printer driver setup that determines the printed appearance of your report. If you save the environment with the report definition, the printer driver setup is also saved. If you include the PDSETUP key word when printing reports, the printer driver setup is loaded and used when the reports are printed.

In FoxPro for Windows, you can include PDSETUP to specify a printer driver setup for printing a character-based report created in FoxPro for MS-DOS. PDSETUP is ignored when you print a graphics-based report created in FoxPro for Windows.

PLAIN

If PLAIN is included, a page heading appears only at the beginning of the report.

PREVIEW

Include PREVIEW to display the report in page preview mode. If PREVIEW is included, the report is displayed in a page preview window and isn't sent to the printer. To print a report you must issue REPORT without the PREVIEW option.

Note that system memory variables are ignored when PREVIEW is included.

TO PRINTER [PROMPT] | TO FILE <file>

Include TO PRINTER to send a report to the printer.

In FoxPro for Windows, you can include the optional PROMPT clause to display a dialog before printing starts. Place the PROMPT key word immediately after TO PRINTER. PROMPT is ignored in FoxPro for MS-DOS.

In this dialog you can adjust printer settings. You can specify the number of copies and page numbers to print and direct print output to a file. The Windows Printer Setup dialog can be opened to adjust additional printer settings. The printer settings you can adjust depend on the currently installed printer driver.

Include TO FILE to send a character-based report created in FoxPro for MS-DOS to the text file specified with <file2>. The file created when you include TO FILE has the default extension .TXT.

In FoxPro for Windows, you cannot use the TO FILE option to send a graphics-based report created in FoxPro for Windows to a text file. Including TO FILE generates a “Feature not Available” error message in FoxPro for Windows. However, you can use the Windows Printer Setup to direct printed output to a file.

SUMMARY

SUMMARY suppresses detail line printing. Only totals and subtotals are printed.

RESTORE FROM

Purpose	Restores memory variables and arrays from a file or a memo field
Syntax	RESTORE FROM <file> FROM MEMO <memo field> [ADDITIVE]
See Also	DIMENSION, PUBLIC, PRIVATE, RELEASE, SAVE TO, STORE

Description RESTORE FROM retrieves memory variables and memory variable arrays saved in a memory variable file or a memo field and places them into memory.

All the memory variables and arrays are restored as PRIVATE when RESTORE FROM is issued in a program. Memory variables and arrays are restored as PUBLIC if RESTORE is issued in the Command window.

When RESTORE FROM is issued, any memory variables or arrays currently in memory are erased unless the ADDITIVE key word is included. System memory variables are not affected by RESTORE FROM.

<file>

Restores memory variables and arrays from the specified memory variable <file>. Memory variable files are assigned a .MEM extension.

MEMO <memo field>

Restores memory variables and arrays from the memo field specified in <memo field>.

ADDITIVE

Include the ADDITIVE key word to prevent any memory variables or arrays currently in memory from being erased. If the number of memory variables or arrays being added with the ADDITIVE key word plus the number of existing memory variables exceeds the memory variable limit, as many memory variables and arrays as possible are brought into memory from the memory variable file or memo field.

If you restore a memory variable that has the same name as an existing memory variable, the value in the existing memory variable is changed to the value of the restored memory variable.

Example In the following example, two memory variables are created. They are saved to a memory variable file and restored without clearing the existing memory variables.

```
val1 = 50
val2 = 'Hello'
SAVE TO temp
CLEAR MEMORY
val3 = DATE( )
RESTORE FROM temp ADDITIVE
DISPLAY MEMORY LIKE val?
```

RESTORE MACROS

Purpose	Restores keyboard macros to memory from a file or a memo field
Syntax	RESTORE MACROS [FROM <file> FROM MEMO <memo field>]
See Also	CLEAR, PLAY MACROS, SAVE MACROS

Description Use RESTORE MACRO to retrieve a set of keyboard macros that were saved in a keyboard macro file or a memo field.

You can define and save a series of keystrokes in memory as a keyboard macro. When you quit FoxPro, the macros you create are lost unless you use SAVE MACROS to store them to a macro file or a memo field. In FoxPro for MS-DOS you can also interactively save macros to a file by choosing **Macros...** from the **System** menu and then choosing **Save**. In FoxPro for Windows, choose **Macros...** from the **Program** menu and then choose **Save**.

In FoxPro for MS-DOS, macros can be restored from a file by choosing **Macros...** from the **System** menu popup and then choosing **Restore**. In FoxPro for Windows, choose **Macros...** from the **Program** menu popup and then choose **Restore** to restore macros from a file.

By default, restoring macros from a file or memo field is additive — the restored macros are appended to the existing set of macros in memory. If a macro name in the file or memo field is the same as an existing macro name, the macro in the file or memo field overwrites the existing macro.

Issuing RESTORE MACROS without a file name or memo field name clears all macros from memory and restores the default macros. The default macros are restored from the keyboard macro file named DEFAULT.FKY. The F2 through F9 keys are restored to the standard FoxPro startup settings if DEFAULT.FKY cannot be found.

Refer to the FoxPro *User's Guide* for more information on keyboard macros.

Clauses **FROM <file>**

Specifies the macro file that macros are to be restored from. Macro files have an .FKY extension.

FROM MEMO <memo field>

Specifies the memo field that macros are restored from.

RESTORE SCREEN

Purpose	Restores a desktop or screen
Syntax	RESTORE SCREEN [FROM <memvar>]
See Also	RESTORE FROM, SAVE SCREEN, SAVE TO
Description	RESTORE SCREEN restores a previously saved desktop (FoxPro for MS-DOS), main FoxPro window (FoxPro for Windows) or user-defined window from the screen buffer, a memory variable or an array element. Use SAVE SCREEN to place the current desktop, main FoxPro window or active user-defined window in the screen buffer, a memory variable or an array element. Memory variables or array elements that have screen or window images as their values appear with a data type of S when you issue DISPLAY or LIST MEMORY. The desktop, the main FoxPro window or user-defined windows saved to memory variables or array elements can also be saved to and restored from memory variable files with SAVE TO and RESTORE FROM.
Clauses	FROM <memvar> Specifies the name of a memory variable or memory variable array element from which a screen is restored. The desktop, the main FoxPro window or user-defined window is restored from the screen buffer if you omit the FROM clause.

RESTORE WINDOW

Purpose	Restores window definitions to memory from a file or a memo field
Syntax	RESTORE WINDOW <window list> ALL FROM <file> FROM MEMO <memo field>
See Also	ACTIVATE WINDOW, DEFINE WINDOW, SAVE WINDOW

Description RESTORE WINDOW restores window definitions to memory from a window file or a memo field. Use SAVE WINDOW to store window definitions in a window file or a memo field.

Any windows in memory with the same names as those restored are overwritten. A window's status (hidden, active, and so on) when it is saved is preserved when it is restored.

Clauses **<window list>**
Comma-delimited list of windows to be restored.

ALL
Restores all window definitions in the window file or memo field.

FROM <file>
<file> is the window file to restore the windows from. A window file has a .WIN extension.

FROM MEMO <memo field>
<memo field> is the memo field to restore the windows from.

Example In the following example, a window named OUTPUT is defined and saved to a memory variable. All windows are cleared and the window named OUTPUT is restored and activated.

```
CLEAR
DEFINE WINDOW output FROM 2,1 TO 13,75 TITLE 'Output' ;
  CLOSE FLOAT GROW SHADOW ZOOM
SAVE WINDOW output TO temp
CLEAR WINDOWS
RESTORE WINDOW output FROM temp
ACTIVATE WINDOW output
WAIT "The window Output has been restored" WINDOW
RELEASE WINDOW output
```


RESUME

Purpose	Resumes the execution of a suspended program
Syntax	RESUME
See Also	CANCEL, CLEAR, RETURN, SUSPEND

Description RESUME continues execution of a suspended program. Use SUSPEND to suspend program execution. The suspended program restarts at the line where execution was suspended.

SUSPEND and RESUMEs are valuable debugging tools. The current status of the FoxPro environment, including memory variables, window, menu bar or popup definitions, can be examined when a program is suspended.



To prevent commands executed when a program is suspended from interfering with subsequent program output, issue CLEAR to clear the desktop (FoxPro for MS-DOS), the main FoxPro window (FoxPro for Windows) or the active user-defined window before you resume program execution.

RETRY

Purpose	Re-executes the previous command
Syntax	RETRY
See Also	ON ERROR, RETURN, SET REPROCESS

Description RETRY returns control to the calling program and re-executes the last line executed in the program. RETRY is similar to RETURN except RETURN executes the *next* line in the calling program.

RETRY is useful in error-handling routines. RETRY is frequently used to re-execute a command until a record- or file-locking function succeeds in locking the record or file. SET REPROCESS can also be used to control retries of a record- or file-locking function. SET REPROCESS is preferable in most network situations.

Example In the following example, records from the CUSTOMER table/.DBF are printed. If an error occurs, the PROCEDURE named STOP is executed. STOP displays a message to check the printer and the program retries the command that caused the error.

```
CLOSE DATABASES
SET ESCAPE ON
SET TALK OFF
USE customer
CLEAR
ON ERROR DO stop
SET DEVICE TO PRINTER
SCAN
    @ PROW( )+1,10 SAY 'Company: ' + company
ENDSCAN
SET DEVICE TO SCREEN
ON ERROR

PROCEDURE stop
IF ERROR( ) = 125
    WAIT WINDOW 'Please check your printer'
    SET DEVICE TO PRINTER
    RETRY
ENDIF
RETURN
```

RETURN

Purpose	Returns program control to a calling program
Syntax	RETURN [<expr> TO MASTER TO <program name>]
See Also	DO, FUNCTION, PARAMETERS, PARAMETERS(), PRIVATE, PROCEDURE, PUBLIC, User-Defined Functions
Description	RETURN terminates execution of a program, procedure or function. Control is returned to the calling program, the highest level calling program, another program or the Command window. PRIVATE memory variables are released when RETURN is executed. RETURN is usually placed at the end of a program, procedure or function to return control to a higher-level program. However, an implicit RETURN is executed if RETURN is omitted.
Clauses	<expr> RETURN can be followed by an expression <expr> that is returned to the calling program. True (.T.) is automatically returned to the calling program if RETURN or the return expression is omitted. TO MASTER TO MASTER returns control to the highest level calling program. If the Run menu pad is present on the FoxPro system menu bar because FOXSTART.APP was executed on startup, RETURN TO MASTER returns control to FOXSTART.APP. TO <program name> Include the TO <program name> clause to return control to the program specified with <program name>. In the following example, the function LONGDATE returns a character string that is suitable for printing from a date.
Example	<pre>SET CENTURY ON ? longdate({08/19/92}) Wednesday, August 19, 1992 FUNCTION longdate PARAMETER mdate RETURN CDOW(mdate) + ', ' + MDY(mdate)</pre>

RIGHT()

Purpose	Returns the specified number of rightmost characters from a character string
Syntax	RIGHT(<expC>, <expN>)
Parameters	<expC> The expression characters are returned from <expN> The number of characters to be returned
Returns	Character
See Also	AT(), LEFT(), LTRIM(), RAT(), RTRIM(), SUBSTR(), TRIM()

Description RIGHT() returns the rightmost characters from a character expression. Characters are returned beginning with the last character on the right and continuing for a specified number of characters.

Parameters <expC>
Rightmost characters are returned from the character expression specified in <expC>.

<expN>
The number of characters returned from the character expression is specified by <expN>. All of the character expression is returned if <expN> is greater than the length of <expC>. The null string is returned if <expN> is negative or 0.

Example ? RIGHT('Redmond, WA',2)
WA

RLOCK()

Purpose	Attempts to lock a table/.DBF record or records						
Syntax	RLOCK([<expN> <expC1>] [<expC2>, <expN> <expC1>])						
Parameters	<table><tr><td><expN></td><td>The work area number for a table/.DBF</td></tr><tr><td><expC1></td><td>The table/.DBF alias</td></tr><tr><td><expC2></td><td>A list of one or more record numbers separated by commas</td></tr></table>	<expN>	The work area number for a table/.DBF	<expC1>	The table/.DBF alias	<expC2>	A list of one or more record numbers separated by commas
<expN>	The work area number for a table/.DBF						
<expC1>	The table/.DBF alias						
<expC2>	A list of one or more record numbers separated by commas						
Returns	Logical						
See Also	CLEAR, CLOSE, FLOCK(), LOCK(), SET MULTILOCKS, SET RELATION, SET REPROCESS, UNLOCK, USE						

Description Use RLOCK() to attempt to place a lock on a record or a set of records. RLOCK() is identical to LOCK().

If the lock or locks are successfully placed, RLOCK() returns true (.T.). Locked records are available for both read and write access by the user who placed the locks and are available for read-only access to all other users on the network.

Executing RLOCK() doesn't guarantee that the attempted record lock or locks are successfully placed. A record lock cannot be placed on a record already locked by another user or in a table/.DBF locked by another user. If the record lock or locks cannot be placed for any reason, RLOCK() returns false (.F.).

By default, RLOCK() makes one attempt to lock a record. Use SET REPROCESS to automatically retry a record lock when the first attempt fails. SET REPROCESS controls the number of lock attempts or the length of time lock attempts are made when the initial lock attempt is unsuccessful. See SET REPROCESS in this chapter for more information about SET REPROCESS and table/.DBF locking.

SET MULTILOCKS determines if you can lock multiple records in a table/.DBF. If MULTILOCKS is OFF (the default), you can only lock a single record in a table/.DBF. Multiple records in a table/.DBF can be locked when MULTILOCKS is ON. See SET MULTILOCKS in this chapter for further information.

Parameters <expN> | <expC1>

If you don't specify a work area or alias, RLOCK() attempts to lock the current record in the table/.DBF in the currently selected work area.

To attempt a record lock on the current record in a table/.DBF open in another work area, include the work area number <expN> or the table/.DBF alias <expC1>.

<expC2>

You must include <expC2> to attempt to lock multiple records. MULTLOCKS must be ON and you must include the work area or alias of the table/.DBF in which you attempt multiple record locks.

<expC2> is a list of one or more record numbers. RLOCK() attempts to lock all of these records. The record numbers in <expC2> are separated by commas. For example, to attempt record locks on the first 4 records in a table/.DBF, <expC2> will contain 1,2,3,4.

You can also lock multiple records by moving the record pointer to the record you would like to lock, issuing RLOCK() or LOCK() and then repeating this process for additional records.

If all the records specified in <expC2> are successfully locked, RLOCK() returns true (.T.). If one or more of the records specified in <expC2> cannot be locked, RLOCK() returns false (.F.) and none of the records are locked. In either case, existing record locks remain in place. Multiple record locking is an additive process — placing additional record locks doesn't release existing record locks.

From a performance perspective, it is always faster to lock the entire table/.DBF rather than to lock even a small number of records.

Unlocking Records

A table/.DBF record can only be unlocked by the user who placed the lock. Record locks can be released by issuing UNLOCK, closing the table/.DBF or exiting FoxPro.

UNLOCK can be used to release record locks in the current work area, a specific work area or in all work areas. See UNLOCK in this chapter for additional information about its use and syntax.

Toggling MULTLOCKS from ON to OFF or from OFF to ON implicitly performs UNLOCK ALL — all record locks in all work areas are released.

Tables/.DBFs can be closed with USE, CLEAR ALL or CLOSE DATABASES.

For additional information about record and file locking and sharing tables/.DBFs on a network, see the FoxPro in a Multi-User Environment chapter in the *FoxPro Developer's Guide*.

Example

The following example locks and unlocks the first four records of CUSTOMER.DBF and OFFICES.DBF.

```

SET MULTILOCKS ON
SET REPROCESS TO AUTOMATIC
STORE '1,2,3,4' TO mreclist
CLOSE DATA
SELECT 0
USE customer
SELECT 0
USE offices

? LOCK('1,2,3,4', 'customer')  && Lock the 1st 4 records in customer
? RLOCK(mreclist, 'offices')   && Lock the 1st 4 records in offices

UNLOCK IN customer
UNLOCK IN offices
SET MULTILOCKS OFF
SET REPROCESS TO 0

```

ROUND()

Purpose	Return a numeric expression rounded to a specified number of decimal places
Syntax	ROUND(<expN1>, <expN2>)
Parameters	<expN1> The numeric expression to be rounded <expN2> The number of decimals places to round to
Returns	Numeric
See Also	INT(), SET DECIMALS

Description ROUND() returns the value of a numeric expression rounded to a specified number of decimal places.

The value returned by ROUND() has the same number of decimal places as <expN2>. The number of decimal places specified by SET DECIMALS is ignored.

Parameters **<expN1>**
<expN1> is the numeric expression whose value is to be rounded.

<expN2>
<expN2> is the number of decimal places to round <expN1> to.

If <expN2> is negative, ROUND() returns a whole number containing <expN2> zeros to the left of the decimal point. For example, if <expN2> is -2 the first and second digits to the left of the decimal point in the value are zero.

Examples

```
SET DECIMALS TO 4
SET FIXED ON      && fix decimal display
? ROUND(1234.1962, 3)
? ROUND(1234.1962, 2)
? ROUND(1234.1962, 0)
? ROUND(1234.1962, -1)
? ROUND(1234.1962, -2)
? ROUND(1234.1962, -3)
SET FIXED OFF     && restore defaults
SET DECIMALS TO 2
```


ROW()

Purpose	Returns the current row position of the cursor
Syntax	ROW()
Returns	Numeric
See Also	COL(), PCOL(), PROW(), WCOLS(), WROWS()

Description ROW() returns the current row position of the cursor.

ROW() is especially useful for directing screen output to a position relative to the current row position of the cursor.

The special operator \$ can be used in place of the ROW() function.

Example

```
@ 5,5 SAY ' '
@ ROW( ) + 6, COL( ) SAY 'Contact person'
@ $ + 6, COL( ) SAY 'Contact person'
```

RTOD()

Purpose	Converts radians to degrees
Syntax	RTOD(<expN>)
Returns	Numeric
See Also	COS(), DTOR(), SIN(), TAN()

Description RTOD() converts a numeric expression that represents radians to an equivalent value in degrees. RTOD() is useful when working with FoxPro trigonometric functions such as COS(), SIN(), TAN() and so on. Use DTOR() to convert degrees to radians.

Parameter <expN>
<expN> is the numeric expression representing radians to be converted to degrees.

Examples

```
? RTOD(ACOS(0))  
90.00
```

STORE -1 to arcangle
? RTOD(ACOS(arcangle))
180.00

? RTOD(ACOS(SQRT(2)/2))
45.00

RTRIM()

Purpose	Returns the specified character expression with all trailing blanks removed
Syntax	RTRIM(<expC>)
Returns	Character
See Also	ALLTRIM(), LTRIM(), TRIM()

Description RTRIM() returns a character string that results from removing the trailing blanks from a character expression. RTRIM() is identical to TRIM().

Parameter <expC>
Include the character expression <expC> to trim the trailing blanks from.

Example

```

CLOSE DATABASES
SELECT 0
USE customer
GOTO 3
? 'The contact for ' + RTRIM(company) + ' is ' + contact
The contact for 1st Software Systems Ltd. is Rance Sivren

```

RUN !!

Purpose	Executes an external program
Syntax	RUN [/N [K]] <MS-DOS command program name> ! [/N [K]] <MS-DOS command program name>
See Also	GETENV()

Description The RUN command runs external operating commands or programs. RUN can be issued from within the Command window or from a program.



To use RUN under MS-DOS, the operating system file COMMAND.COM must be in the current directory or otherwise locatable by the MS-DOS COMSPEC parameter. See your MS-DOS manual for more information on SET COMSPEC.

Do not RUN disk reorganization programs like CHKDSK from within FoxPro. They modify the contents of your disk in a manner that can confuse FoxPro.

RUN and FoxPro for MS-DOS

FoxPro includes FoxSwap, a memory manager utility. FoxSwap is used to make more memory available for RUN.

The RUN command options /N or /NK specify the amount of memory FoxPro makes available for running external programs. N is a numeric value that specifies the amount of memory in kilobytes (K) that is made available to the RUN command. Do not include the letter N when specifying how much memory to make available.

Specifying N = 0 uses FoxSwap to its fullest extent. As much memory as possible is made available to the RUN command.

If a value greater than 0 is specified for N, FoxPro does the following:

- Buffers are flushed to disk.
- If N kilobytes of memory are available to RUN, RUN is invoked. If N kilobytes of memory are not available, FoxSwap is invoked.

For example, you can issue the following commands to make 120 kilobytes of memory available to run an external text editor called MYEDITOR:

```
RUN /120 MYEDITOR
! /120K MYEDITOR.
```

Although you can run larger external programs with FoxSwap, it takes additional time to executed FoxSwap.



Determine the amount of memory required to run an external program. Do not specify more than is needed. If enough memory is available, FoxSwap isn't used and the RUN command executes faster.

Example

```
STORE DIOC( DATE( ) ) TO today    && save today's date
! DATE 01/01/90
? DATE( )
RUN DATE &today    && restore today's date
? DATE( )
```

RUN and FoxPro for Windows

When RUN is used to run programs outside of FoxPro for Windows, it searches for programs in a slightly different manner than FoxPro for MS-DOS.

If the program you specify in RUN doesn't have an extension, FoxPro for Windows first looks along the MS-DOS path for a PIF (Program Information File) with the name you specify. If the PIF is found, the program specified in the PIF is executed with the PIF parameters. If a PIF isn't found, the DOS path is then searched for an executable file with the name you specify.

When a PIF isn't found, FoxPro for Windows uses FOXRUN.PIF, a PIF installed in the FoxPro for Windows directory. FOXRUN.PIF is configured to run the program in a window. You can modify FOXRUN.PIF to run programs in another configuration.

A PIF lets you run a non-Windows program under Windows. You can specify parameters for the program, whether the program is run in a window or full screen, the amount of memory allocated for the program and so on.

FOXRUN.PIF

FOXRUN.PIF allows you to execute MS-DOS and Windows programs and commands from within FoxPro for Windows by including one of the following commands in the Command window or in a program:

```
! [/N [<expN>]] <program name or command>
```

```
RUN [/N [<expN>]] <program name or command>
```

FOXRUN.PIF must be in the same directory as FOXPROW.EXE.

In FoxPro for MS-DOS /N and /NK specifies a numeric value for the amount of memory reserved for running external program. In FoxPro for Windows /N means NOWAIT. In FoxPro for Windows include the letter N (do not replace N with a numeric value and do not include K) to execute another Windows-based application. For example, the following opens the Windows character map accessory:

```
! /N CHARMAP.EXE
```

This example opens the Windows color picker from the Windows Control Panel:

```
! /N CONTROL COLOR
```

A Windows-based application executed with RUN /N or ! /N behaves the same way the application does when you open it through the Program Manager or the File Manager. You can switch between the application and FoxPro for Windows with the standard Windows operations.

You can include an optional numeric value immediately after /N to specify how the window application is opened. Do not include any spaces between /N and the numeric value. The following table lists the values for <expN> and how the Windows-based application is opened:

<expN1>	Application Attributes
1	Active and normal size
2	Active and minimized
3	Active and maximized
4	Inactivate and normal size
7	Inactivate and minimized

Running MS-DOS Programs in FoxPro for Windows

By default, FOXRUN.PIF runs the specified external MS-DOS program in a window. While the MS-DOS program or command is running, the title of the window is FoxPro Run Command. After the external program or command is finished executing, the window title changes to Inactive FoxPro Run Command. The FoxPro menu system is disabled until this window is closed.

You can use the Windows PIF editor to customize FOXRUN.PIF. You can edit the PIF to close the window automatically by checking **Close Window on Exit**, open external programs in a full screen by checking **Full Screen**, allocate memory for the program and so on.

Memory Considerations

By default, FOXRUN.PIF allocates a minimum of 256K of memory for running an external command or program. If you do not have 256K of free conventional memory, FoxPro displays an error message. To correct this:

- Close applications and files to free up additional memory.
- Edit FOXRUN.PIF to reduce the amount of memory required in the **KB Required** text box.

If the external command requires more than 256K, MS-DOS displays an “Insufficient memory” error in the FoxPro Run Command window. To correct this, edit your FOXRUN.PIF to increase the amount of memory required in the **KB Required** text box.

SAVE MACROS

Purpose	Saves a set of keyboard macros to a keyboard macro file or to a memo field
Syntax	SAVE MACROS TO <file name> TO MEMO <memo field>
See Also	CLEAR, PLAY MACRO, RESTORE MACROS

Description You can save a series of keystrokes in memory as a keyboard macro. When you quit FoxPro, the macros you create are lost unless you use SAVE MACROS to store them to a keyboard macro file or a memo field.

In FoxPro for MS-DOS, you can also save a macro to a file by choosing **Macros...** from the **System** menu and then choosing **Save**. In FoxPro for Windows, choose **Macros...** from the **Program** menu, and then choose **Save** to save a macro to a file.

See the FoxPro *User's Guide* for more information about creating, saving and restoring keyboard macros.

Clauses **TO <file name>**

Specify the file you want to save the macros to with <file name>. Keyboard macro file names must begin with a letter or an underscore and can contain any combination of a maximum of eight letters, numbers and underscores. A macro file cannot begin with a number. The keyboard macro file is assigned an .FKY extension. If you assign an extension other than .FKY, you must include that extension in RESTORE MACROS FROM.

TO MEMO <memo field>

Saves the macros to the memo field specified in <memo field>. The table/.DBF containing the memo field must be open. However, it does not have to be in the selected work area. To save macros to a table/.DBF in another work area, include the table/.DBF's alias when referencing the memo field. If you use a table/.DBF with multiple records, be sure the record pointer is positioned on the correct record before saving the macros.

Example The following example saves the current set of macros to a file called MYMACROS.FKY.

```
SAVE MACROS TO mymacros
```


SAVE SCREEN

Purpose	Saves a screen or window image to a buffer, a memory variable, or an array element
Syntax	SAVE SCREEN [TO <memvar>]
See Also	RESTORE FROM, RESTORE SCREEN, SAVE TO
Description	SAVE SCREEN saves an image of the current desktop in FoxPro for MS-DOS, the main FoxPro window in FoxPro for Windows or the active window to the screen buffer, a memory variable or an array element. An image stored to the screen buffer with SAVE SCREEN can be redisplayed with RESTORE SCREEN. Use RESTORE SCREEN FROM to redisplay images saved to a memory variable or an array element. Memory variables or array elements containing stored screen or window images have an S data type when you view the variables or array elements with DISPLAY or LIST MEMORY.
Clause	TO <memvar> <memvar> is the memory variable or array element you want to store the screen or window image to. If you omit this clause, the image is saved to the screen buffer.

SAVE TO

Purpose	Saves memory variables to a memory variable file or a memo field
Syntax	SAVE TO <file> TO MEMO <memo field> [ALL LIKE EXCEPT <skel>]
See Also	PRIVATE, PUBLIC, RELEASE, RESTORE FROM

Description SAVE stores current memory variables and memory variable arrays to a file or a memo field.

Use RESTORE FROM to load memory variables and memory variable arrays back into memory from a file or a memo field.

Clauses **TO <file>**
Specify the memory variable file to save memory variables to with <file>. The default extension for a memory variable file is .MEM.

TO MEMO <memo field>
Specify the memo field to save memory variables to with <memo field>.

ALL LIKE <skel>
Include the ALL LIKE clause to save all memory variables and arrays that match the skeleton <skel>. The skeleton can contain the wildcard characters ? and *.

ALL EXCEPT <skel>
Include the ALL EXCEPT clause to save all memory variables and arrays except those that match the skeleton <skel>. The skeleton can contain the wildcard characters ? and *.

Example

```
val1 = 50
val2 = 'Hello'
SAVE TO temp
CLEAR MEMORY
val3 = DATE( )
RESTORE FROM temp ADDITIVE
DISPLAY MEMORY LIKE val?
```


VAL3	Priv	D	10/25/92
VAL1	Priv	N	50
VAL2	Priv	C	"Hello"

SAVE WINDOWS

Purpose	Saves all or specified window definitions to a window file or a memo field
Syntax	SAVE WINDOWS <window list> ALL TO <file> TO MEMO <memo field>
See Also	DEFINE WINDOW, RESTORE SCREEN, RESTORE WINDOW, SAVE SCREEN
Description	SAVE WINDOWS saves current window definitions and window statuses to a file or a memo field. Window definitions can later be restored with RESTORE WINDOW. For example, if a window is hidden when it is saved to a file or memo field, it remains hidden when it is restored.
Clauses	<window list> Specify one or more windows to be saved with <window list>. Separate the window names with commas. ALL Include ALL to save all window definitions. TO <file> Specify the window file to save the window definitions to with <file>. If you don't specify an extension when naming the file, the default extension of .WIN is assigned. If you specify another extension when you save the window definitions to a file, you must include the extension when you restore the window definitions from the file. TO MEMO <memo field> Saves the window definitions to the memo field specified in <memo field>. The table/.DBF containing the memo field must be open. However, it does not have to be in the selected work area. To save window definitions to a table/.DBF in another work area, include the table's alias when referencing the memo field. If you use a table/.DBF with multiple records, be sure the record pointer is positioned on the correct record before saving the window definitions.

Example In this example, a window named Output is created, and the window definition is saved to the file TEMP.WIN. All windows are cleared and the Output window is restored from the file and activated.

```
CLEAR
DEFINE WINDOW output FROM 2,1 TO 13,75 TITLE 'Output' ;
    CLOSE FLOAT GROW SHADOW ZOOM
ACTIVATE WINDOW output

@ 1,1 SAY 'This is the contents of the window'

SAVE WINDOWS output TO temp
CLEAR WINDOWS

WAIT WINDOW 'The window has been saved - Press a key'

RESTORE WINDOW output FROM temp
ACTIVATE WINDOW output
WAIT WINDOW 'The window has been restored - Press a key'
DEACTIVATE WINDOW output
RELEASE WINDOW output

DELETE FILE temp.win
```

SCAN ... ENDSCAN

Purpose	Moves the record pointer through the current table/.DBF and executes a block of commands for each record that meets the specified conditions
Syntax	SCAN [NOOPTIMIZE] [<scope>] [FOR <expL1>] [WHILE <expL2>] [<statements>] [LOOP] [EXIT] ENDSCAN
See Also	DO CASE ... ENDCASE, DO WHILE ... ENDDO, FOR ... ENDFOR
Description	SCAN moves through a table/.DBF and executes a block of commands. SCAN automatically advances the record pointer to the next record that meets the specified conditions and executes the commands again. You can place comments after ENDSCAN on the same line. The comments are ignored during program compilation and execution.
Clauses	NOOPTIMIZE To disable Rushmore optimization of SCAN, include NOOPTIMIZE. For further information on NOOPTIMIZE, see SET OPTIMIZE in this chapter, or consult the discussion of Rushmore optimization in the Optimizing Your Application chapter in the <i>FoxPro Developer's Guide</i>. <scope> The scope clauses are: ALL, NEXT <expN>, RECORD <expN>, and REST. These are explained in the Overview of the FoxPro Language chapter of this manual. Commands which include <scope> operate only on the table/.DBF in the selected work area. You can specify a scope of records that are scanned. Only the records that fall within the range of records specified by the scope are scanned. The default scope for SCAN is ALL records.

FOR <expL1>

Including the FOR clause lets you filter out undesired records. If the FOR clause is included, the commands are executed for all records within the scope for which <expL1> is true.

Rushmore optimizes a query created with SCAN ... FOR if <expL1> is an optimizable expression. For best performance, use an optimizable expression in the FOR clause. A discussion of Rushmore optimizable expressions appears in the Optimizing Your Application chapter in the *FoxPro Developer's Guide*.

WHILE <expL2>

If the WHILE clause is included, the commands are executed as long as <expL2> remains true.

<statements>

Specify the FoxPro commands to be executed with <statements>.

LOOP

LOOP returns control directly back to SCAN and can be placed anywhere between SCAN and ENDSCAN.

EXIT

EXIT transfers control outside the SCAN ... ENDSCAN loop to the first command immediately following ENDSCAN.

SCATTER

Purpose	Copies data from the current record to an array or a set of memory variables
Syntax	SCATTER [FIELDS <field list>] [MEMO] TO <array> TO <array> BLANK MEMVAR MEMVAR BLANK
See Also	APPEND FROM ARRAY, COPY TO ARRAY, DECLARE, DIMENSION, GATHER

Description SCATTER copies data from the current record to a set of memory variables or a memory variable array. SCATTER automatically creates the memory variables or arrays if they don't already exist.

Use GATHER to copy memory variables or array elements to table/.DBF records.

Clauses **FIELDS <field list>**

Only the fields contained in the field list you specify with <field list> are transferred to the memory variables or the array. All fields are transferred if you omit FIELDS <field list>. The field list can contain memo fields if you include the MEMO key word.

MEMO

By default, memo fields are ignored in SCATTER. Include the MEMO key word if you include a memo field or fields in the field list.

You must have sufficient memory to scatter large memo fields to memory variables or an array. The message “Insufficient memory” is displayed if you lack sufficient memory. If a memo field is too large to fit in memory, neither it nor any additional memo fields in the field list are scattered. If a memo field isn't scattered, its memory variable or array element is set to false (.F.).

TO <array>

The contents of the fields in the record are copied into each element of the array specified with <array> in sequential order starting with the first field.

Extra array elements are left unchanged if the array has more elements than the number of fields. A new array is automatically created if the array doesn't already exist or if it has fewer elements than the number of fields. The array elements have the same data types as the corresponding fields.

TO <array> BLANK

An array is created with empty elements if BLANK is included. The elements are the same size and type as the fields in the table/.DBF.

MEMVAR

MEMVAR scatters the data to a set of memory variables instead of to an array. One memory variable is created for each field in the table/.DBF. Each memory variable is filled with data from the corresponding field in the current record and is assigned the same name, type and size as its field.

A memory variable is created for each field in the field list if a field list is included.

Preface the memory variable name with the M. qualifier to reference a memory variable that has the same name as a field in the current table/.DBF.



Don't include TO with MEMVAR. An array named MEMVAR is created if TO is included.

MEMVAR BLANK

A set of empty memory variables is created if MEMVAR BLANK is included. Each memory variable is assigned the same name, data type and size as its field. If a field list is included, a memory variable is created for each field in the field list.

Example

This program uses SCATTER to copy the data in the current record to a set of memory variables. The data is displayed in a window for editing and then copied back to the current record using GATHER (unless Escape is pressed).

```

SET TALK OFF
USE salesman
COPY TO temp
USE temp
SCATTER MEMVAR MEMO      && save old values
DEFINE WINDOW menter FROM 5,10 to 20,70 PANEL
ACTIVATE WINDOW menter
@ 1,3   SAY 'Name: '   GET m.name
@ 3,3   SAY 'Address: ' GET m.address
@ 5,3   SAY 'City: '   GET m.city
@ 7,3   SAY 'State: '  GET m.state
@ 7,18  SAY 'Zip: '    GET m.zip
@ 9,13  SAY 'Notes:'
@ 9,23  EDIT m.notes SIZE 3,26
@ 13,13 SAY 'Press Escape to discard changes'
READ

IF LASTKEY( ) != 27
    GATHER MEMVAR  && save changes
    WAIT WINDOW 'Changes saved' NOWAIT
ELSE
    && no GATHER
    WAIT WINDOW 'Changes not saved' NOWAIT
ENDIF
DEACTIVATE WINDOW menter
BROWSE
USE

```

SCHEME()

Purpose	Returns a color pair or a color pair list from a specified color scheme
Syntax	SCHEME(<expN1> [, <expN2>])
Parameters	<expN1> The color scheme number <expN2> The position of the color pair in the color scheme
Returns	Character
See Also	SET COLOR

Description SCHEME() returns a color pair or color pair list from a color scheme.

Parameters <expN1>
<expN1> is the number of the color scheme for which you want a complete color listing. 10 color pairs are returned.

<expN2>
Include <expN2> to return a single color pair from a color scheme. The position of the color pair in the color scheme is specified with <expN2>. For example, if <expN2> is 4, the fourth color pair is returned.

Example The following command returns the third color pair from color scheme number 4:

? SCHEME(4,3)

SCOLS()

Purpose	Returns the number of columns available on the screen
Syntax	SCOLS()
Returns	Numeric
See Also	COL(), ROW(), SET DISPLAY, SROWS(), WCOLS(), WROWS()

Description This function returns the number of columns available on the desktop in FoxPro for MS-DOS or the main FoxPro window in FoxPro for Windows.

The value returned by SCOLS() depends on the current display mode. Display modes can be changed with SET DISPLAY in FoxPro for MS-DOS. In FoxPro for Windows, use Windows Setup to change the display mode.

SCROLL

Purpose	Scrolls an area of the desktop, the main FoxPro window or a user-defined window up, down, left, or right
Syntax	SCROLL <row1, column1>, <row2, column2>, <expN1> [, <expN2>]
See Also	MOVE WINDOW

Description Use this command to scroll an area of the desktop or a window vertically or horizontally.

Clauses <row1, column1>, <row2, column2>

Scrolling occurs in a rectangular region of the desktop, the main FoxPro window or the active user-defined window. The upper-left corner of the region is specified with <row1, column1>, and the lower-right corner is specified with <row2, column2>.

<expN1>

The numeric expression <expN1> specifies how many rows the area inside the rectangular region is scrolled up or down. The area is scrolled upward <expN1> rows if <expN1> is positive. The area is scrolled downward <expN1> rows if <expN1> is negative. The area is cleared if <expN1> is 0 and <expN2> is omitted.

<expN2>

The numeric expression <expN2> specifies how many columns the area inside the rectangular region is scrolled left or right. The area is scrolled to the right <expN2> columns if <expN2> is positive. The area is scrolled to the left <expN2> columns if <expN2> is negative. The area can be scrolled diagonally if both <expN1> and <expN2> are included.

For example, the following command scrolls a small rectangular region.

```
CLEAR
@ 4, 1 FILL TO 10, 8 COLOR GR+/B
WAIT WINDOW 'Press key to scroll off left top corner'
SCROLL 0, 0, 5, 5, -2, 1
```

SECONDS()

Purpose	Returns the number of seconds that have elapsed since midnight
Syntax	SECONDS()
Returns	Numeric
See Also	SYS(2), TIME()

Description SECONDS() returns the number of seconds that have elapsed since midnight with a resolution of 1 millisecond. The numeric value is returned in the format seconds.thousandths of a second.

Examples ? SECONDS()
 71798.533

 ? SECONDS() / (60 * 60)
 19.944

SEEK

Purpose	Searches the current indexed table/.DBF
Syntax	SEEK <expr>
See Also	EOF(), FOUND(), INDEX, LOCATE, RECNO(), SEEK(), SET EXACT, SET NEAR

Description SEEK searches the current indexed table/.DBF for the first occurrence of a record whose index key expression matches a general expression. You can use SEEK only with indexed tables/.DBFs, and you can search only on the index key expression. The match must be exact unless SET EXACT is OFF.

If SEEK finds a matching record, RECNO() returns the record number of the matching record, FOUND() returns true (.T.) and EOF() returns false (.F.).

If no match is found, RECNO() returns the number of records in the table/.DBF plus 1, FOUND() returns false (.F.) and EOF() returns true (.T.).

The record pointer is positioned immediately after the closest matching record if SET NEAR is ON. The record pointer is positioned at the end of the file if SET NEAR is OFF. In either case, RECNO(0) returns the record number of the closest record.

Clause **<expr>**

Specify the index key expression you want SEEK to search for with <expr>.

Example In this example, CUSTOMER.DBF is opened and indexed on the COMPANY field. SEEK is used to find the index key expression that matches the value contained in the memory variable MVAL.

```
CLOSE DATABASES
USE customer
INDEX ON UPPER(company) TAG Company
SET EXACT OFF
STORE 'AZ INC' TO mseekval
SEEK mseekval
IF FOUND( )
    DISPLAY FIELDS company, contact
ENDIF
```

SEEK()

Purpose	Searches the current indexed table/.DBF and returns a value that indicates whether the search is successful
Syntax	SEEK(<expr> [, <expN> <expC>])
Parameters	<expr> The expression to search for <expN> The work area number of a table/.DBF <expC> The table/.DBF or work area alias
Returns	Logical
See Also	EOF(), FOUND(), LOCATE, SEEK

Description SEEK() searches the current indexed table/.DBF for the first occurrence of a record whose index key expression matches a specified expression. If a match is found, true (.T.) is returned and the record pointer is moved to the matching record. If a match isn't found, false (.F.) is returned and the record pointer is positioned at the end of the file.



Issuing SEEK() is equivalent to issuing both SEEK and FOUND().

The table/.DBF you search must be indexed.

Clauses **<expr>**
 The index key expression to search for is specified with <expr>.

<expN> | <expC>
 SEEK() searches for a matching record in the table/.DBF in the current work area. To search for a matching record in a table/.DBF open in another work area, specify the work area number with <expN> or the table/.DBF or work area alias with <expC>.

Examples `USE customer ORDER on`
`? SEEK('2')`
`.T.`

SELECT

Purpose	Activates the specified work area
Syntax	SELECT <expN> <expC>
See Also	ALIAS(), SELECT(), USE

Description SELECT activates a work area.

Clauses **<expN> | <expC>**

By default, work area number 1 is active when FoxPro is started. You can specify a different work area with <expN>. The lowest-numbered unused work area is selected if <expN> is 0.

When a table/DBF is opened in a work area, you can select the table's work area by specifying the table's alias with <expC>.

Table/DBF Fields

Fields in tables/DBFs in any work area can be included in FoxPro commands and functions. Use the following formats to access fields in a table/DBF open in a work area other than the current work area:

<alias>.<field> or <alias> -> <field>

Example The following example demonstrates ways of selecting work areas.

```
CLOSE DATABASES
SELECT 1  && Work area 1
USE customer
SELECT 2  && Work area 2
USE invoices
SELECT customer  && Work area 1
BROWSE
SELECT B  && Work area 2
BROWSE
```


SELECT()

Purpose	Returns the number of the current work area or the unused work area with the highest number
Syntax	SELECT([0 1])
Returns	Numeric
See Also	ALIAS(), SELECT, SET COMPATIBLE

Description SELECT() returns the number of the current work area if SET COMPATIBLE is OFF. If SET COMPATIBLE is ON, SELECT() returns the number of the unused work area with the highest number.

A work area can be selected (activated) with SELECT.

Parameters 0 | 1

The number of the current work area is returned if you include 0 in SELECT(). The number of the unused work area with the highest number is returned if you include 1.

SELECT — SQL

Purpose	Retrieves data from one or more table/.DBF
Syntax	<pre>SELECT [ALL DISTINCT] [<alias>.]<select_item> [AS <column_name>] [, [<alias>.]<select_item> [AS <column_name>] ...] FROM <table> [<local_alias>] [, <table> [<local_alias>] ...] [[INTO <destination>] [TO FILE <file> [ADDITIVE] TO PRINTER [PROMPT] TO SCREEN]] [PREFERENCE <name>] [NOCONSOLE] [PLAIN] [NOWAIT] [WHERE <joincondition> [AND <joincondition> ...] [AND OR <filtercondition> [AND OR <filtercondition> ...]]] [GROUP BY <groupcolumn> [, <groupcolumn> ...]] [HAVING <filtercondition>] [UNION [ALL] <SELECT command>] [ORDER BY <order_item> [ASC DESC] [, <order_item> [ASC DESC]...]]</pre>
See Also	CREATE QUERY, CREATE TABLE — SQL, INSERT — SQL, MODIFY QUERY, SET ANSI, SET PATH, _TALLY

Description SELECT is a SQL command you use to retrieve data from one or more table/.DBF. When you use SELECT to pose a query, FoxPro interprets the query and retrieves the specified data from the tables/.DBFs. SELECT is built into FoxPro like any other FoxPro command. You can create a SELECT query:

- In the Command window
- In a FoxPro program (like any other FoxPro command)
- In the RQBE window

When you issue SET TALK ON and execute SELECT, FoxPro displays the query time and the number of records in the results. _TALLY stores the number of records in the query results.

Clauses **[ALL | DISTINCT]**

The **SELECT** clause specifies the fields, constants and expressions that will be displayed in the query results.

By default, all of the rows (**ALL**) in the query results are displayed. Include **DISTINCT** to exclude duplicates of any rows from the query results.



You can use **DISTINCT** only once per **SELECT** clause.

**[<alias>.<select_item> [AS <column_name>]
[, [<alias>.<select_item> [AS <column_name>] ...]**

<select_item> can be one of the following:

- The name of a field from a table/.DBF in the **FROM** clause
- A constant specifying that the same constant value is to appear in every row of the query results
- An expression that can be the name of a user-defined function (UDF)

Each item you specify with <select_item> generates one column of the query results. If more than one item has the same name, be sure to qualify the matching names by including the table/.DBF alias and a period before the item name.



Although using UDFs in the SELECT clause has obvious benefits, you should also consider the following restrictions:

1. The speed of operations performed with SELECT may be limited by the speed at which such UDFs are executed. High-volume manipulations involving UDFs may better be accomplished by using API and UDFs written in C or assembly language.
2. You can assume *nothing* about the FoxPro input/output (I/O) or table/.DBF environment in UDFs invoked from SELECT. In general, you don't know which work area is selected, the name of the current table/.DBF or even the names of the fields being processed. The value of these variables depend on precisely where in the optimization process the UDF is invoked.
3. It isn't safe to change the FoxPro I/O or table/.DBF environment in UDFs invoked from SELECT. In general, the results are unpredictable.
4. The *only* reliable way to pass values to UDFs invoked from SELECT is by the argument list passed to the function when it is invoked.
5. If you experiment and discover a supposedly forbidden manipulation that works correctly in a certain version of FoxPro, there is no guarantee that it will continue to work in later versions.

Apart from these restrictions, feel free to use UDFs. However, don't forget the effect of using SELECT on performance.

The following field functions are available for use with a select item that is a field or an expression involving a field:

- AVG(<select_item>) — Averages a column of numeric data.
- COUNT(<select_item>) — Counts the number of select items in a column. COUNT(*) counts the number of rows in the query output.
- MIN(<select_item>) — Determines the smallest value of <select_item> in a column.

- **MAX(<select_item>)** — Determines the largest value of <select_item> in a column.
- **SUM(<select_item>)** — Totals a column of numeric data.

You cannot nest field functions.

The optional **AS <column_name>** clause specifies the heading for a column in the query output. This is useful when <select_item> is an expression or contains a field function and you want to give the column a meaningful name. <column_name> can be an expression but cannot contain characters (for example, spaces) that aren't permitted in table/.DBF field names.

FROM <table> [<local_alias>] [, <table> [<local_alias>] ...]

The FROM clause lists the tables/.DBFs that contain the data to be retrieved by the query. If no table/.DBF is open in a work area in the current directory or the FoxPro path, the Open dialog appears so you can specify the file location.

If a table/.DBF in the query isn't open in a work area, it is opened and remains open in a work area once the query is complete.

<local_alias> is a temporary name for a table/.DBF specified with <table>. If you specify a local alias, you must use the local alias in place of the table/.DBF name throughout SELECT. The local alias doesn't affect the FoxPro environment.



If several tables have fields with the same name, which must be qualified, you may want to use a short local alias.

INTO <destination>

The INTO clause determines where the query results are stored. If you include the INTO clause, no output display is produced. This means that if you include an INTO clause and a TO clause in the same query, the TO clause is ignored. If you don't include the INTO clause, query results are displayed in a Browse window. Query results can also be directed to the printer or a file with the TO clause.

<destination> can be one of the following:

- **ARRAY <array>** — Stores query results in a memory variable array named <array>. The array isn't created if the query selects 0 records.
- **CURSOR <cursor>** — Stores query results in a cursor named <cursor>. If you specify the name of an open table/.DBF, FoxPro closes the table/.DBF and creates a cursor by that name without warning if SET SAFETY is OFF. After SELECT is executed, the temporary cursor remains open and is active but is read-only. Once you close this temporary cursor, it is deleted. Cursors can exist as a temporary file on the SORTWORK drive.
- **DBF <table> | TABLE <table>** — Stores query results in a table/.DBF named <table>. If you specify a table/.DBF that is already open, FoxPro closes the table/.DBF and reopens it without warning if SET SAFETY is OFF. If you don't specify an extension, the table/.DBF is given a .DBF extension. The table/.DBF remains open and active after SELECT is executed.

TO FILE <file> [ADDITIVE]

| TO PRINTER [PROMPT]

| TO SCREEN

If you include a TO clause but not an INTO clause, you can direct query results to an ASCII text file named <file> or to the printer in addition to the desktop or main FoxPro window.

If query results are directed to a text file, including ADDITIVE appends output to any existing contents of the file. If you don't include a TO clause or an INTO clause, the query results appear on the screen by default unless you specify NOCONSOLE.

In FoxPro for Windows, if query results are directed to the printer, you can include the optional PROMPT clause to display a dialog before printing starts. Place the PROMPT key word immediately after TO PRINTER. PROMPT is ignored in FoxPro for MS-DOS.

In this dialog you can adjust printer settings. You can specify the number of copies and page numbers to print and direct print output to a file. The Windows Printer Setup dialog can also be opened to adjust additional printer settings. The printer settings you can adjust depend on the currently installed printer driver.

By default, query results are sent to a Browse window. Include TO SCREEN to direct query results to the desktop (FoxPro for MS-DOS), the main FoxPro window (FoxPro for Windows) or an active user-defined window.

When query results are output, columns are named according to the following rules:

- If a select item is a field with a unique name, the output column name is the field's name.
- If more than one select item has the same name (for example, CUST.ZIP and STATE.ZIP) output columns are named <extension>_A and <extension>_B (ZIP_A and ZIP_B). For a select item with a 10-character name, the name is truncated to add the underscore and letter.
- If a select item is an expression, its output column is named EXP_A. Any other expressions are named EXP_B, EXP_C, and so on.
- If a select item contains a field function such as COUNT(), the output column is named CNT_A. If another select item contains SUM(), its output column is named SUM_B.

PREFERENCE <name>

If query results are sent to a Browse window, you can include PREFERENCE to save the Browse window's attributes and options for later use. PREFERENCE saves the Browse window's attributes indefinitely in the FOXUSER resource file. Preferences can be retrieved at any time. Issuing SELECT with a PREFERENCE <name> for the first time creates the preference. Issuing SELECT later with the same preference name restores the Browse window to that preference state. When the Browse window is closed, the preference is updated.

If you exit a Browse window by pressing Ctrl+Q, any Browse window changes are not saved to the resource file.

NOCONSOLE

Including NOCONSOLE prevents the query results from being displayed on the desktop or main FoxPro window. NOCONSOLE can be used whether or not a TO clause is present. If an INTO clause is included, NOCONSOLE is ignored.

PLAIN

Including PLAIN prevents column headings from appearing in the query output displayed. PLAIN can be used whether or not a TO clause is present. If an INTO clause is included, PLAIN is ignored.

NOWAIT

When query results are directed to a Browse window in a program, including **NOWAIT** continues program execution after the Browse window is opened. The program doesn't wait for the Browse window to be closed, but continues execution on the program line immediately following the **SELECT** statement.

When **TO SCREEN** is included to direct output to the desktop (FoxPro for Windows), the main FoxPro window (FoxPro for Windows) or a user-defined window, output pauses when the desktop or main FoxPro window is full of query results. Press a key to see the next set of query results. If **NOWAIT** is included, the query results are scrolled off the desktop, the main FoxPro window or the user-defined window without pausing for a key press. **NOWAIT** is ignored if included with the **INTO** clause.

**WHERE <joincondition> [AND <joincondition> ...]
[AND | OR <filtercondition> [AND | OR <filtercondition> ...]]**

WHERE is required to retrieve data from multiple tables. It tells FoxPro to include only certain records in the query results.

<joincondition> specifies fields that link the tables in the **FROM** clause. If you include more than one table/.DBF in a query, you should specify a join condition for every table/.DBF after the first.



If you include two table in a query and don't specify a join condition, every record in the first table/.DBF is joined with every record in the second table/.DBF as long as the filter conditions are met. Such a query can produce lengthy results.

Be careful when using functions (DELETED(), RECNO(), EOF(), FOUND(), RECCOUNT(), and so on) that support an optional alias or work area in join conditions. Including an alias or work area in these functions may yield unexpected results. SELECT doesn't use your work areas; it performs the equivalent of USE ... AGAIN. Single-table queries that use these functions without an optional alias or work area will return proper results. However, multiple-table queries that use these functions — even without an optional alias or work area — may return unexpected results.

Use caution when joining tables with empty fields because FoxPro matches empty fields. For example, if you join on CUSTOMER.ZIP and INVOICE.ZIP, and CUSTOMER contains 100 empty zip codes and INVOICE contains 400 empty zip codes, the query output contains 40,000 extra records resulting from the empty fields. Use the EMPTY() function to eliminate empty records from the query output.

Multiple join conditions must be connected with the AND operator. Each join condition has the following form:

<field1> <comparison> <field2>

where <field1> is the name of a field from one table, <field2> is the name of a field from another table, and <comparison> is one of the following operators:

<comparison>
=
< >, !=, #
= =
>
> =
<
< =

When you use the = operator with strings, it acts differently depending on the setting of SET ANSI. When SET ANSI is OFF, FoxPro treats string comparisons in the manner with which Xbase users are familiar. When SET ANSI is ON, string comparisons follow ANSI standards.

<filtercondition> specifies the criteria that records must meet to be included in the query results. A query can include as many filter conditions as desired, connected with the AND or OR operator. You can also use the NOT operator to reverse the value of a logical expression or use EMPTY() to check for an empty field. <filtercondition> can take one of these forms:

Form: <field1> <comparison> <field2>

Example: customer.cust_id = payments.cust_id

Form: <field> <comparison> <expression>

Example: payments.amount >= 1000

Form: <field> <comparison> ALL (<subquery>)

Example: taxrate < ALL ;

(SELECT taxrate FROM customer WHERE state = "WA")

When the filter condition includes ALL, the field must meet the comparison condition for all values generated by the subquery before its record is included in the query results.

Form: <field> <comparison> ANY | SOME (<subquery>)

Example: taxrate < ANY ;

(SELECT taxrate FROM customer WHERE state = "WA")

When the filter condition includes ANY or SOME, the field must meet the comparison condition for at least one of the values generated by the subquery.

Form: <field> [NOT] BETWEEN <start_range> AND <end_range>

Example: customer.taxrate BETWEEN 5.50 AND 6.00

This example checks to see if the values in the field are within a specified range of values.

Form: [NOT] EXISTS (<subquery>)

Example: EXISTS ;

(SELECT * FROM invoices WHERE customer.zip = invoices.zip)

This example checks to see if at least one row meets the criteria in the subquery. When the filter condition includes EXISTS, the filter condition evaluates to true unless the subquery evaluates to the empty set.

Form: <field> [NOT] IN <value_set>

Example: customer.zip NOT IN (“98052”, “98072”, “98034”)

When the filter condition includes IN, the field must contain one of the values before its record is included in the query results.

Form: <field> [NOT] IN (<subquery>)

Example: customer.cust_id IN ;

(SELECT tranfile.cust_id FROM tranfile WHERE tranfile.item=“FoxPro”)

Here, the field must contain one of the values returned by the subquery before its record is included in the query results.

Form: <field> [NOT] LIKE <expC>

Example: customer.state NOT LIKE “WA”

This filter condition searches for each field that matches <expC>. You can use the wildcard characters % (percent sign) and _ (underscore) as part of <expC>. The underscore represents a single unknown character in the string.

A subquery is SELECT within a SELECT and *must* be enclosed in parentheses. You can have multiple subqueries at the same level (not nested) in the WHERE clause. Subqueries can contain multiple join conditions.

GROUP BY <groupcolumn> [, <groupcolumn> ...]

Including the GROUP BY clause groups rows in the query based on values in one or more columns. <groupcolumn> can be the name of a regular table/.DBF field, or a field that includes an SQL field function, or a numeric expression indicating the location of the column in the result table/.DBF (the leftmost column number is 1).

HAVING <filtercondition>

If you include the HAVING clause, FoxPro will include groups in the query results as long as they meet the filter condition specified with <filtercondition>. <filtercondition> cannot contain a subquery.

HAVING <filtercondition> should be used with GROUP BY to specify the criteria that groups must meet to be included in the query results. HAVING can include as many filter conditions as desired, connected with the AND or OR operator. You can also use NOT to reverse the value of a logical expression.

A HAVING clause without a GROUP BY clause acts like a WHERE clause. You can use local aliases and field functions in the HAVING clause. Use a WHERE clause for faster performance if your HAVING clause contains no field functions.

[UNION [ALL] <SELECT command>]

Include the UNION clause to combine the final results of one SELECT with the final results of another SELECT. By default, UNION checks the combined results and eliminates duplicate rows. Use the optional ALL key word to prevent UNION from eliminating duplicate rows from the combined results.

You can use parentheses to combine multiple UNION clauses.

UNION clauses follow these rules:

- You cannot use UNION to combine subqueries.
- Both SELECTs must output the same number of columns.
- Each column in the query results of one SELECT must have the same data type and width as the corresponding column in the other SELECT.
- Only the final SELECT can have an ORDER BY clause, which must refer to output columns by number. If an ORDER BY clause is included, it affects the entire result.

**ORDER BY <order_item> [ASC | DESC]
[, <order_item> [ASC | DESC] ...]**

ORDER BY sorts the query results based on the data in one or more columns. Each order item must correspond to a column in the query results and can be one of the following:

- A field in a FROM table that is also a select item in the main SELECT clause (not in a subquery).
- A numeric expression indicating the location of the column in the result table. (The leftmost column is number 1.)

You can specify the optional DESC key word if you want your query results to appear in descending order according to the order item or items. ASC specifies an ascending order for query results and is the default for ORDER BY.

Query results appear unordered if you don't specify an order with ORDER BY.

Examples

This example displays the names of all companies in the CUSTOMER.DBF (one field from one table).

```
SELECT customer.company ;
FROM customer
```

This example displays the contents of three fields from two tables and joins the two tables based on the CUST_ID field. It uses local aliases for both tables.

```
SELECT a.company, b.idate, b.itotal ;
FROM customer a, invoices b ;
WHERE a.cno = b.cno
```

This example displays only records with unique data in the specified fields.

```
SELECT DISTINCT a.company, b.idate, b.itotal ;
FROM customer a, invoices b ;
WHERE a.cno = b.cno
```

This example displays STATE, ZIP and COMPANY fields in ascending order.

```
SELECT state, zip, company, cno;
FROM customer ;
ORDER BY state, zip, company
```

This example stores the contents of fields from two tables in a third table.

```
SELECT a.company, b.idate, b.itotal ;
  FROM customer a, invoices b ;
  WHERE a.cno = b.cno ;
  INTO TABLE custinv.dbf
BROWSE
CLOSE DATA
```

This example displays only records with an invoice date earlier than 05/08/90.

```
SELECT a.company, b.idate, b.itotal ;
  FROM customer a, invoices b ;
  WHERE a.cno = b.cno ;
  AND b.idate < {05/08/90}
```

This example displays the names of all companies from CUSTOMER.DBF with a zip code that matches a zip code in the INVOICES table/.DBF.

```
SELECT company FROM customer a WHERE ;
  EXISTS (SELECT * FROM offices b WHERE a.zip = b.zip)
```

This example displays all records from CUSTOMER.DBF having a company name that begins with an upper-case C and is an unknown length.

```
SELECT * FROM customer a WHERE a.company LIKE "C%"
```

This example displays all records from CUSTOMER.DBF having a state name that begins with an upper-case N and is followed by one unknown character.

```
SELECT * FROM customer a WHERE a.state LIKE "N_"
```

This example displays the names of all cities in CUSTOMER.DBF in upper-case and names the output column CityList.

```
SELECT UPPER(city) AS CityList FROM customer
```

SET

Purpose	Opens the View window
Syntax	SET
See Also	SET VIEW

Description SET opens the View window, which is one of the most useful FoxPro tools: It provides an easy way to open tables/.DBFs, establish relations or set or change many FoxPro options.

For more information about the View window, see the FoxPro *User's Guide*.

SET ALTERNATE

Purpose	Directs screen or printer output to a text file
Syntax	SET ALTERNATE ON OFF or SET ALTERNATE TO [<file> [ADDITIVE]]
See Also	CLOSE, SET PRINTER

Description	Use SET ALTERNATE to direct output created with ?, ??, DISPLAY or LIST to a text file.
Clauses	ON Include ON to direct output to the text file. OFF Include OFF to disable output to the text file. This is the default setting. TO <file> [ADDITIVE] SET ALTERNATE TO <file> creates the text file, which is assigned the default extension .TXT unless you include a different extension with the file name. The last file you created using SET ALTERNATE is closed if <file> is omitted. Include ADDITIVE to append output to the end of the file specified with <file>. If ADDITIVE is omitted, the contents of the file are over-written.

SET ANSI

Purpose	Specify how SQL string comparisons are made
Syntax	SET ANSI ON OFF
See Also	CREATE QUERY, MODIFY QUERY, SELECT — SQL, SET EXACT

Description SET ANSI determines how comparisons between strings of different lengths are made with the = operator in FoxPro SQL commands. SET ANSI determines whether the shorter of the two strings is padded with blanks when the comparison is made. SET ANSI has no effect on the == operator; the shorter string is always padded with blanks for the comparison when you use the == operator.

Clauses ON

When SET ANSI is ON, the shorter string is padded with the number of blanks needed to equal the length of the longer string. The two strings are then compared character for character for the entire length of the strings. Consider this comparison:

```
'Tommy' = 'Tom'
```

The result is false (.F.) if SET ANSI is ON because when padded, 'Tom' becomes 'Tom ' and the strings 'Tom ' and 'Tommy' don't match character for character.

The == operator uses this method for comparisons in FoxPro SQL commands.

OFF

When SET ANSI is OFF, the shorter string isn't padded with blanks. The two strings are compared character for character until the end of the shorter string is reached. Consider this comparison:

```
'Tommy' = 'Tom'
```

The result is true (.T.) when SET ANSI is OFF because the comparison stops after 'Tom'.

SET ANSI and RQBE

FoxPro builds an SQL SELECT command in the RQBE window when you create a query. When creating join and filter conditions, if you choose the Like or Exactly Like options, the = or == operator is included in the generated SELECT. The ANSI setting can affect the results of queries you create and perform in the RQBE window.

String Order

In SQL commands, the left-to-right order of the two strings in a comparison is irrelevant — switching a string from one side of the = or == operator to the other doesn't affect the result of the comparison.

The following table shows the effect of the ANSI setting on comparisons using the = or == operator.

Comparison	Result with SET ANSI ON	Result with SET ANSI OFF
'Tommy' = 'Tom'	False	True
'Tom Fred' = 'Tom'	False	True
'Tom' = 'Tom '	True	True
'Tom ' = 'Tom'	True	True
'Tom' == 'Tommy'	False	False
'Tom' == 'Tom '	True	True
'Tom ' == 'Tom'	True	True

SET AUTOSAVE

Purpose	Determines whether or not FoxPro flushes data buffers to disk
Syntax	SET AUTOSAVE ON OFF
See Also	FFLUSH(), FLUSH

Description SET AUTOSAVE determines whether or not FoxPro flushes data buffers to disk when you exit a READ or return to the Command window. Flushing buffers may reduce the chance of data loss if your computer loses power.

Clauses **ON**

Buffers are flushed to disk whenever you exit a READ or return to the Command window when SET AUTOSAVE is ON.

OFF

When SET AUTOSAVE is OFF, buffers are flushed only if five minutes have elapsed since the last flush occurred, and only when a READ is exited or you return to the Command window.

The default for SET AUTOSAVE is OFF.

SET BELL

Purpose	Turns the computer bell on or off and sets the bell attributes
Syntax	SET BELL ON OFF or SET BELL TO [<frequency>, <duration>]
Description	SET BELL enables or disables the sounding of the bell during editing when the end of a field is reached or invalid data is entered.
Clauses	ON SET BELL ON turns the bell on. This is the default setting. OFF SET BELL OFF turns the bell off. TO <frequency>,<duration> You can adjust the pitch of the bell and the length of time the bell is sounded with SET BELL TO. Specify the pitch with <frequency>, which can be 19 through 10,000 hertz. The default frequency is 512 hertz. Specify the length of time with <duration>, which can be 1 through 19 seconds. The default duration is 2 seconds.

SET BLINK

Purpose	Determines whether or not you can specify a blinking attribute or a high-density attribute
Syntax	SET BLINK ON OFF
Remarks	FoxPro for Windows only
See Also	SET COLOR commands

Description SET BLINK only affects FoxPro for MS-DOS users with EGA or VGA monitors. If you specify a BLINK setting in your configuration file, it is overridden by the BLINK setting of the color set that loads when you start FoxPro.

SET BLINK is ignored in FoxPro for Windows and on other display hardware.

Clauses **ON**

Screen elements (borders, shadows, text, and so on) can be made to blink if SET BLINK is ON.

The default setting is ON.

OFF

The background colors can be brightened if SET BLINK is OFF, doubling the number of available colors.

For more information on color settings, see the SET COLOR commands in this chapter.

SET BLOCKSIZE

Purpose	Specifies how FoxPro allocates disk space for the storage of memo fields
Syntax	SET BLOCKSIZE TO <expN>
See Also	COPY, MODIFY STRUCTURE, PACK

Description SET BLOCKSIZE specifies how disk space is allocated for the storage of memo fields.

The default value for SET BLOCKSIZE is 64. To reset BLOCKSIZE to a different value after the file has been created, set BLOCKSIZE to a new value, and then use COPY to create a new table/.DBF. The new table/.DBF will have the specified blocksize.

Parameter <expN>

If <expN> is an integer between 1 and 32, disk space for the storage of memo fields is allocated in blocks of <expN> * 512 bytes. If <expN> is greater than 32, disk space is allocated in blocks of <expN> bytes. Specifying a value for BLOCKSIZE greater than 32 can result in a substantial savings in disk space.

SET BORDER

Purpose	Defines a border for popups created with DEFINE POPUP and for windows created with DEFINE WINDOW
Syntax	SET BORDER TO [SINGLE DOUBLE PANEL NONE <border string1> [, <border string2>]]
See Also	@ ... TO, DEFINE POPUP, DEFINE WINDOW

Description Use SET BORDER to specify the borders of boxes, popups and windows. Boxes are created with @ ... TO, popups are created with DEFINE POPUP and windows are created with DEFINE WINDOW. SET BORDER does not affect boxes created with @ ... BOX.

SET BORDER provides compatibility with FoxBASE+ style popups. When you start FoxPro, the default border for FoxBASE+ style popups is a double line. All other objects (windows, popups created with DEFINE POPUP and so on) default to a single-line border. If you issue SET BORDER TO SINGLE or SET BORDER TO DOUBLE, all objects including FoxBASE+ style popups are drawn with a single- or double-line border, respectively.

In FoxPro for Windows, the DOUBLE option creates a PANEL border for popups and windows, and the <border string> option is ignored.

If you issue SET BORDER TO without any additional arguments, the default borders are restored.

Clauses SINGLE

Including SINGLE creates a single-line border. This the default setting.

DOUBLE

Including DOUBLE creates a double-line border.

PANEL

Including PANEL creates a wide border.

NONE

Including NONE suppresses the border entirely.

<border string1> [, <border string2>]

Use the border definition strings <border string1> and <border string2> to create your own borders. <border string1> is used for menus, popups and boxes and for the current output window. <border string2> specifies the border that appears when output isn't being directed to the window. <border string2> doesn't affect boxes or popups.

A border definition string is a set of ASCII character values defined [**<1>**] [, [**<2>**] [, [**<3>**] [, [**<4>**] [, [**<5>**] [, [**<6>**] [, [**<7>**] [, [**<8>**]]]]]]]. The border string entries 1 through 8 designate the top, bottom, left side, right side, upper-left corner, upper-right corner, lower-left corner and lower-right corner of the border, respectively. If you include just one ASCII value in the border string definition, the entire border is drawn using that one character.

If a border string definition is changed after a popup or window has been defined, the original border string is retained. To use the new border string, you must recreate the popup or window.

SET BRSTATUS

Purpose	Enables or disables the display of the status bar in a Browse window
Syntax	SET BRSTATUS ON OFF
Remarks	Included for backward compatibility
See Also	SET STATUS

Description SET BRSTATUS controls the display of the status bar when a Browse window is opened. The status bar displays the current drive, the active table/.DBF, the record pointer position, the number of records in the table/.DBF and the state of the Insert, NumLock, and Caps-Lock keys.

If the status bar is already displayed when you open a Browse window, it remains displayed.

Clauses **ON**

The status bar is displayed at the bottom of the screen when a Browse window is opened if SET BRSTATUS is ON. If more than one Browse window has been opened, the status bar displays information for the active Browse window.

In FoxPro for Windows, opening a Browse window displays the character status bar if the Windows-style status bar is turned off.

OFF

If SET BRSTATUS is OFF, the status bar isn't displayed when a Browse window is opened. This is the default setting.

SET CARRY

Purpose	Determines whether or not FoxPro carries data forward from the current record to the new record created with APPEND or INSERT
Syntax	SET CARRY ON OFF or SET CARRY TO [<field list> [ADDITIVE]]
See Also	APPEND
Description	Use SET CARRY to enable data to be carried forward from the current record to a new record or to prevent data from being carried forward. Fields that generally remain unchanged during an editing session can be brought forward for each new record. For example, a field containing the current date can be brought forward for each new record so a new date doesn't have to be entered again.
Clauses	ON SET CARRY ON brings data from all fields in all work areas forward from the current record to a new record. OFF SET CARRY OFF prevents data from all fields from being passed to a new record. This is the default setting. TO <field list> [ADDITIVE] By default, data from all fields is carried forward to a new record. Use SET CARRY TO <field list> to specify from which field data is carried forward. Data from only the fields specified in the field list are brought forward. If ADDITIVE is included, the fields in the field list are added to the current set of fields being carried forward. SET CARRY TO affects the current work area. Issuing SET CARRY TO implicitly performs SET CARRY ON. Use SET CARRY TO without <field list> to restore the default setting (all fields are carried forward).

SET CENTURY

Purpose	Determines whether or not FoxPro displays the century portion of date expressions
Syntax	SET CENTURY ON OFF
See Also	DATE(), YEAR()

Description Use SET CENTURY to specify how date variables and functions are displayed.

Clauses **ON**

SET CENTURY ON specifies a four-digit year format that includes 10 characters (including date delimiters).

OFF

SET CENTURY OFF specifies a two-digit year format that includes eight characters and assumes the twentieth century for date calculations. This is the default setting.

SET CLEAR

Purpose	Determines whether or not issuing SET FORMAT and QUIT clears the screen
Syntax	SET CLEAR ON OFF
See Also	CLEAR

Description SET CLEAR determines whether the desktop in FoxPro for MS-DOS or the main FoxPro window in FoxPro for Windows is cleared when SET FORMAT is issued.

Clauses **ON**

If SET CLEAR is ON, the desktop or main FoxPro window isn't cleared when you open a format file with SET FORMAT. This is the default setting.

OFF

The desktop or main FoxPro window is cleared if SET CLEAR is OFF.

In FoxPro for MS-DOS, if SET CLEAR is OFF, the desktop isn't cleared when you exit FoxPro. You can display a sign-off screen when you quit FoxPro.

SET CLOCK

Purpose	Determines whether or not FoxPro displays the system clock and specifies the clock location on the screen
Syntax	SET CLOCK ON OFF STATUS or SET CLOCK TO [<row, column>]
Remarks	STATUS clause — FoxPro for Windows only
See Also	SET BRSTATUS, SET HOURS, SET STATUS

Description Use SET CLOCK to display the system clock in a specified location.

Clauses **ON**
SET CLOCK ON displays the clock in the upper-right corner of the desktop in FoxPro for MS-DOS or the main FoxPro window in FoxPro for Windows.

OFF
SET CLOCK OFF removes the clock from the desktop, the main FoxPro window or the Windows-style status bar. This is the default setting.

STATUS — FoxPro for Windows only
In FoxPro for Windows, issue SET CLOCK STATUS to display the clock in the Windows-style status bar. Issue SET STATUS BAR ON to display the Windows-style status bar.

TO <row, column>
You can designate where the clock is displayed on the desktop or main FoxPro window by including the coordinates <row, column>. Use SET CLOCK TO without the coordinates to display the clock in its default position in the upper-right corner of the desktop or main FoxPro window.

In FoxPro for Windows, if the clock is placed in the Windows-style status bar with SET CLOCK STATUS and you specify a location in the main FoxPro window with TO <row, column>, the clock is removed from the status bar and is placed at the location you specify.

SET COLOR Commands

FoxPro offers a sophisticated set of commands for full control of colors.

In FoxPro for MS-DOS, you can define the colors of elements of the FoxPro system interface (the system menu bar, windows, alerts, dialogs, and so on) and user-defined objects (controls, menus, windows, and so on). You can set colors directly with the SET COLOR commands or interactively with the Color Picker dialog, which is available by choosing **Color...** from the **Window** menu. The Color Picker dialog provides the fastest way to set colors and allows you to save color sets for later use. See the FoxPro *User's Guide* for more information on setting colors with the Color Picker dialog.

By default, FoxPro for Windows takes its colors from the Windows Control Panel settings. The Windows colors are mapped to the FoxPro for Windows color schemes. The colors can be overwritten using the FoxPro SET COLOR commands described in this section.

In FoxPro for Windows, you can set colors directly with the SET COLOR commands or interactively in the Windows Control Panel. Refer to your Windows documentation for more information about setting colors in Windows.

Certain commands, like DEFINE WINDOW, support optional COLOR and COLOR SCHEME clauses that let you specify colors of user-defined objects directly. The following terms are used in the descriptions of the SET COLOR commands.

Color Pair

A *color pair* consists of codes that specify the foreground and background colors. The following table lists the colors that are available and their codes:

Color	Code	Color	Code
Black	N	Inverse	I
Blank	X	Magenta	RB
Blue	B	Red	R
Brown	GR	White	W
Cyan	BG	Yellow	GR+
Green	G	Underlined	U

An asterisk (*) can be used to denote blinking or bright (depending on your video hardware and the BLINK setting) for the background color. In FoxPro for Windows, including an asterisk always makes the background bright. A plus sign (+) denotes high intensity for the foreground color.

On monochrome monitors, only four colors are available: white (W), black (N), underlined (U) and inverse video (I). The blank (X) color is useful for entering passwords.

In FoxPro for Windows, a color pair can also be specified with a set of 6 RGB (Red Green Blue) color values separated by commas. The first 3 color values specify the foreground color and the second 3 color values specify the background color. The color values can range from 0 through 255.

In FoxPro for Windows, the R/W color pair in the example above can also be specified with this RGB color pair:

```
RGB(255,0,0,255,255,255)
```

Color Pair List

A *color pair list* consists of one to ten color pairs separated by commas. For example, in FoxPro for MS-DOS:

```
W+/B, W+/BG, GR+/B, GR+/B, R+/B, W+/GR, GR+/RB, N+/N, GR+/B, R+/B
```

A color pair list for a monochrome monitor might look like this:

```
W/N, N+/W, W+/N, W+/N, W/N, U+/N, W+/N, -, W+/N, W/N
```

Color pair lists for monochrome monitors may have a dash (-) in the eighth color pair location to indicate that a shadow isn't present.

In FoxPro for Windows, a color pair list looks like this:

```
RGB(0,255,0,255,0,0), RGB(127,255,0,0,0,0), ...
```

Color Scheme

A *color scheme* is a set of 10 color pairs. In FoxPro for MS-DOS, color schemes control the colors of interface elements such as system windows, user-defined windows, menus, etc. Each color pair corresponds to an item in a color scheme. Use `SCHEME( )` to return the color pairs for a color scheme.

When FoxPro for MS-DOS is first started, a default set of color schemes is loaded. You can specify your own startup color schemes in your FoxPro configuration file. Include the following line for each color scheme you want to change.

```
COLOR OF SCHEME <expN> = <color pair list>
```

When FoxPro for Windows is started the Window Control Panel color settings are loaded into the FoxPro color schemes. You can specify your own colors. Not all FoxPro for Windows interface elements can be controlled by color schemes — FoxPro for Windows system windows (View and Command Windows, the system menu bar, etc.) are always controlled by the Windows Control Panel color settings.

Color Set

A *color set* consists of 24 color schemes. You can save the complete color environment in a color set. Color sets, like keyboard macros and memory variables, can be saved for later use. You can assign to a color set a name up to 10 characters long. Color sets are stored in the `FOXUSER.DBF` resource file.

You can load a color set with `SET COLOR SET TO`. In FoxPro for MS-DOS, you can also load a color set with the Color Picker dialog.

When FoxPro is first started, a default color set is loaded unless a different color set is specified in your FoxPro configuration file. To specify a startup color set in your configuration file, include the following line:

```
COLOR SET = <colorsetname>
```


SET COLOR OF

Purpose	Specifies the colors of user-defined menu systems and windows
Syntax	SET COLOR OF NORMAL MESSAGES TITLES BOX HIGHLIGHT INFORMATION FIELDS TO [<standard>]
Remarks	Included for backward compatibility — use SET COLOR OF SCHEME
See Also	SET COLOR OF SCHEME

Description SET COLOR OF affects the color settings of color schemes 1 and 2. With this command, you can set selected color attributes of user-defined windows and selected color attributes of user-defined menus.

SET COLOR OF SCHEME

Purpose	Specifies the colors of a color scheme or copies one color scheme to another color scheme
Syntax	SET COLOR OF SCHEME <expN1>TO [SCHEME <expN2> <color pair list>]
Remarks	RGB color pairs — FoxPro for Windows only
See Also	CREATE COLOR SET, SET COLOR SET

Description With SET COLOR OF SCHEME, you can set the colors of a color scheme by specifying another color scheme or a color pair list.

In FoxPro for MS-DOS, if you issue SET COLOR OF SCHEME <expN1> TO without including an optional clause, color scheme <expN1> derives its colors from the last named color set in this order: the last color set loaded or saved, the last color set defined with SET COLOR SET TO, the last color set loaded from CONFIG.FP or the FoxPro default color set. If there is no last named color set, the hardware default is restored.

In FoxPro for Windows, not all interface elements can be controlled by color schemes — FoxPro for Windows system windows (View and Command Windows, the system menu bar, etc.) are always controlled by the Windows Control Panel color settings. Issuing SET COLOR OF SCHEME <expN1> TO without including another color scheme or a color pair list restores the colors from the current Windows color scheme.

Clauses **<expN1>**
<expN1> is the number of the color scheme you want to change. This can be a value from 1 through 24.

TO <color pair list>
If you include a color pair list, you can specify up to 10 color pairs you would like to change. You can selectively change colors in a color scheme by including a comma for each color pair that you don't want to change. For example, to change the third color pair in color scheme 1 to white and red and leave the rest of the color settings unchanged, use this command:

SET COLOR OF SCHEME 1 TO , , W/R

In FoxPro for Windows, a color pair can also be specified with a set of 6 RGB (Red Green Blue) color values separated by commas. To change the third color pair in color scheme 1 to white and red and leave the rest of the color settings unchanged as in the example above, you can use this command:

```
SET COLOR OF SCHEME 1 TO , , RGB(255,255,255,0,0,255)
```

SCHEME <expN2>

Set the colors of a color scheme to those of another color scheme with TO SCHEME <expN2>. <expN2> is the number of the color scheme you want to copy.

SET COLOR SET

Purpose	Loads a previously defined color set
Syntax	SET COLOR SET TO [<color set name>]
See Also	CREATE COLOR SET, SET COLOR OF SCHEME

Description SET COLOR SET loads colors from a previously defined color set. Color sets can be created with SET COLOR OF SCHEME and saved with CREATE COLOR SET.

In FoxPro for MS-DOS, you can also create a color set with the Color Picker dialog by choosing **Color...** from the **Window** menu. After you create a color set, be sure to save it in the Color Picker or with CREATE COLOR SET.

Clause **<color set name>**

In FoxPro for MS-DOS, if <color set name> isn't included, FoxPro loads the last named color set. The last named color set is the last color set loaded or saved, the last color set defined with SET COLOR SET TO, the last color set loaded from your configuration file on start-up, or the FoxPro default color set. If there is no last named color set, the hardware default is restored.

In FoxPro for Windows, if <color set name> isn't included, the current color set remains loaded.

SET COLOR TO

Purpose	Specifies the colors of user-defined menus and windows
Syntax	SET COLOR TO [[<standard>] [, [<enhanced>] [, [<border>]]]]
Remarks	Included for backward compatibility. Use SET COLOR OF SCHEME instead.
See Also	CREATE COLOR SET, SET COLOR OF SCHEME, SET COLOR SET

Description SET COLOR TO affects multiple color pairs of color scheme number 1 and color scheme number 2. Use SET COLOR OF SCHEME for maximum control over the color environment.

SET COMPATIBLE

Purpose	Controls compatibility with FoxBASE+ and other Xbase languages
Syntax	SET COMPATIBLE FOXPLUS DB4 ON OFF
Description	SET COMPATIBLE controls the compatibility of programs created for use with FoxBASE+ and other Xbase languages. Commands and functions affected by SET COMPATIBLE include LIKE(), PLAY MACRO, SELECT() and STORE (when STORE is used with arrays). SET COMPATIBLE doesn't create compatibility with other Xbase commands, functions or features that are not supported in FoxPro. For example, it doesn't allow you to open a report form created in other Xbase products with the FoxPro Report Writer. A complete listing of commands that are affected by SET COMPATIBLE is included the following table.
Clauses	FOXPLUS OFF FOXPLUS and OFF can be used interchangeably. Include either key word to allow programs created in FoxBASE+ to run in FoxPro for MS-DOS or FoxPro for Windows without changes. This is the default setting. DB4 ON Including DB4 or ON affects the behavior of commands and functions listed below.

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 is searched when FoxPro looks for a file.
	FOXPLUS (OFF)	If a path is set and the command specifies a drive, the specified drive is searched, then the FoxPro path.
	An example: <pre>SET PATH TO D:\TESTDIR DO C:TEST</pre> In this example, TEST is a program file located in D:\TESTDIR. It won't be found and executed if SET COMPATIBLE is DB4, because only the specified drive is searched. If SET COMPATIBLE is FOXPLUS, however, the file will be found and executed because the FoxPro path will also be searched.	
Parameters passed by reference	DB4 (ON)	Parameters that are passed by reference (with SET UDFPARMS ON) remain available to the called procedure.
	FOXPLUS (OFF)	Parameters that are passed by reference (with SET UDFPARMS ON) become "hidden" to the called procedure.
@...GET...RANGE	DB4 (ON)	The range is always checked.
	FOXPLUS (OFF)	The range is checked only if data has been changed.
@...SAY (special characters)	DB4 (ON)	All special characters are output except CHR(7), which rings the bell.
	FOXPLUS (OFF)	All special characters are output including CHR(7), which doesn't ring the bell.
@...SAY (scrolling)	DB4 (ON)	Output that extends beyond the lower-right corner of the desktop or the main FoxPro window is displayed, causing the desktop or main FoxPro window to scroll upward.
	FOXPLUS (OFF)	Output that extends beyond the lower-right corner of the desktop or main FoxPro window is truncated.

Command or Function	SET COMPATIBLE	Effect
@...SAY (with SET STATUS ON)	DB4 (ON)	Output can overwrite the status bar. Text that extends beyond the end of the status bar display wraps above the status bar, scrolling upward from that point.
	FOXPLUS (OFF)	Output can not overwrite the status bar. Text that extends beyond the end of the desktop or main FoxPro window is truncated.
@...SAY ... PICTURE	DB4 (ON)	When numeric data is displayed with a PICTURE clause, the rightmost digit in the PICTURE clause is rounded.
	FOXPLUS (OFF)	When numeric data is displayed with a PICTURE clause, the rightmost digit isn't rounded — extra digits are truncated.
ACTIVATE SCREEN and ACTIVATE WINDOW	DB4 (ON)	When the desktop, main FoxPro window or a user-defined window is activated, the cursor position is set to 0,0.
	FOXPLUS (OFF)	The cursor remains at its current position.
APPEND MEMO	DB4 (ON)	The default extension for the file is .TXT if one isn't specified.
	FOXPLUS (OFF)	There is no default extension.
GO GOTO (with TALK ON)	DB4 (ON)	FoxPro outputs a message that identifies the current work area alias and the record number of the record pointer position.
	FOXPLUS (OFF)	FoxPro doesn't output the message indicating position.
MENU and POPUP commands	DB4 (ON)	Popups are placed in the active output window and, once they are activated, the cursor is positioned on an option in the popup. If the popup is placed on row 0, the entire row is used as part of the menu bar.
	FOXPLUS (OFF)	Popups are placed in their own windows and the cursor isn't moved from its position in the active output window. If the popup is placed on row 0, only the pads of the menu are treated as part of the menu bar.

Command or Function	SET COMPATIBLE	Effect
PLAY MACRO	DB4 (ON)	When a keyboard macro in the range of A-Z is played (for example, PLAY MACRO A), an implicit Alt+F10 is added before the letter and FoxPro executes the macro associated with the Alt+F10+letter combination (for example, 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 key combination (for example, Alt+F1).
	FOXPLUS (OFF)	FoxPro executes the macro associated with the given name without adding any implicit keystrokes.
READ (with a VALID clause on a GET)	DB4 (ON)	If Escape is pressed when the cursor is in a field that has a VALID clause, validation is performed. If the input is invalid, the "Invalid Input" alert is displayed.
	FOXPLUS (OFF)	Validation isn't performed if Escape is pressed. If validation is performed (Escape isn't pressed), the "Invalid Input" alert isn't displayed if input is invalid.
READ and TRANSFORM (with a numeric PICTURE clause)	DB4 (ON)	The value of the variable or expression is rounded to the number of decimal places specified in the PICTURE clause.
	FOXPLUS (OFF)	The value of the variable or expression is truncated to fit the PICTURE clause.
Nested READs	DB4 (ON)	FoxPro performs an implicit CLEAR GETs on the lower level before returning to a higher level READ.
	FOXPLUS (OFF)	When FoxPro returns to a higher level READ, pending GETs in the lower level remain pending.
RUN/!	DB4 (ON)	The cursor moves to the first column in the last row (24). All subsequent output is displayed from this point.
	FOXPLUS (OFF)	The cursor remains at its current position. All subsequent output is displayed from this point.

Command or Function	SET COMPATIBLE	Effect
RUN! (with SET STATUS ON)	DB4 (ON)	Output from the program that was run is scrolled up three lines before control returns to FoxPro.
	FOXPLUS (OFF)	Output from the program that was run is scrolled up two lines before control returns to FoxPro.
SET COLOR TO	DB4 (ON)	If SET STATUS is ON, the last line in the desktop or main FoxPro window and the second to the last line are redrawn with the new SET COLOR TO colors.
	FOXPLUS (OFF)	If SET STATUS is ON, the last three lines on the desktop or main FoxPro window are all redrawn with the new colors.
SET FIELDS	DB4 (ON)	Using SET FIELDS TO without a field list or the ALL clause changes SET FIELDS to OFF and removes all fields from the field list.
	FOXPLUS (OFF)	Using SET FIELDS TO without a field list or the ALL clause changes SET FIELDS to the null string and removes all fields from the field list.
SET MESSAGE	DB4 (ON)	The character expression specified in this command appears immediately on the last line of the screen and the text is output in a different color.
	FOXPLUS (OFF)	The character expression is displayed only when SET STATUS is ON, and both the character expression and the line it is on are displayed in a different color.
SET PRINT TO <file>	DB4 (ON)	The output file is given a .PRT extension unless another extension is specified.
	FOXPLUS (OFF)	The output file is given an extension only when one is specified.
STORE	DB4 (ON)	STORE cannot be used to initialize all elements of an array.
	FOXPLUS (OFF)	STORE can be used to initialize all elements of an array to a specified value.

Command or Function	SET COMPATIBLE	Effect
SUM	DB4 (ON)	The number specified in SET DECIMALS is the number of decimal places that are output by SUM.
	FOXPLUS (OFF)	The number of decimal places (specified in the database structure) for the field being summed determines the number of decimal places that are output.
INKEY() and LASTKEY()	DB4 (ON)	Home and Shift+Home return 26 and Ctrl+Left returns 1.
	FOXPLUS (OFF)	Refer to the table in INKEY() in this chapter for values.
LIKE()	DB4 (ON)	Trailing blanks in both the pattern and target are trimmed before the comparison is made.
	FOXPLUS (OFF)	The pattern and target are used as is, and trailing blanks are significant.
SELECT()	DB4 (ON)	SELECT() returns the number of the unused work area with the highest number.
	FOXPLUS (OFF)	SELECT() returns the number of the currently selected work area.
SYS(2001,'COLOR')	DB4 (ON)	The value returned is the current setting of SET COLOR.
	FOXPLUS (OFF)	The value returned is the number for the SET COLOR TO color pairs.

SET CONFIRM

Purpose	Specifies whether or not Enter or Tab must be pressed to exit an input field
Syntax	SET CONFIRM ON OFF
See Also	@ ... GET

Description When SET CONFIRM is OFF (the default value) and the cursor reaches the last character in an @ ... GET input field, the cursor moves to the next field and the bell sounds (if SET BELL is ON).

If SET CONFIRM is ON, you must press Enter or Tab to move to the next object.

SET CONFIRM also affects menu bars and popups created with DEFINE MENU and DEFINE POPUP. If SET CONFIRM is OFF, you can choose an option from a popup or a pad from a menu bar by pressing the key corresponding to the first letter of the option.

If SET CONFIRM is ON, typing the first letter of the option or choosing a pad selects that option or pad. To choose the option or pad, you must press Enter or the Spacebar.

If a popup option or menu bar pad is defined with a hot key, the option or pad can be chosen by pressing the corresponding hot key. SET CONFIRM has no effect on hot keys.

The default value of SET CONFIRM is OFF.

SET CONSOLE

Purpose	Enables or disables output to the screen or a window
Syntax	SET CONSOLE ON OFF
See Also	SYS(100)

Description SET CONSOLE enables or disables output to the desktop in FoxPro for MS-DOS, the main FoxPro window in FoxPro for Windows or the active user-defined window from within programs. All output is sent to the desktop, the main FoxPro window or the active user-defined window if SET CONSOLE is ON.

Output to the desktop, the main FoxPro window or the active user-defined window is suppressed if SET CONSOLE is OFF.

SET CONSOLE is ON when you are using FoxPro interactively and cannot be changed to OFF in the Command window. You can change SET CONSOLE to OFF only from within a program.

The CONSOLE setting affects some interactive FoxPro dialogs. For example, if SET CONSOLE is OFF and you issue BROWSE when no table/.DBF is open, an error message is displayed. The Open dialog is displayed if SET CONSOLE is ON.

SET CONSOLE doesn't affect output from @ ... SAY. Output from @ ... SAY is controlled by the DEVICE setting.



An error always sets CONSOLE to ON. Use SYS(100) to check the CONSOLE setting before the error occurred. See SYS(100) in this chapter for additional information about SYS(100).

The default value of SET CONSOLE is ON.

SET CURRENCY

Purpose	Defines the currency symbol and specifies its position in the display of numeric expressions
Syntax	SET CURRENCY TO [<expC>] or SET CURRENCY LEFT RIGHT
See Also	SET DECIMALS, SET SEPARATOR
Description	SET CURRENCY defines the currency symbol and specifies its position in the display of numeric expressions using the \$ PICTURE or FUNCTION code.
Clauses	<expC> <expC> is a character string representing the currency symbol and can be one to nine characters long. Issue SET CURRENCY TO without <expC> to reset the currency symbol to the default dollar sign (\$). LEFT RIGHT Use LEFT or RIGHT to position the currency symbol to the left or right of the currency value. SET CURRENCY LEFT places the currency symbol to the left of the currency value, and SET CURRENCY RIGHT places the symbol to the right of the currency value. The default is LEFT.

Example The following example demonstrates how you can display two currency symbols (DM and \$), one on either side of the currency value. If you use PICTURE to display the currency symbol, be sure to include @ before the dollar sign.

```

STORE SET('CURRENCY') TO currposit
STORE 1234.56 TO dollaramnt
CLEAR

SET CURRENCY TO 'DM'
@ 2,2 SAY dollaramnt PICTURE '@$99,999.99'

IF currposit = 'LEFT'
    SET CURRENCY RIGHT
ELSE
    SET CURRENCY LEFT
ENDIF
@ 4,2 SAY dollaramnt FUNCTION '$99,999.99'

DM1234.56

1234.56DM

```

SET CURSOR

Purpose	Determines whether the cursor is displayed during a pending @ ... GET
Syntax	SET CURSOR ON OFF
See Also	@ ... GET, INKEY(), SET(), SYS(2002), WAIT

Description You can turn the cursor on or off with SET CURSOR. SET CURSOR is similar to SYS(2002).

Clauses **ON**

SET CURSOR ON causes the cursor to be displayed during a pending @ ... GET, @ ... EDIT, WAIT, or INKEY() function. This is the default setting.

OFF

SET CURSOR OFF prevents the cursor from being displayed during a pending @ ... GET, @ ... EDIT, WAIT or INKEY().

SET DATE

Purpose	Specifies the format for the display of date expressions
Syntax	SET DATE [TO] AMERICAN ANSI BRITISH FRENCH GERMAN ITALIAN JAPAN USA MDY DMY YMD
See Also	DATE(), SET CENTURY, SET MARK

Description Use this command to set the format used to display date expressions. Here are the settings and the resulting date formats:

Setting	Format
AMERICAN	mm/dd/yy
ANSI	yy.mm.dd
BRITISH/FRENCH	dd/mm/yy
GERMAN	dd.mm.yy
ITALIAN	dd-mm-yy
JAPAN	yy/mm/dd
USA	mm-dd-yy
MDY	mm/dd/yy
DMY	dd/mm/yy
YMD	yy/mm/dd

The default date setting is AMERICAN.

SET DEBUG

Purpose	Enables or disables access to the Debug and Trace windows
Syntax	SET DEBUG ON OFF
See Also	SET ECHO

Description SET DEBUG enables or disables menu access to the Debug and Trace windows.

For more information on the Trace and Debug windows, see the Debugging Your Application chapter in the FoxPro *Developer's Guide*.

Clauses **ON**

If SET DEBUG is ON, you can open both the Debug and Trace windows using the FoxPro system menu. ON is the default setting.

OFF

If SET DEBUG is OFF, neither window can be opened from the FoxPro system menu. However, when SET DEBUG is OFF, the Debug window can be opened with SET ECHO ON or ACTIVATE WINDOW DEBUG, and the Trace window can be opened with SET STEP ON or ACTIVATE WINDOW TRACE.

SET DECIMALS

Purpose	Specifies the number of decimal places displayed in numeric expressions
----------------	---

Syntax	SET DECIMALS TO [<expN>]
---------------	---------------------------------------

See Also	SET FIXED
-----------------	-----------

Description SET DECIMALS specifies the minimum number of decimal places used to display the results of division, multiplication and trigonometric and financial functions.

Clause <expN>
Specify the minimum number of decimal places to display with <expN>. The default is two decimal places. The maximum number of decimals places is 18, and the minimum is 0.

SET DEFAULT

Purpose	Specifies the drive and directory
Syntax	SET DEFAULT TO [<expC>]
See Also	SYS(5), SYS(2003)

Description This command instructs FoxPro to change the default FoxPro directory to the directory you specify.

FoxPro searches for a file in the default FoxPro directory. If you create a file and don't specify where to place the file, the file is placed in the default FoxPro directory.

When you start FoxPro, the default directory is the directory from which you start FoxPro. However, you can specify a different default directory in your FoxPro configuration file or a startup program.

When you quit FoxPro for MS-DOS, you are placed on the drive from which you started FoxPro and in the last directory specified in <expC> if the directory is on the startup drive. The default drive is the drive from which FoxPro is started.

When you quit FoxPro for Windows, you are returned to Windows. When you exit Windows, you are returned to the drive and directory that you started Windows from.



SYS(5) returns the default drive. SYS(2003) returns the default directory with no drive designator. SYS(5) + SYS(2003) returns the default drive and directory.

Clause

<expC>

<expC> can be one of the following: a drive designator, a drive designator with a directory name, a child directory name, or MS-DOS shorthand notation (\ or ..).

Examples You can change the default drive to the A drive using either of the following commands:

```
SET DEFAULT TO A
SET DEFAULT TO A:
```

You can also specify a directory:

```
SET DEFAULT TO A:\sales
SET DEFAULT TO C:\sales\data
```

You can specify a child directory. If the root directory on the C drive is the default FoxPro directory, issue this command to change the default directory to C:\SALES:

```
SET DEFAULT TO sales
```

You can use MS-DOS shorthand notation. If the current directory is C:\SALES\DATA, issue the following command to make the root directory the default directory:

```
SET DEFAULT TO \
```

You can also move the default directory one directory toward the root directory with this command:

```
SET DEFAULT TO ..
```

To restore the FoxPro startup directory, issue SET DEFAULT without any arguments:

```
SET DEFAULT TO
```

SET DELETED

Purpose	Specifies whether or not FoxPro processes records marked for deletion
Syntax	SET DELETED ON OFF
See Also	DELETE, DELETED(), PACK, RECALL

Description SET DELETED determines whether records marked for deletion are available for use in other commands.



SET DELETED is ignored if the default scope for the command is the current record or you include a scope of a single record. INDEX and REINDEX always ignore SET DELETED and index all records in the table/.DBF.

Queries that use DELETED() to test the status of records can be optimized using Rushmore technology if the table/.DBF is indexed on DELETED(). See the Optimizing Your Application chapter in the FoxPro *Developer's Guide* for more information on optimizing queries.

You can mark records for deletion by issuing DELETE from within a Browse, Change or Edit window or by choosing **Delete...** from the **Record** menu.

You can recall records by issuing RECALL or by choosing **Recall...** from the **Record** menu.

Clauses **ON**

All commands that select records (including records in related tables/.DBFs) using a scope ignore records marked for deletion if SET DELETED is ON.

OFF

When SET DELETED is OFF, records marked for deletion are unavailable for use in other commands.

The default for SET DELETED is OFF.

SET DELIMITERS

Purpose	Specifies whether or not @ ... GET input fields are delimited
Syntax	SET DELIMITERS ON OFF or SET DELIMITERS TO <expC> TO DEFAULT
Remarks	Included for backward compatibility
See Also	@ ... GET

Description SET DELIMITERS determines whether @ ... GET input fields are delimited.

Clauses **ON**

Issuing SET DELIMITERS ON displays delimiters in @ ... GET fields. The default delimiting characters are colons (:). A colon is placed on the left and right side of the field to indicate where to enter data.

OFF

If SET DELIMITERS is OFF, no delimiters are displayed in fields created with @ ... GET. This is the default setting.

TO <expC>

Specify one or two delimiting characters with <expC>. Fields are delimited on both sides by the same character if <expC> includes one character. If <expC> includes two characters, the first character is the left delimiter and the second character is the right delimiter.

TO DEFAULT

Use SET DELIMITERS TO DEFAULT to reset the delimiters to the default colons.

SET DEVELOPMENT

Purpose	Causes FoxPro to compare the creation date and time of a program with those of its compiled object file when the program is run
Syntax	SET DEVELOPMENT ON OFF
See Also	COMPILE, MODIFY COMMAND

Description SET DEVELOPMENT compares the creation date and time of a source program with those of its corresponding compiled object file. FoxPro recompiles the source program before it executes if SET DEVELOPMENT is ON and the source program is more current than its compiled object program. This ensures that the most current version of a program is executed.

The source and compiled versions of the program aren't compared if SET DEVELOPMENT is OFF. You may not always be executing the most current version of a program if SET DEVELOPMENT is OFF.

The most current version of a program modified with the FoxPro editor invoked with MODIFY COMMAND is always executed, regardless of the DEVELOPMENT setting.

SET DEVELOPMENT needs to be ON only when programs are modified outside of FoxPro. Using an external editor (for example, a TSR editor) may require you to issue CLEAR PROGRAM before you execute the modified program. See CLEAR PROGRAM in this chapter for additional information. Use SET DEVELOPMENT OFF for optimum performance.

When SET DEVELOPMENT is ON, program execution can be canceled during a READ. The **Cancel** option on the **Program** menu is enabled when SET DEVELOPMENT is ON and a READ is active. Choosing **Cancel** during the READ cancels program execution. If SET DEVELOPMENT is OFF, the **Cancel** option on the **Program** menu is disabled during a READ.

The default value of SET DEVELOPMENT is ON.

SET DEVICE

Purpose	Directs output of @ ... SAYs to the screen, a printer, or a file
Syntax	SET DEVICE TO SCREEN TO PRINTER [PROMPT] TO FILE <file>
Remarks	PROMPT clause — FoxPro for Windows only
See Also	SYS(101)

Description SET DEVICE specifies where output from an @ ... SAY is directed. @ ... SAY output can be sent to the desktop in FoxPro for MS-DOS, the main FoxPro window in FoxPro for Windows, the active user-defined window, the printer or a file.

Clauses TO SCREEN

Including TO SCREEN directs @ ... SAY output to the desktop, the main FoxPro window or the active user-defined window.

TO PRINTER [PROMPT]

Including TO PRINTER directs @ ... SAY output to the printer. A page eject is issued when the coordinates in @ ... SAY specify a location on the page higher than the location specified by the coordinates in the previous @ ... SAY.

In FoxPro for Windows, you can include the optional PROMPT clause to display a dialog before printing starts. Place the PROMPT key word immediately after TO PRINTER. PROMPT is ignored in FoxPro for MS-DOS.

In this dialog you can adjust printer settings. You can specify the number of copies and page numbers to print and direct print output to a file. The Windows Printer Setup dialog can be opened to adjust additional printer settings. The printer settings you can adjust depend on the currently installed printer driver.

TO FILE <file>

Including TO FILE <file> sends @ ... SAY output to the file specified with <file>.

SET DISPLAY

Purpose	Enables you to change the current display mode on monitors that support different modes
Syntax	SET DISPLAY TO CGA COLOR EGA25 EGA43 MONO VGA25 VGA50
See Also	SET MESSAGE, SYS(2006)

Description In FoxPro for MS-DOS, SET DISPLAY enables you to switch between different display modes on monitors that support different modes.

In FoxPro for Windows, SET DISPLAY changes the font for the main FoxPro window to 7 or 9 point FoxFont. The size of the main FoxPro window is increased if necessary to accommodate the number of lines for the option you specify. If the Windows-style status bar is displayed when you issue SET DISPLAY, it is turned off.

The error message “Display mode not available” is displayed if an option isn’t supported by your video hardware. The SET MESSAGE line is reset to the last line of the desktop (FoxPro for MS-DOS) or the main FoxPro window whenever SET DISPLAY is issued.

Clauses **CGA**

In FoxPro for MS-DOS, switches display mode to CGA.

In FoxPro for Windows, switches the main FoxPro window font to 9 point FoxFont.

COLOR

In FoxPro for MS-DOS, switches display mode to color.

EGA25

In FoxPro for MS-DOS, switches display mode to EGA and 25 lines.

In FoxPro for Windows, switches the main FoxPro window font to 7 point FoxFont and the main FoxPro window to 25 lines.

EGA43

In FoxPro for MS-DOS, switches display mode to EGA and 43 lines.

In FoxPro for Windows, switches the main FoxPro window font to 9 point FoxFont and the main FoxPro window to 50 lines.

MONO

In FoxPro for MS-DOS, switches display mode to monochrome.

VGA25

In FoxPro for MS-DOS, switches display mode to VGA and 25 lines.

In FoxPro for Windows, switches the main FoxPro window font to 7 point FoxFont and the main FoxPro window to 25 lines.

VGA50

In FoxPro for MS-DOS, switches display mode to VGA and 50 lines.

In FoxPro for Windows, switches the main FoxPro window font to 9 point FoxFont and the main FoxPro window to 50 lines.

SET DOHISTORY

Purpose	Determines whether or not commands from a program are placed into the Command window
Syntax	SET DOHISTORY ON OFF
Remarks	Included for backward compatibility. Use the Trace window instead.
See Also	SET ECHO, SET STEP

Description Use SET DOHISTORY to place commands from a program into the Command window as the program executes. The commands can then be edited and re-executed as if they had been entered directly into the Command window.



Programs execute many times slower when SET DOHISTORY is ON. DOHISTORY constructs a temporary file as the program executes since the Command window is a special type of FoxPro editor session. This file can fill up even a large disk *very quickly*.

Use SET DOHISTORY only as a debugging aid to isolate particularly stubborn bugs. Be sure to remove any SET DOHISTORY ON commands from your programs before you execute or distribute them.

Clauses **ON**

SET DOHISTORY ON places commands from a program into the Command window.

OFF

SET DOHISTORY OFF doesn't place commands from a program into the Command window. This is the default setting.

SET ECHO

Purpose	Activates the Trace window for program debugging
Syntax	SET ECHO ON OFF
See Also	RESUME, SET STEP, SUSPEND

Description SET ECHO opens the Trace window for debugging. You can also set breakpoints within a Trace window to suspend program execution. For more information about the Trace window, see the Debugging Your Applications chapter in the FoxPro *Developer's Guide*.

Clauses **ON**

When SET ECHO is ON, the source code for the program that is executing is displayed in the Trace window. The line currently executing is highlighted.

OFF

SET ECHO OFF closes the Trace window in FoxPro versions earlier than 2.0. From within a program, the Trace window can be closed with DEACTIVATE WINDOW TRACE.

The default value of SET ECHO is OFF.

SET ESCAPE

Purpose	Determines whether or not pressing the Escape key interrupts program and command execution
Syntax	SET ESCAPE ON OFF
See Also	ON ESCAPE, ON KEY LABEL

Description Use SET ESCAPE to control the effect of pressing the Escape key on program and command execution.

Clauses **ON**

SET ESCAPE ON enables command execution to be interrupted by pressing the Escape key.

If you press Escape during the execution of a command while the cursor is in the Command window, the following message is displayed:

*** INTERRUPTED ***

If you press Escape during program execution, processing completes at the current program line, and an alert appears with three options:

- Choose **Cancel** to immediately stop program execution and return to the Command window. This is the default choice.
- Chose **Suspend** to pause program execution and return to the Command window. This option is useful for debugging a program. Choose **Resume** from the **Program** menu or enter RESUME in the Command window to restart the program at the line at which it was paused.
- Choose **Ignore** to continue program execution at the line at which it was paused.

The default for SET ESCAPE is ON.

OFF

If SET ESCAPE is OFF, pressing the Escape key doesn't interrupt command and program execution.

SET EXACT

Purpose	Specifies the rules used when comparing strings of different lengths
Syntax	SET EXACT ON OFF
See Also	SET NEAR

Description SET EXACT specifies the rules used when comparing two strings with different lengths. The EXACT setting has no effect if both strings are the same length.

Clauses **ON**

If SET EXACT is ON, expressions must match character for character for the expressions to be equivalent. Any trailing blanks in the expressions are ignored for the comparison. For the comparison, the shorter of the two expressions is padded on the right with blanks to match the length of the longer expression.

OFF

If SET EXACT is OFF, the expressions are considered equivalent if they match character for character until the end of the expression on the right side is reached. The default value of SET EXACT is OFF.

String Comparisons

FoxPro has two relational operators that test for equality.

The = operator performs a comparison between two values of the same type. This operator is suited for comparing character, numeric, date and logical data.

However, when you compare character expressions with the = operator, the results may not be exactly what you expect. Character expressions are compared character for character from left to right until one of the expressions isn't equal to the other, *or* until the end of the expression on the right side of the = operator is reached (SET EXACT OFF) or until the ends of both expressions are reached (SET EXACT ON).

The == operator can be used when an exact comparison of character data is needed. If two character expressions are compared with the == operator, the expressions on both sides of the == operator must contain exactly the same characters, including blanks, to be considered equal. The SET EXACT setting is ignored when character strings are compared with ==.

The following table shows how the choice of operator and the SET EXACT setting affect comparisons. (An underscore represents a blank space.)

Comparison	= EXACT OFF	= EXACT ON	== EXACT OFF or ON
“abc” = “abc”	Match	Match	Match
“ab” = “abc”	No match	No match	No match
“abc” = “ab”	Match	No match	No match
“abc” = “ab_”	No match	No match	No match
“ab” = “ab_”	No match	Match	No match
“ab_” = “ab”	Match	Match	No match
“” = “ab”	No match	No match	No match
“ab” = “”	Match	No match	No match
“__” = “”	Match	Match	No match
“” = “__”	No match	Match	No match
TRIM(“__”) = “”	Match	Match	Match
“” = TRIM(“__”)	Match	Match	Match

SET EXCLUSIVE

Purpose	Specifies whether FoxPro opens table/.DBF files for exclusive or shared use on a network
Syntax	SET EXCLUSIVE ON OFF
See Also	FLOCK(), RLOCK(), USE

Description SET EXCLUSIVE determines whether tables/.DBFs opened on a network are available for exclusive use by one user or can be shared by every user.

Changing the setting of SET EXCLUSIVE doesn't change the status of previously opened tables/.DBFs. For example, if a table/.DBF is opened with SET EXCLUSIVE ON, and SET EXCLUSIVE is later changed to OFF, the table/.DBF retains its exclusive use status.

Clauses ON

A table/.DBF opened on a network when SET EXCLUSIVE is ON is made available only to the user who opened the table/.DBF. The table/.DBF isn't accessible to other users on the network. Unlike FLOCK(), SET EXCLUSIVE ON also prevents all other users from having read-only access. A file can also be opened on a network for exclusive use by including EXCLUSIVE with USE. It isn't necessary to perform record or file locking on a table/.DBF opened for exclusive use.

Opening a table/.DBF for exclusive use ensures that the file can't be changed by other users. Some commands require that a table/.DBF be opened for exclusive use before they can be executed. These commands are INSERT, INSERT BLANK, MODIFY STRUCTURE, PACK, REINDEX and ZAP.

The default for SET EXCLUSIVE is ON.

OFF

If a table/.DBF is opened on a network when SET EXCLUSIVE is OFF, the table/.DBF can be shared and modified by any user on the network.

For additional information about record and file locking and sharing tables/.DBFs on a network, see the FoxPro in a Multi-User Environment chapter in the *FoxPro Developer's Guide*.

SET FIELDS

Purpose	Specifies whether all or only specified fields in a table/.DBF can be accessed
Syntax	SET FIELDS ON OFF or SET FIELDS TO [[<field1> [, <field2> ...]] ALL]
See Also	SET FILTER

Description SET FIELDS specifies which fields in a table/.DBF can be accessed.

Clauses **ON**
Only the fields specified in the field list can be accessed when SET FIELDS is ON.

OFF
All the fields in the current table/.DBF are accessible if SET FIELDS is OFF. SET FIELDS OFF is the default setting.

TO [<field1> [, <field2> ...]] | TO ALL
<field1> and <field2> are the names of fields that can be accessed in the current table/.DBF. An alias must be included with a field name in these cases:

- When referring to a field in a table/.DBF in another work area
- When the field names are the same in two or more tables/.DBFs

SET FIELDS TO is additive — issuing SET FIELDS TO with a field list adds the specified fields to those that are currently accessible. Use SET FIELDS TO ALL to access all the fields in the current table/.DBF.

Issuing SET FIELDS TO implicitly performs SET FIELDS ON. Issue SET FIELDS TO without including a field list or ALL to remove all fields from the field list for the current table/.DBF.

SET FILTER

Purpose	Specifies a condition that records in the current table/.DBF must meet to be accessible
Syntax	SET FILTER TO [<expL>]
See Also	FILTER()
Description	SET FILTER is used to designate a group of records within the current table/.DBF that satisfy the condition specified in the logical expression <expL>. Once SET FILTER is issued, only the records that satisfy the condition are displayed in the table/.DBF. All commands that access the table/.DBF respect the SET FILTER condition. The condition specified by SET FILTER isn't evaluated until the record pointer is moved in the table/.DBF.
Clause	<expL> <expL> specifies the condition that records must satisfy. Issuing SET FILTER TO without <expL> turns off the filter for the current table/.DBF. A separate filter can be set for every open table/.DBF.

SET FIXED

Purpose	Specifies whether or not the number of decimals places used in the display of numeric data is fixed
Syntax	SET FIXED ON OFF
See Also	SET DECIMALS

Description SET FIXED determines whether a fixed number of decimal places is used for displaying numeric data.

Clauses **ON**

The number of decimal places displayed is determined by SET DECIMALS if SET FIXED is ON. The default number of decimal places is 2.

OFF

If SET FIXED is OFF, the number of decimal places displayed in results depends on the specific constants, variables and operators used in a numeric expression. The contents of fields are displayed with the declared number of decimal places.

The default for SET FIXED is OFF.

SET FORMAT

Purpose	Opens a format file
Syntax	SET FORMAT TO [<file> ?]
Remarks	Included for backward compatibility. Use a multiple window READ instead.
See Also	@ ... SAY, @ ... GET, DO, READ, SYS(7)
Description	SET FORMAT opens a format file for use with APPEND, BROWSE, INSERT and READ.
Clauses	<file> ? Specify the format file to open with <file>. SET FORMAT TO ? displays a list of available format files. Format files have a .FMT or .PRX extension. Issue SET FORMAT TO without <file> to close a format file in the current work area.

SET FULLPATH

Purpose	Specifies whether or not CDX(), DBF(), IDX() and NDX() return the path in a file name
Syntax	SET FULLPATH ON OFF
See Also	CDX(), DBF(), IDX(), NDX()
Description	Use SET FULLPATH to include or exclude a path in the file name returned by DBF() and NDX().
Clauses	ON If SET FULLPATH is ON, CDX(), DBF(), IDX(), and NDX() return the drive designator, path and file name. This is the default setting. OFF Only the drive designator and file name are returned if SET FULLPATH is OFF.

SET FUNCTION

Purpose	Assigns an expression to a function key
Syntax	SET FUNCTION <expN> <key label> TO [<expr>]
See Also	CLEAR MACROS, FKLABEL(), FKMAX(), ON KEY LABEL
Description	SET FUNCTION assigns a keyboard macro to a function key.
Clauses	<expN> <key label> Specify the number of the function key you want to assign the macro to with <expN>. For example, use SET FUNCTION 2 to specify the F2 function key. FoxPro also supports key combinations that include function keys. You can use the Ctrl or Shift key in combination with a function key to create additional programmable keys. Use <key label> to specify the function key combination to program. See ON KEY LABEL for a list of key label expressions. <expr> Specify the series of keystrokes stored to the function key with <expr>. FoxPro translates a semicolon (;) in the expression as a carriage return. Function key definitions can be cleared with CLEAR MACROS.

SET HEADING

Purpose	Determines whether or not column headings are displayed for fields
Syntax	SET HEADING ON OFF
See Also	AVERAGE, CALCULATE, DISPLAY, LIST, SUM
Description	SET HEADING specifies whether a field name is displayed as a column heading above each field in the output of AVERAGE, CALCULATE, DISPLAY, LIST and SUM.
Clauses	ON The field names are displayed if SET HEADING is ON. This is the default setting. OFF The field names aren't displayed if SET HEADING is OFF.

SET HELP

Purpose	Enables or disables the FoxPro online help facility or specifies a help file
Syntax	SET HELP ON OFF or SET HELP TO [<file>]
See Also	HELP, SET HELPFILTER, SET TOPIC

Description Use SET HELP to enable or disable the FoxPro online help facility or to specify a different help file.

See the FoxPro *User's Guide* for more information on the FoxPro online help facility.

Clauses **ON**

If SET HELP is ON, you can display the Help window by pressing F1 or issuing HELP in the Command window. In FoxPro for MS-DOS, you can also choose **Help...** from the **System** menu. In FoxPro for Windows, choose **Contents...** from the **Help** menu.

The default setting for SET HELP is ON.

OFF

If SET HELP is OFF, the FoxPro online help facility isn't available.

<file>

You can specify a different help file with <file>. This is a convenient way to provide a tailored online help facility within a custom application or to switch between the different help files in FoxPro for Windows.

If you issue SET HELP TO without including a file name, FoxPro for MS-DOS looks for a help file with the name FOXHELP.DBF.

Choosing an Online Help Style

FoxPro for Windows gives you the option of using either Windows-style help or FoxPro DBF-style help. If you run other Windows-based applications and are accustomed to Windows-style help, use Windows-style help. If you use FoxPro for MS-DOS and like its help system, use DBF-style help.

When you install FoxPro for Windows, you are prompted to install either one or both styles of help. To use either help style, the required help files must be installed. To use Windows-style help, FOXHELP.HLP must be installed. To use DBF-style help, FOXHELP.DBF and FOXHELP.FPT must be installed. If you install both, Windows-style help is used by default.

Changing Help Styles

You can switch between help styles at any time in the Command or View window or from within a program. A default startup help style can be specified in your CONFIG.FPW configuration file.

Specifying a Help Style in the Command Window

Issue one of the following commands in the Command window or from within a program to change help styles:

- SET HELP TO foxhelp.hlp — Switch to Windows-style help.
- SET HELP TO foxhelp.dbf — Switch to DBF-style help.

Specifying a Help Style in the View Window

Open the View window by issuing SET in the Command window or by choosing **View** from the **Window** menu. Choose the **Files** push button, and then choose the **Help...** push button.

When you choose the **Help...** push button, the Open dialog appears. To specify a help style:

- For Windows-style help, choose FOXHELP.HLP from the list.
- For FoxPro DBF-style help, choose FOXHELP.DBF from the scrollable list.

Specifying a Startup Help Style in Your Configuration File

Place one of the following in your CONFIG.FPW configuration file:

- HELP = FOXHELP.HLP — Specifies Windows-style help.
- HELP = FOXHELP.DBF — Specifies DBF-style help.

If the help file you specify in the Command or View window or your configuration file is in a directory other than the current default directory, include a path with the help file name.

SET HELPFILTER

Purpose	Enables FoxPro to display a subset of help topics in the Help window
Syntax	SET HELPFILTER [AUTOMATIC] TO [<expL>]
See Also	HELP, SET HELP, SET TOPIC

Description SET HELPFILTER enables FoxPro to display a subset of help topics in the Help window by evaluating the logical expression <expL> and displaying only the topics for which <expL> evaluates to true (.T.).

In FoxPro for Windows, you can set a filter only for DBF-style help. You cannot set a filter for Windows-style help.

Clauses **AUTOMATIC**

You can automatically remove the criteria specified with SET HELPFILTER after the Help window is closed by including the AUTOMATIC clause. Including AUTOMATIC is identical to issuing SET HELPFILTER TO immediately after the Help window is closed. AUTOMATIC must be placed directly before TO <expL>.

<expL>

The help topics are filtered based on the evaluation of the logical expression <expL>. Only records in the help table/.DBF that satisfy the logical expression are displayed in the Help window. <expL> typically includes a the name of a field from the help .DBF. You can include the name of any field from the help .DBF.

Help File Structure

In FoxPro for MS-DOS, the default help table/.DBF used by the FoxPro online help facility is FOXHELP.DBF. In FoxPro for Windows, FOXHELP.DBF is used when you specify DBF-style help. The following table lists the structure of FOXHELP.DBF.

Field	Field type and description
TOPIC	A character field. The list of topics is displayed in the Help window.
DETAILS	A memo field. The details are displayed in the second panel of the Help window.
CODE	A character field. Codes are used to classify the help topics.

The CODE field contains two-letter abbreviations that classify each record and can be used to filter help topics. The following table contains the help categories and their codes.

Help Filter Category	Code
General	General
What's New	wn
Compatibility	cm
Configuration	cf
Error Messages	em
Commands	Command
Table/.DBF (Fields, Indexes, Records, Relations)	db
Enhanced in FoxPro version 2.5	ex
Environment (SET Commands, Screen, Keys)	en
Errors and Debugging	er
Event Handlers (ON ERROR, ON KEY)	eh
File Management	fm
Indexing	ix
Keyboard and Mouse	km
Memory Variables and Arrays	mv
Menus and Popups (System Names)	mp
New in FoxPro 2.5	nx
Printing	pr
Program Execution	pe
Structured Programming	sp
SQL	sq
Text Merge	tm
Windows	wi
Interface	in
Multi-User	mu

Help Filter Category	Code
Functions	Function
Character	ch
Numeric	nu
Date and Time	dt
File Input/Output	fi
Logical	lo
Table/.DBF (Fields, Indexes, Records, Relations)	db
Data Conversion	dc
Environment	en
File Management	fm
Keyboard and Mouse	km
Low-Level	ll
Memory Variables and Arrays	mv
Menus and Popups (System Names)	mp
Windows	wi
Printing	pr
Multi-User	mu
System Memory Variables	Sysmemvar
Desk Accessories	da
Printing	pr
Interface Operations	in
Text Merge	tm
System Menu Names	sn
Interface	Interface
Dialogs	di
General	ge
Menus	me
Windows	wi

Choosing a topic from the **See Also** popup may display a “No help found for...” message. If you have issued SET HELPFILTER TO, topics in the **See Also** popup must meet the filter condition for additional help to be displayed.

Issue SET HELPFILTER TO without additional arguments to remove a filter. The filter criteria are removed, and all help topics are displayed in the Help window.

Example

```
SET HELP TO foxhelp.dbf
SET HELPFILTER TO topic = 'ACTIVATE'
SET HELPFILTER TO 'WINDOW' $ topic
SET HELPFILTER AUTOMATIC TO 'BROWSE' $ details
SET HELPFILTER TO 'Command' $ class AND 'db' $ class
```

SET HOURS

Purpose	Sets the system clock to a 12- or 24-hour time format
Syntax	SET HOURS TO [12 24]
See Also	SECONDS(), SET CLOCK, TIME()
Description	SET HOURS sets the system clock to a 12- or 24-hour format. Use SET HOURS TO without 12 or 24 to return to the default 12-hour format. TIME() always returns a value in 24-hour format and isn't affected by SET HOURS.
Clauses	12 Include 12 to specify a 12-hour format. This is the default setting. 24 Include 24 to specify a 24-hour format.

SET INDEX

Purpose	Opens one or more index files for use with the current table/.DBF
Syntax	SET INDEX TO [<index file list> ?] [ORDER <expN> <idx index file> [TAG] <tag name> [OF <cdx file>][ASCENDING DESCENDING]] [ADDITIVE]
See Also	CLOSE INDEXES, INDEX, SET ORDER, USE

Description SET INDEX opens one or more index files for the current table/.DBF. Records in a table/.DBF that has an index file or files open can be displayed and accessed in an order determined by one of the index files. Both .IDX and .CDX (compound) index files can be opened with SET INDEX. If a table/.DBF has a structural compound index file, the file is opened automatically when the table/.DBF is opened.

Only one .IDX index file (the controlling index file) or tag from a .CDX compound index file (the controlling tag) controls the order in which records in the table/.DBF are displayed or accessed. Certain commands (SEEK, for example) use the controlling index file or tag to search for records.

Issuing SET INDEX TO without any additional arguments closes all open index files (except a structural compound index file) in the current work area.

Clauses **<index file list> | ?**

You can open a single index file by specifying the index file name with <index file list>. You don't have to include the file name extensions unless another .IDX or .CDX file exists with the same name.

You can open more than one index file by including a list of index file names separated by commas in the index file list. The index file list can contain any combination of .IDX and .CDX index file names.

The first index file in the index file list becomes the controlling index file, which controls how records are accessed and displayed. The table/.DBF records are displayed and accessed in physical record order if the first index file is a compound index file and SET ORDER TO TAG has not been issued.

You can also open a single .IDX index file with SET INDEX TO ?. The Open dialog appears displaying a list of available index files to choose from.

ORDER <expN>

Use ORDER <expN> to designate a controlling index file or tag. The numeric expression <expN> specifies the index files as they appear in the index file list. .IDX index files are numbered first in the order in which they appear in the index file list. Tags in the structural compound index file (if one exists) are numbered in the order in which the tags are created. Finally, tags in any independent .CDX compound index files are numbered in the order in which they were created. See SET ORDER in this chapter for a further discussion of the numbering of index files and tags.

If <expN> is 0, records in the table/.DBF are displayed and accessed in physical record order, but the index files remain open. ORDER 0 enables you to update open index files while accessing records in physical order. ORDER with no arguments is identical to ORDER 0.

The error message “Index tag not found” is displayed if <expN> is greater than the number of .IDX index files and .CDX compound index file tags.

ORDER <idx index file>

Include <idx index file> to designate the specified .IDX index file as the controlling index file.

ORDER [TAG] <tag name> [OF <cdx file>]

Include <tag name> to designate a tag of the specified .CDX index file as the controlling tag. The tag can be from the structural compound index file or any open compound index file.

Include OF <cdx file> if tags with the same name exist in open compound index files.

ASCENDING | DESCENDING

Include ASCENDING or DESCENDING to specify whether table/.DBF records are displayed and accessed in ascending or descending order. The index files or tags aren't changed in any way; only the order in which the records are displayed and accessed is changed. Include the ASCENDING or DESCENDING key word immediately after the ORDER clause.

ADDITIVE

When you open an index file or files for a table/.DBF with SET INDEX, any previously opened index files (except a structural compound index) are closed unless you include ADDITIVE.

SET INTENSITY

Purpose	Determines whether or not FoxPro uses the enhanced screen color attribute for the display of editing fields
Syntax	SET INTENSITY ON OFF
Remarks	FoxPro for MS-DOS only. Included for backward compatibility. Use SET COLOR OF SCHEME instead.
See Also	SET BLINK

Description In FoxPro for MS-DOS, SET INTENSITY specifies whether the enhanced screen attribute is used for fields displayed with @ ... GET, APPEND, BROWSE, CHANGE or EDIT.

SET INTENSITY is ignored in FoxPro for Windows.

Clauses **ON**

The enhanced screen attribute is used to highlight fields if SET INTENSITY is ON. This is the default setting.

OFF

The normal screen attribute is used if SET INTENSITY is OFF.

SET KEYCOMP

Purpose	Controls FoxPro keystroke navigation on different platforms
Syntax	SET KEYCOMP TO DOS WINDOWS
Remarks	FoxPro for Windows only
See Also	SET COMPATIBLE

Description SET KEYCOMP determines the keystrokes that you use to move through the FoxPro for Windows interface and lets you specify the keystroke and keystroke combinations used to navigate the interface controls (buttons, lists, popups, and so on).

The effect of the SET KEYCOMP setting depends on the control.

If you are familiar with FoxPro for MS-DOS, you can issue SET KEYCOMP TO DOS to navigate using the keystrokes you are accustomed to. If you are familiar with Windows and prefer to navigate with the keystrokes Windows uses, issue SET KEYCOMP TO WINDOWS.

Default Buttons

SET KEYCOMP TO DOS

The default button in a dialog is always the same and is chosen by pressing Ctrl+Enter.

SET KEYCOMP TO WINDOWS

The default button in a dialog can change when you move between controls. The default button is surrounded by a bolder border to indicate it is the current default and is chosen by pressing Enter. Pressing Enter always performs the default button's action.

For an illustration of how the default button changes in a dialog, issue SET KEYCOMP TO WINDOWS and tab through the Open dialog.

Popups

SET KEYCOMP TO DOS

Popups are opened by pressing Enter or the Spacebar. Keyboard steering within a popup is unavailable until the popup is open.

SET KEYCOMP TO WINDOWS

Popups are opened by pressing the Spacebar, Alt+Up Arrow or Alt+Down Arrow. Keyboard steering within a popup is available when the popup is selected (not open) *and* when it is open.

For example, a selected popup contains a list of available drives. If the A, B and C drives are available and the B drive is currently displayed, you can choose the C drive without opening the popup by pressing C or the Down Arrow key. The C drive is chosen and you move to the next dialog option.

Hot Keys**SET KEYCOMP TO DOS**

A hot key for a control is a single key. If you aren't in a control that has keyboard steering (a popup or a list), pressing the hot key moves you to the control and chooses it.

SET KEYCOMP TO WINDOWS

A hot key for a control can be a single key or a key combination. If the current control has keyboard steering (a popup or a list), you can press Alt and the hot key to move you to the control. To move to other controls, you can press the hot key or Alt and the hot key.

Radio Buttons**SET KEYCOMP TO DOS**

Pressing the Tab key when a group of radio buttons is selected moves you among the radio buttons.

SET KEYCOMP TO WINDOWS

Pressing the Tab key when a group of radio buttons is selected moves you from the radio buttons to the next control. To move among buttons in a set of radio buttons, press the Left and Right Arrow keys.

You can specify a startup KEYCOMP setting in your FoxPro for Windows CONFIG.FPW configuration file. For example, placing the following line in your CONFIG.FPW configuration file sets KEYCOMP to DOS:

```
KEYCOMP = DOS
```

SET LIBRARY

Purpose	Opens an external library file
Syntax	SET LIBRARY TO [<file name> [ADDITIVE]]
See Also	CALL, DISPLAY STATUS, LIST, LOAD, RELEASE

Description Use SET LIBRARY to open external API (Application Program Interface) routine libraries. An API routine library extends the capabilities of the FoxPro language and user interface. Once an external API routine library is opened, you can use the API functions as if they were FoxPro functions. The functions available in the library can be displayed with DISPLAY STATUS or LIST STATUS.

You can use existing API libraries or create your own API routine libraries with the FoxPro Library Construction Kit.

Clauses **<file name>**

Specify the file name of the API library to open with <file name>.

FoxPro for MS-DOS assumes a .PLB extension for libraries. If the library has a .PLB extension, you don't have to include the extension with the file name. If a library has an extension other than .PLB, you must include the extension with the file name.

FoxPro for Windows assumes an .FLL extension for libraries. If the library has an .FLL extension, you don't have to include the extension with the file name. If a library has an extension other than .FLL, you must include the extension with the file name.



You cannot use libraries built for one platform on another platform. For example, libraries created for FoxPro for MS-DOS cannot be used in FoxPro for Windows; libraries created for FoxPro for Windows cannot be used in FoxPro for MS-DOS.

You cannot use libraries built for FoxPro for MS-DOS version 2.0 in FoxPro for MS-DOS version 2.5.

ADDITIVE

To open additional API libraries, include ADDITIVE after the file name in successive SET LIBRARY commands.

To remove all API libraries from memory, issue SET LIBRARY TO without including <file name> or ADDITIVE. To remove an individual library from memory, issue RELEASE LIBRARY <library name>.

SET LOCK

Purpose	Enables or disables automatic file locking in certain commands
Syntax	SET LOCK ON OFF
See Also	FLOCK(), LOCK(), RLOCK(), SET MULTLOCKS

Description You can enable or disable automatic file locking with SET LOCK. FoxPro doesn't place a lock on a file when executing commands that require read-only access to a table/.DBF. Here is a list of commands that require read-only access to a table/.DBF:

Commands		
AVERAGE	DISPLAY (with a scope)	REPORT
CALCULATE	INDEX	SORT
COPY TO	JOIN (both files)	SUM
COPY TO ARRAY	LIST	TOTAL
COUNT	LABEL	

While they are being executed, these commands don't change the contents of a table/.DBF, and access to the table/.DBF is available to other users on the network. The table/.DBF can be changed while you are executing one of these commands. For example, you may begin printing a report before another user changes a record included in the report. Your report now contains outdated information.

Clauses **ON**

Use SET LOCK ON to ensure that you are using the most current data. If SET LOCK is ON, the commands listed above automatically lock the table/.DBF. This provides read-only access to other users on the network.

The results in the above example can be achieved by issuing FLOCK() followed by REPORT FORM and UNLOCK. In this case, files aren't locked automatically.

OFF

Use SET LOCK OFF to enable shared access of tables/.DBFs with the commands listed above. SET LOCK OFF can be used if having the most current information isn't essential.

The default for SET LOCK is OFF.

SET LOGERRORS

Purpose	Determines whether or not FoxPro sends compilation error messages to a text file
----------------	--

Syntax	SET LOGERRORS ON OFF
---------------	-------------------------------

See Also	COMPILE
-----------------	---------

Description Using SET LOGERRORS, you can save compilation error messages to a text file when programs are compiled.

Clauses **ON**

Issuing SET LOGERRORS ON creates a compilation error message log file that has the same name as the compiled program and an .ERR extension. If a log file with the same name already exists, it is overwritten.

The default for SET LOGERRORS is ON.

OFF

A log file isn't created if a program compiles without error or if SET LOGERRORS is OFF. If a log file exists with the same name as the compiled program and the program compiles without error, the log file is deleted.

SET MACKEY

Purpose	Specifies a key or key combination that displays the Macros dialog
Syntax	SET MACKEY TO [<expC>]
See Also	CLEAR, ON KEY LABEL, PLAY MACRO, RESTORE MACRO, SAVE MACROS

Description You can define a key or key combination that displays the Macros dialog. By default, pressing Shift+F10 brings forward the Macros dialog. You can change this key combination to another key or key combination with SET MACKEY.

Clause **<expC>**

Specify the key or key combination that displays the Macros dialog with <expC>. See ON KEY LABEL for the syntax of the key names and key combinations.

If you issue SET MACKEY TO without a key name or key combination name, you cannot activate the Macros dialog by pressing a key or key combination.

SET MARGIN

Purpose	Specifies the left margin in printed output
Syntax	SET MARGIN TO <expN>
See Also	<code>_LMARGIN</code> , <code>_PLOFFSET</code>

Description This command sets the left printer margin and affects all output directed to the print device. If you use SET MARGIN to adjust the left margin, the value specified with SET MARGIN is stored to the system memory variable `_PLOFFSET`. You can also set the left margin by storing a value directly to `_PLOFFSET`.

The value of the system memory variable `_LMARGIN` also affects the left margin setting.



The left margin setting specified in SET MARGIN doesn't affect reports created with the Report Writer and run with REPORT FORM. Although `_PLOFFSET` is adjusted during the running of a report created with the Report Writer, it is reset to its original value after the report is run. The Printer Indent setting in the Page Layout dialog of the Report Writer determines the offset from the left edge of the paper.

Clause **<expN>**
Specify the left margin in columns with <expN>. The default is 0 columns.

SET MARK OF

Purpose	Specifies a menu pad or popup option mark character
Syntax	SET MARK OF MENU <menu name1> TO <expC1> <expL1> or SET MARK OF PAD <pad name> OF <menu name2> TO <expC2> <expL2> or SET MARK OF POPUP <popup name1> TO <expC3> <expL3> or SET MARK OF BAR <expN> OF <popup name2> TO <expC4> <expL4>
Remarks	Mark characters cannot be changed in FoxPro for Windows
See Also	CNTBAR(), CNTPAD(), DEFINE BAR, DEFINE MENU, DEFINE PAD, DEFINE POPUP, MRKBAR(), MRKPAD()

Description Use these commands to specify the mark character for a menu pad or popup option.

In FoxPro for Windows, you cannot specify a different mark character for a menu pad or popup option. The mark character is always a check mark (✓). However, you can use SET MARK OF to display or remove the check mark for a menu pad or popup option.

You can place a mark character to the left of a menu bar pad or a popup option. A mark character can be used to indicate that a certain condition exists; for example, that a popup is available for the pad, an option controlled by the pad is on or off, and so forth.

Mark characters can be placed to the left of pads in the system menu bar and options on the system popups.

These commands also let you display or remove the mark character. MRKPAD() and MRKBAR() facilitate the display or removal of mark characters for menu pad and popup options.

Clauses MENU <menu name1>

A mark character can be specified for every pad in a menu with SET MARK OF <menu name1>. Specify the name of the menu with <menu name1>.

TO <expC1> — FoxPro for MS-DOS only

In FoxPro for MS-DOS, the mark character is specified with <expC1>. If <expC1> contains more than one character, only the first character is used as the mark character.

<expL1>

If the logical expression <expL1> evaluates to true (.T.), the mark character is displayed beside every pad in the menu. Set <expL1> to false (.F.) to remove the mark character.

PAD <pad name> OF <menu name2>

You can specify a mark character for an individual menu pad with SET MARK OF PAD <pad name> OF <menu name2>. Specify the name of the menu with <menu name2> and the name of the pad with <pad name>.

TO <expC2> — FoxPro for MS-DOS only

In FoxPro for MS-DOS, a different mark character can be specified for each pad in a menu. The mark character for a pad is specified with <expC2>.

<expL2>

Use the logical expression <expL2> to display or clear the mark character. If <expL2> evaluates to true (.T.), the mark character is displayed. The mark character is cleared if <expL2> evaluates to false (.F.).

POPUP <popup name1>

Place mark characters beside every option on a popup with SET MARK OF POPUP <popup name1>.

TO <expC3> — FoxPro for MS-DOS only

In FoxPro for MS-DOS, <expC3> is the mark character you want to display beside every option on a popup.

<expL3>

Mark characters for all options are displayed or cleared with the logical expression <expL3>.

SET MARK OF

BAR <expN> OF <popup name2>

Place mark characters beside individual popup options with SET MARK OF BAR <expN> OF <popup name2>.

TO <expC4> — FoxPro for MS-DOS only

In FoxPro for MS-DOS, <expC4> is the mark character you want to display beside individual options.

TO <expL4>

The mark characters for individual popup options are displayed or cleared with <expL4>.

You cannot mark options on a popup created with any of the PROMPT clauses (FIELD, FILES or STRUCTURE).

Example See CNTBAR().

SET MARK TO

Purpose	Specifies a delimiter to use in the display of date expressions
Syntax	SET MARK TO [<expC>]
See Also	SET DATE

Description SET MARK TO specifies the character that separates the month, day and year in displayed dates.

Clause <expC>
<expC> is the character you want to use as the date delimiter.
Use SET MARK TO without <expC> to reset the delimiter to the default forward slash (/).

SET MEMOWIDTH

Purpose	Specifies the displayed width of memo fields and character expressions
Syntax	SET MEMOWIDTH TO <expN>
See Also	ATCLINE(), ATLINE(), MEMLINES(), MLINE()

Description SET MEMOWIDTH specifies the width of output sent to the desktop in FoxPro for MS-DOS, the main FoxPro window in FoxPro for Windows or a user-defined window by commands like ???, DISPLAY or LIST. It affects the output width of memo fields and character expressions that are longer than 254 characters. It also affects the values returned by the ATCLINE(), ATLINE(), MEMLINE() and MLINE() functions.

In FoxPro for Windows, if output is directed to the main FoxPro window, the width of the output is determined by the font for the main FoxPro window. If output is directed to a user-defined window, the width of the output is determined by the font for user-defined window.

Clause **<expN>**

You can specify a width between 8 and 256 columns. The default width for output is 50 columns.

SET MESSAGE

Purpose	Defines a message for display under the status bar or specifies the location of messages defined with user-defined menu bars and popup commands
Syntax	SET MESSAGE TO [<expC>] or SET MESSAGE TO [<expN> [LEFT CENTER RIGHT]] or SET MESSAGE WINDOW [<window name>]
See Also	SET STATUS, SET STATUS BAR

Description SET MESSAGE lets you create a message. It also lets you specify where to display messages created with @ ... PROMPT, DEFINE BAR, DEFINE MENU, DEFINE PAD or DEFINE POPUP.

In FoxPro for MS-DOS, messages are displayed on the last line of the desktop by default. In FoxPro for Windows, messages are placed on the last line of the main FoxPro window if the Windows-style status bar isn't displayed. If the Windows-style status bar is displayed, messages are placed in the status bar.

Clauses **TO [<expC>]**
Specify the message to display with <expC>. The status bar must be displayed before the message is displayed.

TO [<expN> [LEFT | CENTER | RIGHT]]
<expN> specifies the row on the desktop or main FoxPro window on which messages are displayed. If <expN> is 0, no messages are displayed. Issuing SET MESSAGE TO without any arguments places messages on the last row of the desktop or in the status bar in FoxPro for Windows.

You can also specify the horizontal screen placement of messages in @ ... PROMPTS with LEFT, CENTER and RIGHT.

In FoxPro for Windows a message location specified with SET MESSAGE is ignored when the Windows-style status bar is displayed.

WINDOW [<window name>]

If WINDOW <window name> is included, the message is displayed in the window specified with <window name>. To remove a message from the window and display it on the screen, issue SET MESSAGE WINDOW.

SET MOUSE

Purpose	Enables or disables the mouse and controls its sensitivity
Syntax	SET MOUSE ON OFF or SET MOUSE TO [<expN>]
Remarks	FoxPro for MS-DOS only
See Also	_DBLCLICK

Description	In FoxPro for MS-DOS, you can enable or disable the mouse and control its sensitivity with SET MOUSE. SET MOUSE is ignored in FoxPro for Windows. However, you can use the Windows Control Panel to make adjustments to the mouse.
Clauses	ON Include ON to enable the mouse. OFF Include OFF to disable the mouse. TO <expN> SET MOUSE TO controls the sensitivity of the mouse. The value of <expN> controls how fast the pointer moves in response to movement of the mouse. <expN> can range from 1 through 10; 1 is the least sensitive and 10 is the most sensitive. The default value for <expN> is 5.

SET MULTLOCKS

Purpose	Determines whether or not you can lock multiple records with LOCK() or RLOCK
Syntax	SET MULTLOCKS ON OFF
See Also	LOCK(), RLOCK()

Description When a table/.DBF is opened for shared use on a network, you can attempt to lock more than one record in a table/.DBF file. The MULTLOCKS setting determines whether you can attempt to lock either a single record or a set of records. Records can be locked with the LOCK() or RLOCK() function.



Note that toggling MULTLOCKS from ON to OFF or from OFF to ON implicitly issues UNLOCK ALL — all record locks in all work areas are released.

See LOCK() and RLOCK() in this chapter for additional information about record locking.

Clauses ON

You can attempt to lock a set of records if SET MULTLOCKS is ON. You can try to lock multiple records by including a set of record numbers in LOCK() or RLOCK().

OFF

You can attempt to lock a single record with LOCK() or RLOCK() if SET MULTLOCKS is OFF.

The default value for SET MULTLOCKS is OFF.

For additional information about record and file locking and sharing tables/.DBFs on a network, see the FoxPro in a Multi-User Environment chapter in the FoxPro *Developer's Guide*.

Example

```
SET MULTLOCKS ON
SELECT 0
USE customer
? LOCK('1,2,3,4,5', 1)
? RLOCK('1,2,3,4,5', 'customer')
USE
```

SET NEAR

Purpose	Determines where the record pointer is positioned after FIND or SEEK unsuccessfully searches for a record
Syntax	SET NEAR ON OFF
See Also	EOF(), FIND, FOUND(), RECNO(), SEEK

Description SET NEAR determines where the record pointer is positioned after an unsuccessful search for a record. A search is unsuccessful when no record meets the search criteria.

Clauses **ON**

The record pointer is positioned on the closest matching record if SET NEAR is ON and a record search using FIND or SEEK is unsuccessful. RECNO() returns the record number of the closest matching record, FOUND() returns false (.F.), and EOF() returns false (.F.).

OFF

The record pointer is positioned at the end of the table/.DBF if SET NEAR is OFF and a record search is unsuccessful. RECNO() returns the number of records in the table/.DBF plus 1, FOUND() returns false (.F.), and EOF() returns true (.T.).

Issuing RECNO() with an argument of 0 returns the record number of the closest matching record if a search is unsuccessful, regardless of the setting of SET NEAR.

The default value for SET NEAR is OFF.

SET NOTIFY

Purpose	Enables or disables the display of certain system messages
Syntax	SET NOTIFY ON OFF
See Also	SET MESSAGE, WAIT

Description You can use SET NOTIFY to suppress the display of some system messages.

Examples of system messages are the “Expression is valid” message in the Expression Builder dialog and the “Do Canceled” message displayed when program execution is canceled.

In FoxPro for MS-DOS, system messages are displayed in a window in the upper right corner of the desktop.

In FoxPro for Windows, system messages are displayed in the status bar at the bottom of the main FoxPro window.

Clauses **ON**

If SET NOTIFY is ON, certain system messages are displayed. This is the default setting.

OFF

If SET NOTIFY is OFF, the display of certain system messages is suppressed.

SET ODOMETER

Purpose	Determines the reporting interval of the record counter for commands that process records
Syntax	SET ODOMETER TO [<expN>]
See Also	SET TALK, _TALLY

Description Use SET ODOMETER to change the interval at which commands display information about the number of records processed.

For example, COPY displays the number of records that are copied to a new file while the command is executing. The record counter can be turned off by setting TALK OFF.

Clause **<expN>**
The value of <expN> can range from 1 to 32,767 records. The default value is 100 records.

SET OPTIMIZE

Purpose	Enables or disables Rushmore optimization
Syntax	SET OPTIMIZE ON OFF
See Also	INDEX, SET ORDER

Description FoxPro uses a technology called Rushmore to optimize data retrieval. Table/.DBF commands that support a FOR clause use Rushmore technology to enhance their performance. In rare cases, you should disable Rushmore optimization. When a command that uses Rushmore technology is issued, Rushmore determines which records match the FOR criterion. The command is executed on records in the table/.DBF that match the Rushmore record set.

If a command that benefits from Rushmore optimization modifies a query’s index keys, the Rushmore record set may become outdated. You can disable Rushmore optimization to ensure that you have the most current information from the table/.DBF.

Rushmore technology can be globally disabled (or enabled) for all commands that benefit from it with SET OPTIMIZE. Every command that uses Rushmore has a NOOPTIMIZE clause that can be included to disable Rushmore optimization for the command. For additional information on Rushmore technology, see the Optimizing Your Application chapter in the FoxPro *Developer’s Guide*.

Here are the commands whose performance is optimized by Rushmore:

Commands That Benefit from Rushmore			
AVERAGE	COUNT	LABEL	SCAN
BROWSE	DELETE	LIST	SORT
CALCULATE	DISPLAY	LOCATE	SUM
CHANGE	EDIT	RECALL	TOTAL
COPY TO	EXPORT	REPLACE	
COPY TO ARRAY	INDEX	REPORT	

Clauses **ON**

Issuing SET OPTIMIZE ON enables Rushmore optimization.

OFF

Issuing SET OPTIMIZE OFF disables Rushmore optimization.

SET ORDER

Purpose	Designates a controlling index file or tag for the current or specified table
Syntax	SET ORDER TO [<expN1> <idx index file> [TAG] <tag name> [OF <cdx file>] [IN <expN2> <expC>] [ASCENDING DESCENDING]]
See Also	INDEX, ORDER(), SET INDEX

Description A table/.DBF can have many index files open simultaneously. However, only one index file (the controlling index file) or tag from a compound index file (the controlling tag) determines the order in which the records in a table/.DBF are displayed or accessed. Certain commands (SEEK, for example) use the controlling index file or controlling tag to search for records. SET ORDER lets you designate the controlling index file or controlling tag. The controlling index also controls the order in which data is accessed.

You can open index files with a table/.DBF by including the INDEX clause in USE. If a table/.DBF has an associated structural compound index file, it is automatically opened with the table/.DBF. After a table/.DBF has been opened, you may open and close index files for the table/.DBF using SET INDEX.

Clauses <expN1>
<expN1> is the name of a controlling index file or controlling tag. <expN1> refers to the index files as they are listed in USE or SET INDEX. Open .IDX index files are numbered first in the order in which they appear in USE or SET INDEX. Tags in the structural compound index file (if one exists) are then numbered in the order in which the tags were created. Finally, tags in any open independent compound index .CDX files are numbered in the order in which they were created.

Example

The following example illustrates how different index file types and tags are numbered. (The file names are for illustration only and don't necessarily exist.) A table named VIDEO.DBF is opened with three indexes (TITLE.IDX, COSTS.CDX and RATING.IDX) in the first work area with this command:

```
USE video INDEX title.idx, costs.cdx, rating.idx IN 1
```

VIDEO.DBF has a structural compound index file (VIDEO.CDX) with two tags, NUMBERSOLD and YEARSOLD. The structural compound index file is automatically opened when VIDEO.DBF is opened. DISPLAY STATUS returns the following information about VIDEO.DBF and its index files:

Current Database:

Select area: 1, Database in Use: C:\foxpro25\video.dbf

Controlling index file: C:\foxpro25\title.idx Key: TITLE

Index file: C:\foxpro25\rating.idx Key: RATING

Structural CDX file: C:\foxpro25\video.cdx

Index tag: NUMBERSOLD Key: NUM_SOLD

Index tag: YEARSOLD Key: YR_SOLD

CDX file: C:\foxpro25\costs.cdx

Index tag: RENTALCOST Key: COST_RENT

Index tag: BUYCOST Key: COST_BUY

Since .IDX index files are numbered first, issue SET ORDER TO 1 to make TITLE.IDX the controlling index and SET ORDER TO 2 to make RATING.IDX the controlling index:

```
SET ORDER TO 1
```

```
Controlling index: C:\FOXPRO2\TITLE.IDX
```

```
SET ORDER TO 2
```

```
Controlling index: C:\FOXPRO2\RATING.IDX
```

The tags in VIDEO.CDX are numbered next:

```
SET ORDER TO 3
```

```
Controlling index: C:\FOXPRO2\VIDEO.CDX Tag: NUMBERSOLD
```

```
SET ORDER TO 4
```

```
Controlling index: C:\FOXPRO2\VIDEO.CDX Tag: YEARSOLD
```

Finally, the tags in the independent COSTS.CDX compound index file are numbered:

```
SET ORDER TO 5
```

```
Controlling index: C:\FOXPRO2\COSTS.CDX Tag: RENTALCOST
```

```
SET ORDER TO 6
```

```
Controlling index: C:\FOXPRO2\COSTS.CDX Tag: BUYCOST
```

SET ORDER TO 0 enables open index files to be updated without a performance penalty for tracing an index file structure. Issuing SET ORDER TO without additional arguments is identical to issuing SET ORDER TO 0.

If <expN> is greater than the number of .IDX index files and .CDX compound index file tags, the error message “Index tag not found” is displayed.

<idx index file>

You can also explicitly designate an .IDX index file as the controlling index file by specifying its name with <idx index file>.

[TAG] <tag name> [OF <cdx file>]

To explicitly designate a tag of a .CDX index file as the controlling tag, specify the tag name with <tag name>. The tag name can be from the structural compound index file or any open compound index file.

If identical tag names exist in open compound index files, include OF <cdx file> with the name of the .CDX file containing the desired tag.



The .IDX file takes precedence if duplicate .IDX file and tag names exist.

IN <expN2> | <expC>

SET ORDER designates the controlling index or controlling tag for the table/.DBF open in the current work area. You can designate the controlling index file or controlling tag for a table/.DBF open in another work area by specifying the work area number with <expN2> or the table/.DBF alias with <expC>.

ASCENDING | DESCENDING

Include ASCENDING or DESCENDING to specify whether table/.DBF records are displayed and accessed in ascending or descending order. Including ASCENDING or DESCENDING doesn't change the index file or tag in any way.

SET PALETTE

Purpose	Specifies whether or not the FoxPro color palette is used
Syntax	SET PALETTE ON OFF
Remarks	FoxPro for Windows only
See Also	@ ... SAY, MODIFY GENERAL

Description You can display and manipulate .BMP (bitmap) pictures and OLE (Object Linking and Embedding) objects in FoxPro for Windows.

.BMP pictures and OLE objects can contain color palettes that determine how the pictures and objects appear when they are displayed. The color palette of the last picture or object displayed is used for all subsequent pictures or objects. Since a single color palette is used for all the pictures and objects, the colors of some of the pictures and objects can be changed in a unexpected manner.

Clauses **ON**

FoxPro for Windows has a default color palette designed to improve the appearance of multiple .BMP pictures and OLE objects. This color palette is used by default. Use SET PALETTE ON to restore the default FoxPro for Windows color palette.

OFF

Use SET PALETTE OFF to use the color palettes from the .BMP pictures and OLE objects.

SET PATH

Purpose	Specifies a path for file searches
Syntax	SET PATH TO [<path>]
See Also	GETFILE(), LOCFILE(), SET DEFAULT
Description	SET PATH is used to designate a set of directories where FoxPro searches for files it can't find in the default directory. The default directory can be specified with SET DEFAULT.
Clause	<path> Specify the directories you want FoxPro to search with <path>. The directories listed in <path> should be separated with commas or semicolons.

SET PDSETUP

Purpose	Loads a printer driver setup
Syntax	SET PDSETUP TO [[<expC> [, <expr1> [, <expr2> ...]]] [WITH <expr3> [, <expr4> ...]]]
See Also	_GENPD, _PDRIVER, _PDSETUP, SET PRINTER

Description SET PDSETUP loads a printer driver setup or clears the current printer driver setup. A printer driver setup is used in FoxPro for Windows when printing character-based reports created in FoxPro for MS-DOS.

A *printer driver setup* is a combination of settings that include the printer driver program and printing information such as page orientation, default font size and style, margins and so on.

Printer driver setups are stored in your FoxPro resource file, FOXUSER.DBF, and can be created interactively and assigned a name in the Printer Setup Editing dialog.

A printer driver setup can also be loaded or cleared with the _PDSETUP system memory variable.

When SET PDSETUP is issued, the current printer setup interface application is executed. The interface application is passed the name of the printer driver setup included in SET PDSETUP. The interface application can be specified with the _GENPD system memory variable. The default interface application is GENPD.APP, the printer setup interface application included with FoxPro for MS-DOS.

Clauses <expC>

You can load a printer driver setup by specifying its name with <expC>.

When you load a printer driver setup, the name of the setup is stored in the _PDSETUP system memory variable, and a special memory variable array, _PDPARMS, can be created.

If the printer driver setup name you specify with <expC> doesn't exist in your resource file, the current printer driver setup application is executed so you can create a setup with this name. If the current printer driver setup application is GENPD.APP, the Printer Setup Editing dialog appears so you can create the setup.

If the name begins with a dash (-), the _GENPD program won't be executed, but the name following the dash is stored in _PDSETUP.

If you issue SET PDSETUP TO without <expC>, the current printer driver setup is cleared, the null string is stored in _PDSETUP and the _PDPARMS array is cleared from memory.

You can load a default printer driver setup when you start FoxPro for MS-DOS by including the following line your CONFIG.FP configuration file:

```
PDSETUP = '<setup name>' WITH <parm list>
```

<setup name> is the name of the setup to load when FoxPro for MS-DOS is started. Be sure to enclose the setup name in quotation marks. The default printer driver setup specified in CONFIG.FP takes precedence over a default printer driver setup specified in the Printer Driver Setup dialog.

<expr1> [, <expr2> ...]

You can also specify any number of optional parameters (<expr1>, <expr2> ...) with the printer driver setup name. These parameters are passed to the printer setup interface application and may be of any type (character, numeric, logical, and so on). The first line in your printer setup interface application must be a PARAMETERS statement to accept the parameters passed from SET PDSETUP.

If you are using GENPD.APP, don't include these optional parameters. GENPD.APP doesn't accept parameters passed from SET PDSETUP, so including them generates an error.

WITH <expr3>, [<expr4> ...]

The optional WITH clause can be included in SET PDSETUP to create the special _PDPARMS printer array. Each parameter you specify with <expr3> and <expr4> becomes an element in _PDPARMS. The first parameter (<expr3>) is stored to the first element of _PDPARMS, the second parameter (<expr4>) is stored to the second element and so on. These parameters may be of any type (character, numeric, logical, and so on).

If you are using GENPD.APP, any parameters you include are overwritten by the application.

SET POINT

Purpose	Determines the decimal point character used in the display of numeric expressions
Syntax	SET POINT TO [<expC>]
See Also	SET DECIMAL, SET FIXED

Description Use SET POINT to change the decimal point from the default, which is a period (.). Issue SET POINT TO without <expC> to reset the decimal point to a period. Although you can set the displayed decimal point to a different character, you must use a period as the decimal point in calculations.

Clause **<expC>**
<expC> is the character you want to represent a decimal point.

Example

```
x = 1.25
newpoint = '_'
SET POINT TO newpoint
? x
1_25

SET POINT TO    && Reset the decimal point to a period (.)
? x
1.25
```


SET PRINTER

Purpose	Enables or disables output to the printer and routes output to a file or a port
Syntax	SET PRINTER ON [PROMPT] OFF or SET PRINTER TO [<file> [ADDITIVE] <port>] or SET PRINTER TO [\\<machine name> \<printer name> = <dest>] or SET PRINTER TO [\\SPOOLER [NB] [F = <expN>] [B = <banner>] [C = <expN>] [P = <expN>]] [S = <server>] [Q = <queue>]
Remarks	PROMPT clause — FoxPro for Windows only
See Also	PRINTSTATUS(), SET DEVICE

Description	These commands let you turn output to a printer on or off and direct the output to a file, port or network printer.
Clauses	ON [PROMPT] Include ON to enable output to the printer. Output formatted with @ ... SAY isn't routed to the printer when SET PRINTER is ON. Use SET DEVICE TO PRINTER to direct output from @ ... SAY to the printer.

In FoxPro for Windows, you can include the optional PROMPT clause to display a dialog before printing starts. Place the PROMPT key word immediately after TO PRINTER. PROMPT is ignored in FoxPro for MS-DOS.

In this dialog you can adjust printer settings. You can specify the number of copies and page numbers to print and direct print output to a file. The Windows Printer Setup dialog can be opened to adjust additional printer settings. The printer settings you can adjust depend on the currently installed printer driver.

OFF

Include OFF to disable output to the printer. This is the default setting.

TO [<file> [ADDITIVE] | <port>]

You can specify a file or a port where output is directed with SET PRINTER TO. To direct output to a file, specify a file name with <file>. Output is appended to the existing contents of the file if you include ADDITIVE. Any existing contents of the file are overwritten if you omit ADDITIVE.

To send output to different local printers, you can specify one of three parallel ports (LPT1, LPT2 or LPT3) or one of two serial ports (COM1 or COM2) with <port>.

Use SET PRINTER TO without an argument to reset output to the default MS-DOS PRN print utility.

Printer output can be redirected to a network printer. Once SET PRINTER is issued for a network printer, output prints or collects in a print spooler until a new SET PRINTER command is issued. Consult your network documentation for additional information about printing on your network.

\\<machine name>\<printername> = <dest>

The first format of the network SET PRINTER command spools printer output to a network printer. <machine name> is the network name assigned to your workstation. This name is assigned by the network administrator and must be unique. <printer name> is a name assigned to the printer and is also assigned by the network administrator. LPT1, LPT2 or LPT3 is used to identify the destination (<dest>) of the installed printer.

The second format of the network SET PRINTER command is specifically designed for printing under Novell[®] Advanced Netware[®] version 2.00 or later.

\\SPOOLER

Although SPOOLER must be included for compatibility, it is ignored. Any character string can be substituted. When using the options below, don't include any space between the clauses.

\\WB

Include this clause to suppress printing a banner page (No Banner).

\\F = <expN>

This clause specifies the form number <expN> (0 through 255) that the output is printed on. This option is useful for producing different types of printed output, such as checks, invoices, letters, and so on.

\\B = <banner>

This clause specifies the heading that is printed on the banner page. <banner> can contain up to 12 characters. The default heading is your network user name.

\\C = <expN>

This clause specifies the number of copies (1 through 255) queued to a network printer. The default is 1.

\\P = <expN>

This clause specifies the network printer where output is sent. <expN> is a number assigned to the printer by the network. The default is 0.

\\S = <server>

This clause specifies the network server to which the printer is attached. <server> shouldn't be a character string (that is, it shouldn't be surrounded by quotation marks).

\\Q = <queue>

This clause specifies the queue name assigned to the printer you send output to. <queue> shouldn't be a character string (that is, it shouldn't be surrounded by quotation marks).

SET PROCEDURE

Purpose	Opens a procedure file
Syntax	SET PROCEDURE TO [<file>]
See Also	PROCEDURE

Description SET PROCEDURE opens the specified procedure file. Only one procedure file can be open at a time. Use SET PROCEDURE TO without a file name to close the currently open procedure file.

When you execute a program or a function, the procedure file is searched if the program or function isn't located in the currently executing program.

See PROCEDURE in this chapter for more information about procedure files.

Clause <file>

Specify the name of the procedure file you want to open with <file>. If <file> doesn't include an extension, .PRG is assumed.

SET REFRESH

Purpose	Determines whether or not your screen displays changes made to records by other users on the network
Syntax	SET REFRESH TO <expN1> [, <expN2>]
See Also	BROWSE, CHANGE, EDIT, MODIFY MEMO

Description Tables can be opened for shared use on a network. It is possible that records you are viewing are being edited by other users on the network. Use SET REFRESH to specify whether open Browse windows are updated with changes made by other users on the network.

SET REFRESH affects records displayed in a Browse window opened with BROWSE, CHANGE or EDIT. Memo fields opened for editing in a Browse window are also updated. SET REFRESH specifies how often these windows are updated.

SET REFRESH can also be used to specify how often data buffered locally on your workstation is updated.

Clauses **<expN1>**

The numeric expression <expN1> specifies how often a Browse or memo editing window is updated. <expN1> is the number of seconds between updates and can be a value from 0 to 3,600; the default value is 0 seconds. When <expN1> is a non-zero value and other users change records you are viewing, the records you are viewing are updated when the refresh interval elapses. The records you are viewing aren't updated if <expN1> is 0.

FoxPro attempts to lock a record when you try to modify the record in a Browse window. Regardless of the setting of SET REFRESH, if a record you are trying to lock is locked by another user on the network, you receive the updated record when the lock is released by the other user. The message "Record has changed" may also be displayed.

<expN2>

The network versions of FoxPro buffer portions of tables/.DBFs in memory on your workstation. <expN2> specifies how often these local data buffers are updated with current data from the network. <expN2> is the number of seconds between data buffer updates. The default value of <expN2> is 5; you can specify a value between 0 and 3,600. The buffers are never refreshed if <expN2> is set to 0.

If you specify a value for <expN1> other than 0 but don't include <expN2>, <expN2> is set to the same value as <expN1>. However, <expN2> is set to 5 if you specify 0 for <expN1> and don't include <expN2>.

Performance can be improved by increasing the value of <expN2>.

SET RELATION

Purpose	Links or unlinks two or more open tables/.DBFs
Syntax	SET RELATION TO [<expr1> INTO <expN1> <expC1> [, <expr2> INTO <expN2> <expC2> ...] [ADDITIVE]]
See Also	INDEX, RELATION(), SET ORDER, SET RELATION OFF, SET SKIP, TARGET()

Description SET RELATION establishes a relationship between two open tables. Before you establish a relationship, one table/.DBF (the parent table/.DBF) must be open in the current work area, and the other (the child table/.DBF) must be open in another work area. Then you can use SET RELATION to create the relationship.

After the relationship is created, moving the record pointer in the parent table/.DBF moves the record pointer to the corresponding record in the child table/.DBF. If a matching record can't be found in the child table/.DBF, the record pointer in the child table/.DBF is positioned at the end of the table/.DBF.

Related tables typically have a common field. For example, suppose a table (CUSTOMER.DBF) contains customer information. It has fields for name, address and a unique customer number. A second table (INVOICES.DBF) contains billing information. It too has a field for the customer number, along with fields for credit limit and payment terms.

You can use SET RELATION to relate these two tables on their common field — the customer number field. To set the relation, the child table/.DBF must be indexed on the common field. After you set the relation, whenever you move the record pointer to a record with a given customer number in the parent (CUSTOMER) table/.DBF, the record pointer in the child (INVOICES) table/.DBF moves to the record with the same customer number.

Clauses **<expr1>**

The general relational expression <expr1> establishes the relationship between the parent and child tables. The relational expression is usually the index expression of the controlling index of the child table, although it can be a numeric expression.

A child table is typically indexed on a character, numeric or date type field. The index for the child table can be a single-entry (.IDX) index, a multiple-entry structural compound index or an independent compound (.CDX) index. If the index is compound, specify the proper index tag to order the child table. SET ORDER can be used to specify the index tag that orders the child table.

For example, consider the CUSTOMER and INVOICES tables described above. Suppose the child (INVOICES) table has been indexed and ordered on the customer number with this command:

```
SET ORDER TO TAG cno
```

To relate the CUSTOMER and INVOICES tables on customer number, use SET RELATION, specifying the index expression as the following relational expression:

```
SET RELATION TO cno INTO invoices
```

The child table must be indexed unless the relational expression is numeric. The alert “Database isn’t ordered” is displayed if you issue SET RELATION with a non-numeric relational expression and the child table isn’t ordered with an index.

If <expr1> is numeric, it is evaluated when the record pointer in the parent table is moved. The record pointer in the child table is then moved to record number <expr1>.

Issue SET RELATION TO to remove all relationships in the current work area. Issue SET RELATION OFF to remove a specific parent-child relationship.

INTO <expN1> | <expC1>

The child table is identified by the work area in which it’s open or its table alias. Specify the child table’s work area number with <expN1> or its table alias with <expC1>.

<expr2> INTO <expN2> | <expC2> ...

You can create multiple relations from a single parent table with a single SET RELATION. Include a list of relations separated by commas to create relations to multiple child tables.

ADDITIVE

Include ADDITIVE to preserve all existing relationships in the current work area and create the specified relationship. If ADDITIVE isn't included, any relationships in the current work area are broken, and the specified relationship is created.

Example

The following example opens CUSTOMER.DBF (the parent) in the first work area, and INVOICES.DBF (the child) in the second work area. SET ORDER is used to specify the order of INVOICES.DBF.

The first work area is selected again, and a relationship is established between CUSTOMER.DBF and INVOICES.DBF. A Browse window is opened for each table. Move the record pointer in the parent Browse window through several records. The record pointer moves to the related record in the child (INVOICES) Browse window.

```

CLOSE DATABASES
USE customer IN 1      && The parent table/.DBF
USE invoices  IN 2      && The child table/.DBF
SELECT invoices
SET ORDER TO TAG cno    && Specify the table/.DBF order
SELECT customer         && Parent work area
SET RELATION TO cno INTO invoices    && Create the relationship

SELECT invoices    && Open two Browse windows
BROWSE NOWAIT      && Moving the record pointer
SELECT customer    && in the parent table/.DBF
BROWSE NOWAIT      && moves the pointer in the child

```

SET RELATION OFF

Purpose	Breaks an established relationship between the current table/.DBF and another open table/.DBF
Syntax	SET RELATION OFF INTO <expN> <expC>
See Also	RELATION(), SELECT, SET RELATION, SET SKIP, TARGET()
Description	SET RELATION OFF breaks a relationship between the current table/.DBF, which is the parent table/.DBF, and another open table/.DBF.
Clauses	INTO <expN> <expC> Specify the child table work area number with <expN> or its table alias with <expC>.

SET REPROCESS

Purpose	Specifies how many times or for how long FoxPro attempts to lock a file or record after an unsuccessful locking attempt
Syntax	SET REPROCESS TO <expN> [SECONDS] TO AUTOMATIC
See Also	FLOCK()

Description A record or file lock isn't always successful on the first attempt. Frequently, a record or file is locked by another user on the network. If an initial record or file lock is unsuccessful, SET REPROCESS controls whether additional attempts will be made to lock the record or file.

With SET PROCESS, you can specify either how many times additional attempts are made or for how long the attempts are made. An ON ERROR routine affects how unsuccessful lock attempts are handled.

Clauses **TO <expN> [SECONDS]**

The numeric expression <expN> specifies how many times FoxPro tries to lock a record or file after an unsuccessful attempt. The default value is 0.

SET REPROCESS TO 0

If <expN> is 0 (the default value), and you issue a command or function that attempts to lock a record or file, FoxPro attempts to lock the record or file indefinitely. The system message "Attempting to lock... Press Escape to Cancel" is displayed while FoxPro attempts to lock the record or file. The lock is placed and the system message is cleared if the record or file becomes available for locking while you wait. If a function attempted to place the lock, the function returns true (.T.).

If you press Escape in response to the system message, an appropriate alert is displayed (for example, "Record is in use by another"). If a function attempted to place the lock, the alert isn't displayed, and the function returns false (.F.).

If an ON ERROR routine is in effect and a command is attempting to lock the record or file, the ON ERROR routine takes precedence over additional attempts to lock the record or file. The ON ERROR routine is immediately executed. No additional record or file locks are attempted, and the system message isn't displayed.

If a function attempted to place the lock, no system message is displayed, an ON ERROR routine isn't executed, and the function returns false (.F.).

SET REPROCESS TO -1

If <expN> is -1, FoxPro attempts to lock the record or file indefinitely. You can't cancel the locking attempts by pressing the Escape key, and an ON ERROR routine isn't executed.

A system message ("Waiting for lock...") is displayed only if SET STATUS is ON.

If a lock has been placed by another user on the record or file you are attempting to lock, you must wait until the user releases the lock.

SET REPROCESS TO <expN> > 0 [SECONDS]

You can specify how many times or for how long FoxPro attempts to lock a record or file if <expN> is greater than 0. To specify the number of times, omit the key word SECONDS; to specify the length of time, include SECONDS.

For example, if <expN> is 30, FoxPro attempts to lock a record or file up to 30 times. If you also include the optional key word SECONDS (SET REPROCESS TO 30 SECONDS), FoxPro continuously attempts to lock a record or file for up to 30 seconds.

A system message ("Waiting for lock...") is displayed if SET STATUS is ON.

If an ON ERROR routine is in effect and attempts by a command to lock the record or file are unsuccessful, the ON ERROR routine is executed. However, if a function attempted the lock, an ON ERROR routine isn't executed and the function returns false (.F.).

If an ON ERROR routine isn't in effect and the record or file lock can't be placed, an appropriate alert is displayed (for example, "Record is in use by another"). If a function attempted to place the lock, the alert isn't displayed, and the function returns false (.F.).

TO AUTOMATIC

If you issue SET REPROCESS TO AUTOMATIC (or SET REPROCESS TO -2), FoxPro attempts to lock the record or file indefinitely. The system message “Attempting to lock... Press Escape to Cancel” is displayed while FoxPro attempts to lock the record or file. If the record or file becomes available to lock while you are waiting, the lock is placed and the system message is cleared. If a function is used to place the lock, the function returns true (.T.).

If an ON ERROR routine isn't in effect and you press Escape in response to the system message, an appropriate alert is displayed (for example, “Record is in use by another”). If a function attempted to place the lock, the alert isn't displayed and the function returns false (.F.).

If an ON ERROR routine is in effect and Escape is pressed, the ON ERROR routine is executed. If a function attempted to place the lock, an ON ERROR routine isn't executed and the function returns false (.F.).

For additional information about record and file locking and sharing tables/DBFs on a network, see the FoxPro in a Multi-User Environment chapter in the *FoxPro Developer's Guide*.

SET RESOURCE

Purpose	Specifies or updates a resource file
Syntax	SET RESOURCE ON OFF or SET RESOURCE TO [<file>]
See Also	SYS(2005)

Description Use SET RESOURCE to update your resource file or specify a different resource file.

The resource file is a FoxPro table that contains information on system and user-defined resources like keyboard macros, preferences, system window locations and sizes, diary entries, and so on.

For further information on resource files, refer to the Customizing FoxPro chapter in the FoxPro *Installation and Configuration* manual.

Clauses **ON**

Any changes made to the FoxPro environment is saved to the resource file if SET RESOURCE is ON.

The default for SET RESOURCE is ON.

OFF

Any changes made to the FoxPro environment aren't saved to the resource file if SET RESOURCE is OFF.

TO <file>

Use SET RESOURCE TO <file> to specify a resource file. You can save the FoxPro environment to a resource file other than the default FOXUSER.DBF resource file by specifying the name of the file with <file>.

Issue SET RESOURCE TO without a resource file name to open the default FOXUSER.DBF resource file. Issuing SET RESOURCE TO performs an implicit SET RESOURCE ON.

SET SAFETY

Purpose	Determines whether or not FoxPro displays a warning before overwriting an existing file
Syntax	SET SAFETY ON OFF
See Also	SET TALK

Description Use SET SAFETY to display a warning before you overwrite a file.

Clauses **ON**

If SET SAFETY is ON and you execute a command that overwrites a file, a dialog is first displayed asking if you want to overwrite the file.

The default for SET SAFETY is ON.

OFF

No dialog is displayed before a file is overwritten if SET SAFETY is OFF.

SET SCOREBOARD

Purpose	Specifies where FoxPro displays the status of the NumLock, CapsLock, and Insert keys
Syntax	SET SCOREBOARD ON OFF
Remarks	Included for backward compatibility — FoxPro for MS-DOS only
See Also	SET BRSTATUS

Description SET SCOREBOARD specifies where the status of the NumLock, CapsLock and Insert keys is displayed in FoxPro for MS-DOS.

SET SCOREBOARD is ignored in FoxPro for Windows.

Clauses **ON**

The status of the NumLock, CapsLock and Insert keys is displayed on line 0 if SET SCOREBOARD is ON and the status bar isn't displayed. Because the FoxPro system menu bar is displayed on line 0, you must press Shift+Ctrl+Alt to hide the menu bar to see the key status.

If the status bar is displayed, the status of the NumLock, CapsLock and Insert keys is displayed in the status bar.

OFF

If both SET SCOREBOARD and SET STATUS are OFF, the status of the NumLock, CapsLock and Insert keys isn't displayed. Issuing SET SCOREBOARD OFF clears the desktop.

The default for SET SCOREBOARD is OFF.

SET SEPARATOR

Purpose	Specifies the character that separates each group of three digits to the left of the decimal point
Syntax	SET SEPARATOR TO [<expC>]
See Also	SET DECIMALS

Description Use SET SEPARATOR to specify the numeric place separator in an @ ... SAY PICTURE clause to a different character.

The default separator is a comma (,).

Clause <expC>

Specify the character you want to use as the numeric place separator with <expC>.

Use SET SEPARATOR TO without <expC> to reset the separator to the default comma.

Example

```
SET SEPARATOR TO '#'  
x = 1000  
@ 2,2 GET x PICTURE '9,999.99'  
READ  
SET SEPARATOR TO && Reset the separator to a comma (,)
```

SET SHADOWS

Purpose	Places or removes shadows from all user-defined windows, popups created with DEFINE POPUP, and FoxPro system windows and popups
Syntax	SET SHADOWS ON OFF
Remarks	FoxPro for MS-DOS only
See Also	DEFINE WINDOW

Description In FoxPro for MS-DOS, SET SHADOWS globally displays or removes shadows from all system and user-defined windows, popups, dialogs and alerts.

SET SHADOWS is ignored in FoxPro for Windows.

A window created with DEFINE WINDOW isn't affected by SET SHADOWS unless the window was created with the SHADOW clause.

If you specify a SHADOW setting in your configuration file, it is overridden by the SHADOW setting of the color set that loads when you start FoxPro.

Clauses **ON**

Issue SET SHADOWS ON to display shadows. This is the default setting.

OFF

Issue SET SHADOWS OFF to remove shadows.

SET SKIP

Purpose	Creates one-to-many relationships between tables/.DBFs
Syntax	SET SKIP TO [<alias1> [, <alias2>] ...]
See Also	RELATION(), SET RELATION, SKIP

Description You can establish relations between tables/.DBFs open in different work areas with SET RELATION. When the record pointer is moved in the parent table/.DBF, the record pointer in the child table/.DBF is moved to the first corresponding record. Where the record pointer is moved in the child table/.DBF is determined by the relational expression in SET RELATION. A one-to-one relation is created — for each record in the parent table/.DBF, the record pointer moves to the first matching record in the child file. If a matching record can't be found in the child table/.DBF, the record pointer in the child table/.DBF is moved to the end of the file.

Frequently, a child table/.DBF contains multiple records that correspond to one record in the parent table/.DBF. SET SKIP lets you establish a one-to-many relationship between one record in the parent table/.DBF and multiple records in the child table/.DBF. When you skip through the parent file, the record pointer remains on the same parent record until the record pointer moves through all related records in the child file.

You establish a one-to-many relationship in a manner similar to the way you create a one-to-one relationship: First, you establish the relationship between the parent and child table/.DBF with SET RELATION; then, you issue SET SKIP to create a one-to-many relationship.

Clause **TO <alias1> [, <alias2>] ...**

If the parent table/.DBF is related to several child tables/.DBFs, include a list of table/.DBF aliases separated by commas to create a one-to-many relationship with each child table/.DBF specified with <alias1>, <alias2>, and so on. In commands that support a scope (DISPLAY, LIST, and so on), records in the parent table/.DBF are repeated for each corresponding record in the child table/.DBF.

Use SET SKIP TO without any additional arguments to remove the one-to-many relationship between tables/.DBFs in the current work area. The one-to-one relationship remains and can be removed with SET RELATION TO.

Example For an example using SET SKIP, see BROWSE in this manual.

SET SKIP OF

Purpose	Enables or disables a menu bar, menu pad, popup or popup bar for user-defined menus or the FoxPro system menu
Syntax	SET SKIP OF MENU <menu name> <expL> or SET SKIP OF PAD <pad name> OF <menu name> <expL> or SET SKIP OF POPUP <popup name> <expL> or SET SKIP OF BAR <expN> <system option name> OF <popup name> <expL>
See Also	CREATE MENU, DEFINE BAR, DEFINE MENU, DEFINE PAD, DEFINE POPUP, MENU — System Menu Names, SKPBAR()

Description Use SET SKIP OF to selectively enable or disable menus, menu pads, popups and popup options. You can use SET SKIP OF with the FoxPro menu system or user-defined menus and popups. For a complete listing of the names of the FoxPro system menu bar, menu pads, popups and popup options, see the MENU — System Menu Names topic in this chapter. (You can also use SYS(2013) to return the system menu names.)

If the logical expression <expL> evaluates to true (.T.), the menu, menu pad, popup or popup option included in SET SKIP OF is disabled (can't be selected) and is displayed in disabled colors. If <expL> evaluates to false (.F.), the menu, menu pad, popup or popup option is enabled (can be selected) and is displayed in enabled colors.

Clauses MENU <menu name> <expL>

SET SKIP OF MENU lets you enable or disable a menu bar. For example, the FoxPro system menu bar _MSYSMENU can be disabled with this command:

```
SET SKIP OF MENU _MSYSMENU .T.
```

It can be enabled with this command:

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

SET SKIP OF MENU can also be used to enable or disable user-defined menus created with DEFINE MENU.

PAD <pad name> OF <menu name> <expL>

SET SKIP OF PAD lets you enable or disable a FoxPro system menu pad or a user-defined menu pad created with DEFINE PAD.

For example, the FoxPro **Edit** menu pad can be disabled with this command:

```
SET SKIP OF PAD _MSM_EDIT OF _MSYSMENU .T.
```

It can be enabled with this command:

```
SET SKIP OF PAD _MSM_EDIT OF _MSYSMENU .F.
```

POPUP <popup name> <expL>

SET SKIP OF POPUP lets you enable or disable a FoxPro system popup or a user-defined popup created with DEFINE POPUP.

For example, the FoxPro **Edit** menu can be disabled with this command:

```
SET SKIP OF POPUP _MEDIT .T.
```

It can be enabled with this command:

```
SET SKIP OF POPUP _MEDIT .F.
```

**BAR <expN> | <system option name>
OF <popup name> <expL>**

SET SKIP OF BAR lets you enable or disable an option in a FoxPro system popup or a user-defined popup option created with DEFINE BAR.

For example, the **New...** option in the FoxPro **File** menu can be disabled with this command:

```
SET SKIP OF BAR _MFI_NEW OF _MFILE .T.
```

It can be enabled with this command:

```
SET SKIP OF BAR _MFI_NEW OF _MFILE .F.
```

SET SPACE

Purpose	Determines whether or not a space is displayed between fields or expressions when you use the ? or ?? command
----------------	---

Syntax	SET SPACE ON OFF
---------------	---------------------------

See Also	? ??
-----------------	--------

Description Issuing SET SPACE inserts or removes a space between fields or expressions when you use the ? or ?? commands.

Clauses **ON**

SET SPACE ON inserts a space between fields and expressions. This is the default setting.

OFF

SET SPACE OFF removes all spaces between fields or expressions and runs them together.

Example

```
x = 'John'
y = 'Smith'
SET SPACE OFF
? x,y
JohnSmith

SET SPACE ON
? x,y
John Smith
```

SET STATUS

Purpose	Displays or removes the status bar
Syntax	SET STATUS ON OFF TIMEOUT TO [<expN>]
Remarks	TIMEOUT clause — FoxPro for Windows only
See Also	SET BRSTATUS, SET MESSAGE, SET SCOREBOARD, SET STATUS BAR

Description SET STATUS displays or removes the status bar. In FoxPro for Windows you can use SET STATUS to specify how long messages are displayed in the Windows-style status bar.

Clauses **ON**

If SET STATUS is ON, the status bar is displayed showing the name of the program currently executing (if any) the alias of the active table/DBF, the current record pointer position, the number of records in the table/DBF, and the state of the Insert, NumLock and CapsLock keys. The record or file lock status is also shown in the status bar when the table/.DBF is opened for shared use. The status bar is updated each time you execute a command that changes status information.

SET STATUS ON displays a character-based status bar in FoxPro for Windows. Use SET STATUS BAR ON to displays a Windows-style status bar.

OFF

Use SET STATUS OFF to remove the status bar.

The default for SET STATUS is OFF.

TIMEOUT TO <expN> — FoxPro for Windows only

Include the TIMEOUT clause to specify how long messages are displayed in the Windows-style status bar in FoxPro for Windows.

The numeric expression <expN> specifies how many seconds messages are displayed in the Windows-style status bar. For example, if <expN> is 10, a message is displayed for 10 seconds and then cleared. If <expN> is 0, messages remain in the status bar until a key is pressed.

By default, messages are displayed in the Windows-style status bar for 5 seconds. To restore the default setting, issue the following command without the numeric expression <expN>:

SET STATUS TIMEOUT TO

SET STATUS BAR

Purpose	Displays a Windows-style status bar in FoxPro for Windows
Syntax	SET STATUS BAR ON OFF
Remarks	FoxPro for Windows only
See Also	SET BRSTATUS, SET MESSAGE, SET SCOREBOARD, SET STATUS

Description In FoxPro for Windows, you can display a Windows-style status bar at the bottom of the main FoxPro window. Use SET STATUS BAR to display the Windows-style status bar or to remove it from the main FoxPro window.

When you select a menu pad or menu option, information about the menu or option is displayed in the status bar. Additional information displayed in the status bar includes the alias of the table/.DBF in the current work area, the current record number, the number of records in the table/.DBF, the currently executing program, and the status of the Insert, NumLock, and CapsLock keys. You can place a clock in the Windows-style status bar with SET CLOCK STATUS.

When the Windows-style status bar is displayed, messages created in commands that support messages (DEFINE MENU, DEFINE POPUP, DEFINE BAR, and so on) always appear in the status bar. A message location specified with SET MESSAGE is ignored when the Windows-style status bar is displayed.

The Windows-style status bar is displayed by default. To prevent the Windows-style status bar from being displayed when FoxPro for Windows is started, place the following line in your CONFIG.FPW configuration file:

```
STATUS BAR = OFF
```

The character-based status bar familiar to FoxBASE+ and FoxPro for MS-DOS users can be displayed with SET STATUS ON. If the Windows-style status bar is displayed, issuing SET STATUS ON removes the Windows-style status bar from the main FoxPro window and displays the character-based status bar. Issuing SET STATUS OFF removes the character-based style status bar from the main FoxPro window.

When the character-based style status bar is displayed, messages created in commands that support messages (DEFINE MENU, DEFINE POPUP, DEFINE BAR, and so on) appear under the status bar. You can specify a different location for messages with SET MESSAGE.

SET STATUS BAR

Clauses ON

Issue SET STATUS BAR ON to display the Windows-style status bar.

OFF

Issue SET STATUS BAR OFF to remove the Windows-style status bar from the main FoxPro window.

SET STEP

Purpose	Opens the Trace window for debugging
Syntax	SET STEP ON OFF
See Also	SET ECHO

Description SET STEP ON opens the Trace window and suspends program execution. Choosing **Step** in the Trace window menu bar executes a single program line. Choosing **Resume** resumes execution without stepping through the program one line at a time.

You can pass parameters to a program and then trace its execution by following these steps:

- Open the Trace window.
- From the **Program** menu in the Trace window, choose the **Open...** option and choose the program to trace.
- Set a breakpoint on the first executable line of the program.
- In the Command window, DO the program WITH the parameters.

See the FoxPro *User's Guide* for more information on the Trace window.

SET STICKY

Purpose	Specifies how the mouse displays menus in the FoxPro menu system
Syntax	SET STICKY ON OFF
Remarks	FoxPro for MS-DOS only
See Also	SET SYSMENU

Description SET STICKY controls the display of FoxPro system menu popups. SET STICKY affects only selections you make with the mouse.

SET STICKY is ignored in FoxPro for Windows.

Clauses **ON**

If SET STICKY is ON and you select a menu pad with the mouse, the popup remains on the desktop until you choose an option, press Escape or click outside the popup.

The default for SET STICKY is ON.

OFF

If SET STICKY is OFF and you select a system menu pad with the mouse, the popup remains on the desktop only while the mouse button is pressed.

SET SYSMENU

Purpose	Enables or disables access to the FoxPro system menu bar during program execution
Syntax	SET SYSMENU ON OFF AUTOMATIC TO [<menu popup list TO [<menu pad list>] TO [DEFAULT] SAVE NOSAVE
See Also	MENU — System Menu Names, POP MENU, PUSH MENU, RELEASE, SET STICKY, SYS(2013)
Description	SET SYSMENU controls the main FoxPro menu bar during program execution and lets you selectively remove and restore menu pads and popup options from the main FoxPro menu system.
Clauses	ON The main FoxPro menu bar is accessible during program execution when SET SYSMENU is ON and FoxPro is waiting for keyboard input (during commands such as BROWSE, READ, MODIFY COMMAND, and so on). The main FoxPro menu bar isn't displayed, but you can display and make it accessible by pressing the Alt or F10 key or by double-clicking the right mouse button. SET SYSMENU is ON by default. OFF If SET SYSMENU is OFF, you can't access the main FoxPro menu bar during program execution. AUTOMATIC Include AUTOMATIC to make the main FoxPro menu bar visible during program execution. The menu bar is accessible as appropriate, and menu options are enabled and disabled as appropriate for the current command.

TO [<menu popup list>]**TO [<menu pad list>]**

You can add a subset of menu popups to the main FoxPro menu bar by including a menu popup list or pad list. The menu popup or pad list can contain any combination of menu popup or pad names separated by commas. The names for the menu popups and pads are listed in the Menu — System Menu Names topic in this chapter.

For example, issuing the following command removes all menus from the main FoxPro menu bar except the **File** and **Window** menus.

```
SET SYSMENU TO _MFILE, _MWINDOW
```

You can't include the names of options that appear in the main FoxPro menus. Use **RELEASE BAR** to specify popup options.

TO [DEFAULT]

Issuing SET SYSMENU TO without any arguments disables the main FoxPro menu bar. If you modify the main menu bar or its popups, you can restore it to the default configuration by issuing SET SYSMENU TO DEFAULT. You can specify a default configuration with SET SYSMENU SAVE.

SAVE

Issuing SET SYSMENU SAVE makes the current menu system the default configuration. If you modify the menu system after issuing SET SYSMENU SAVE, you can restore the previous configuration by issuing SET SYSMENU TO DEFAULT.

NOSAVE

Issuing SET SYSMENU NOSAVE returns the main menu bar and its popups to the standard FoxPro startup configuration.

Example

Type the following commands in the Command window.

```
SET SYSMENU SAVE
SET SYSMENU TO _msm_file, _msm_edit
SET SYSMENU TO
SET SYSMENU TO DEFAULT
```

SET TALK

Purpose	Determines whether or not FoxPro displays command results
Syntax	SET TALK ON OFF WINDOW [<window name>] NOWINDOW
See Also	DEFINE WINDOW, SET NOTIFY, SET ODOMETER, SYS(103), _TALLY

Description Some table/DBF processing commands return information about their status while they execute. This status information is often referred to as “talk.” These commands include the following:

APPEND FROM	COUNT	PACK	SORT
AVERAGE	DELETE	REINDEX	SUM
CALCULATE	INDEX	REPLACE	TOTAL
COPY TO	SELECT – SQL		

Talk can be directed to the desktop, the system message window or a user-defined window in FoxPro for MS-DOS. In FoxPro for Windows, talk can be directed to the the main FoxPro window, the Windows-style status bar or a user-defined window. Talk can also be turned off.

Clauses **ON**

Talk can be turned on with SET TALK ON. If SET TALK is OFF and is then changed to ON, the talk is directed to the same location (the desktop, the main FoxPro window, the status bar, the system message window or a user-defined window) it was sent to before you issued SET TALK OFF. ON is the default setting.

OFF

Use SET TALK OFF to prevent talk from being sent to the desktop, the main FoxPro window, the status bar, the system message window or a user-defined window.

WINDOW [<window name>]

In FoxPro for MS-DOS, talk is displayed in the system message window if you issue SET TALK WINDOW and SET NOTIFY is ON. The system message window is removed when you press a key or move the mouse.

You can direct talk to a user-defined window by specifying a window name with <window name>. You must create the user-defined window before you can direct talk to the window. A user-defined window doesn't have to be active or visible to direct talk to it. In FoxPro for MS-DOS, talk is directed to the FoxPro system window if the window you specify doesn't exist.

NOWINDOW

To direct talk to the desktop or main FoxPro window, issue SET TALK NOWINDOW.

Reporting Interval

The TALK reporting interval can be specified with SET ODOMETER. The default value for SET ODOMETER is 100. Note that execution speed can be degraded when SET TALK is ON because the desktop, main FoxPro window or user-defined window must be frequently updated. If you just need the total number of records processed by a command, you can issue SET TALK OFF and display the _TALLY system memory variable once the command has completed executing.

Example

The following example demonstrates how you can save the TALK setting without generating talk on the desktop, main FoxPro window or user-defined window.

```
IF SET('TALK') = 'ON'    && TALK handled as a special case.
    SET TALK OFF        && Turn TALK OFF
    savetalk = 'ON'     && TALK was ON, save the setting
ELSE    && TALK is OFF
    savetalk = 'OFF'    && TALK was OFF, save the setting
ENDIF
```

*** Additional commands ***

```
SET TALK &savetalk      && Restore original setting
```


SET TEXTMERGE

Purpose	Enables or disables the evaluation of fields, functions and expressions that are surrounded by text merge delimiters
Syntax	SET TEXTMERGE [ON OFF] [TO [<file>] [ADDITIVE]] [WINDOW <window name>] [SHOW NOSHOW]
See Also	\, \\\, _PRETEXT, SET TEXTMERGE DELIMITERS, _TEXT
Description	\, \\\ and TEXT ... ENDTEXT are used to merge text with the contents of tables/DBFs, memory variables, array elements and the results of functions and expressions. If a field, memory variable, array element, function or expression is surrounded by text merge delimiters (by default, << and >>), it can be evaluated and merged with text. This text merge capability lets you produce letters, programs and templates that create programs. SET TEXTMERGE determines how fields, memory variables, array elements, functions or expressions surrounded by the text merge delimiters are evaluated. SET TEXTMERGE also lets you direct text merge output to the desktop in FoxPro for MS-DOS, main FoxPro window in FoxPro for Windows, a user-defined window or a file. For additional information about merging text, see the Text Merge chapter in the FoxPro <i>Developer's Guide</i>.
Clauses	ON If SET TEXTMERGE is ON, any fields, memory variables, array elements and functions and expressions surrounded by the text merge delimiters are evaluated and output when placed after the \ or \\\ or between TEXT and ENDTEXT.

Example

The following short program example (run on the Fourth of July) demonstrates how the contents of the variable MTODAYDATE and the DATE() and TIME() functions are evaluated when SET TEXTMERGE is ON. MTODAYDATE, DATE() and TIME() are evaluated because they are enclosed by the text merge delimiters and SET TEXTMERGE is ON.

```
CLEAR
SET TALK OFF
STORE 'Today is: ' TO mtodaydate
SET TEXTMERGE ON
\<<mtodaydate>>
\\<<DATE( )>>
\The time is:
\\ <<TIME( )>>
```

Here is the output from the program above:

```
Today is: 07/04/92
The time is: 10:55:19
```

OFF

If SET TEXTMERGE is OFF, any fields, memory variables, array elements, functions and expressions are literally output along with the text merge delimiters surrounding them. Note the difference in output when SET TEXTMERGE is OFF in the previous example.

```
CLEAR
SET TALK OFF
STORE 'Today is: ' TO mtodaydate
SET TEXTMERGE OFF
\<<mtodaydate>>
\\<<DATE( )>>
\The time is:
\\ <<TIME( )>>
```

Here is the output from this program:

```
<<mtodaydate>><<DATE( )>>
The time is: <<TIME( )>>
```

The default for TEXTMERGE is OFF.

TO [<file>] [ADDITIVE]

By default, output from \, \\ and TEXT ... ENDTEXT is sent to the desktop or main FoxPro window. You can also direct the output to a text file by including FILE and specifying a file name with <file>. If a file with that name doesn't exist, a new file is created. If a file with the same name already exists, you are given the option of overwriting the existing file if SET SAFETY is ON.

The text file is opened as a low-level file, and its file handle is stored to the _TEXT system memory variable. You can close the file by issuing SET TEXTMERGE TO without any additional arguments. If the file handle of another file was previously stored in _TEXT, that file is closed.

You can append output from \, \\ and TEXT ... ENDTEXT to the end of an existing file by including the ADDITIVE key word.

See the _TEXT system memory variable in the next chapter for further information on directing text merge output to a file.

WINDOW <window name>

Output from \, \\ and TEXT ... ENDTEXT is directed to the desktop or main FoxPro window by default. You can redirect output to a window by including WINDOW. Specify the name of the window you want to redirect output to with <window name>. The window must first be created with DEFINE WINDOW before output can be sent to it. The window doesn't have to be active or visible.

SHOW | NOSHOW

Output generated by \, \\ and TEXT ... ENDTEXT is sent to the desktop, the main FoxPro window or a user-defined window by default. You can suppress output to the desktop, the main FoxPro window or a user-defined window by including NOSHOW. To display text merge output again, include SHOW in SET TEXTMERGE. The default is SHOW.

Memo Fields and Text Merge

Memo fields can be used to nest merged text. If a memo field contains field names, memory variables, functions or expressions surrounded by the current text merge delimiters, they are evaluated and output with the contents of the memo field. The memo field name must also be enclosed in text merge delimiters.

SET TEXTMERGE DELIMITERS

Purpose	Specifies the text merge delimiters
Syntax	SET TEXTMERGE DELIMITERS [TO <expC1> [, <expC2>]]
See Also	\, \\\, _PRETEXT, SET TEXTMERGE, _TEXT

Description \, \\\ and TEXT ... ENDTEXT are used to merge text with the contents of tables/DBFs, memory variables, array elements and the results of functions and expressions. If a field, memory variable, array element, function or expression is surrounded by text merge delimiters, it can be evaluated and merged with text. This text merge capability lets you produce letters, programs and templates that create programs.

The default text merge delimiters are double angle brackets (<< and >>). You can specify a different set of text merge delimiters with SET TEXTMERGE DELIMITERS.

The current delimiters can be displayed with DISPLAY STATUS.

Clause **TO <expC1> [, <expC2>]**

If you specify only one delimiter with <expC1>, both the left and right delimiters are set to <expC1>. If you specify both delimiters by including <expC1> and <expC2>, the left delimiter is set to <expC1> and the right delimiter is set to <expC2>.

If you issue SET TEXTMERGE DELIMITERS without any additional arguments, the default delimiters are restored.

SET TOPIC

Purpose	Specifies the initial help topic when you invoke the FoxPro help system
Syntax	SET TOPIC TO [<expC> <expL>]
See Also	HELP, SET HELP, SET HELPFILTER

Description SET TOPIC TO specifies the criteria used to select help topics.

Clauses <expC>

<expC> is the name of the help topic you want to display.

<expL>

You can also specify the help topic or topics to open based on the value of the logical expression <expL>. This is useful for creating your own online help. See the Customizing Help chapter in the FoxPro *Developer's Guide* for more information on creating your own help system.

SET TRBETWEEN

Purpose	Enables or disables tracing between breakpoints in the Trace window
Syntax	SET TRBETWEEN ON OFF
See Also	ACTIVATE WINDOW, SET ECHO, SET STEP
Description	The Trace window displays the source code for a program as the program executes. The executing program line is highlighted. The Trace window can also be opened by issuing ACTIVATE WINDOW TRACE, SET ECHO ON or SET STEP ON.
Clauses	ON Every line of program code is displayed and highlighted in the Trace window as it is executed when SET TRBETWEEN is ON. Issuing SET TRBETWEEN ON is identical to turning on the Trace Between Breaks option in the Trace window Program menu. The default for TRBETWEEN is ON. OFF Only the last line at which program execution was paused is highlighted in the Trace window when SET TRBETWEEN is OFF. Issuing SET TRBETWEEN OFF is identical to turning off the Trace Between Breaks option.

SET TYPEAHEAD

Purpose	Specifies the maximum number of characters that can be stored in the typeahead buffer
Syntax	SET TYPEAHEAD TO <expN>
See Also	INKEY(), ON KEY

Description SET TYPEAHEAD controls the number of characters that are stored in the typeahead buffer. The typeahead buffer can store as many as 128 characters until they are ready to be processed.

Clause **<expN>**
Specify the maximum number of characters to store in the typeahead buffer with <expN>.

No characters are held in the typeahead buffer if SET TYPEAHEAD is 0. Issuing SET TYPEAHEAD TO 0 deactivates INKEY() and ON KEY.

The default value for SET TYPEAHEAD is 20.

SET UDFPARMS

Purpose	Specifies whether FoxPro passes parameters to a user-defined function (UDF) by value or by reference
Syntax	SET UDFPARMS TO VALUE REFERENCE
See Also	DO, PARAMETERS, PARAMETERS(), PROCEDURE, User-Defined Functions

Description SET UDFPARMS determines whether variables are passed to a UDF by value or by reference.

By default, variables are passed to UDFs by value. (Variables are always passed to procedures by reference.) When a variable is passed by value, the variable’s value can be changed in the procedure or UDF, but the variable’s original value in the calling program isn’t changed.

When a variable is passed by reference and the UDF changes the value of the passed variable, the variable’s original value in the calling program is also changed.

Example The following example illustrates the difference between passing variables by value and by reference.

*** Pass memory variable by value ***

```
CLEAR
SET TALK OFF
WAIT 'Press a key to pass by value' WINDOW
SET UDFPARMS TO VALUE
STORE 1 TO X
*** The value of x is unchanged ***
@ 2,2 SAY 'UDF value: ' + STR(plusone(X))
@ 4,2 SAY 'Value of X: ' + STR(X)
```

*** Pass memory variable by reference ***

```
WAIT 'Press a key to pass by reference' WINDOW
CLEAR
SET UDFPARMS TO REFERENCE
STORE 1 TO X
*** The value of x is changed ***
@ 2,2 SAY 'UDF value: ' + STR(plusone(X))
@ 4,2 SAY 'Value of X: ' + STR(X)
```



```
*** This is a UDF that adds one to a number ***
```

```
FUNCTION plusone
PARAMETER Z
Z = Z + 1
RETURN Z
```

```
*** End of UDF ***
```

You can force parameters to be passed by value or reference, regardless of the setting of UDFPARMS. Enclose a variable in parentheses to force the variable to be passed by value. Preface the variable with an @ symbol to force the variable to be passed by reference.



Entire arrays can be passed to a procedure or UDF. The entire array is passed if you issue SET UDFPARMS TO REFERENCE or preface the array name with @. The first element of the array is passed by value if you issue SET UDFPARMS TO VALUE or enclose the array name by parentheses. Array elements are always passed by value.

Example

Here is the above example with variables passed by value and reference through the use of parentheses and @, respectively:

```
*** Pass memory variable by value ***
```

```
CLEAR
SET TALK OFF
WAIT 'Press a key to pass by value' WINDOW
STORE 1 TO X
@ 2,2 SAY 'UDF value: ' + STR(plusone((X)))
@ 4,2 SAY 'Value of X: ' + STR(X)
```

```
*** Pass memory variable by reference ***
```

```
WAIT 'Press a key to pass by reference' WINDOW
CLEAR
STORE 1 TO X
@ 2,2 SAY 'UDF value: ' + STR(plusone(@X))
@ 4,2 SAY 'Value of X: ' + STR(X)
```

```
*** This is a UDF that adds one to a number ***
```

```
FUNCTION plusone
PARAMETER Z
Z = Z + 1
RETURN Z
*** End of UDF ***
```

SET UNIQUE

Purpose	Specifies whether or not records with duplicate index key values appear in an index file
Syntax	SET UNIQUE ON OFF
See Also	INDEX, REINDEX

Description SET INDEX determines whether records with duplicate index keys are maintained in an index file.

An index file retains its UNIQUE status when you REINDEX. See INDEX and REINDEX in this chapter for more information.

Clauses **ON**

Any record with a duplicate key isn't included in the index file if SET UNIQUE is ON. Only the first record with the duplicate key is included in the index file.

OFF

If SET UNIQUE is OFF, records with duplicate index keys are included in the index file.

The default for SET UNIQUE is OFF.

SET VIEW

Purpose	Opens or closes the View window or restores the FoxPro environment from a view file
Syntax	SET VIEW ON OFF SET VIEW TO <file> ?
See Also	CREATE VIEW, SET

Description Issue SET VIEW ON to open the View window. The View window is one of the most useful FoxPro tools. It provides an easy way to open tables/.DBFs, establish relations and set or change many FoxPro options. See the FoxPro *User's Guide* for more information on the View window.

See CREATE VIEW in this chapter for more information on view files.

Clauses **ON**

Include ON to open the View window.

OFF

Include OFF to close the View window.

The default for SET VIEW is OFF.

TO <file> | ?

Issuing SET VIEW TO <file> restores the FoxPro environment to the state it was in when the view file specified with <file> was created. View files are created with CREATE VIEW.

SET WINDOW OF MEMO

Purpose	Specifies a window in which memo fields are edited
Syntax	SET WINDOW OF MEMO TO <window name>
Remarks	Included for backward compatibility. Use @ ... EDIT instead.
See Also	@ ... EDIT, MODIFY MEMO

Description	Use SET WINDOW to edit memo fields within a user-defined window. You can then open the memo editing window by issuing @ ... GET, APPEND, BROWSE, CHANGE, EDIT, or MODIFY MEMO. The user-defined window in which the memo editor is placed must be defined before you include it in SET WINDOW OF MEMO. To edit a memo field within the specified window, move the cursor to the memo field and press Ctrl+Home, Ctrl+PgUp, Ctrl+PgDn, or double-click the memo field.
Clause	<window name> Specify the name of the window in which you want to edit the memo fields with <window name>.

SET()

Purpose	Returns the status of a SET command
Syntax	SET(<expC> [, 1])
Parameters	<expC> A SET command 1 An option to return additional information
Returns	Character or Numeric
See Also	DISPLAY STATUS, LIST STATUS, SET, SYS(2001)

Description This function returns the status of various SET commands. Not all SET commands can be specified with the SET() function; those that can are listed in the table at the end of this topic.

SET() recognizes the four-character abbreviation for all FoxPro SET key words. For example, STAT and PRIN can be used for SET STATUS and SET PRINTER, respectively.

The SET() function is identical to SYS(2001).

Parameters <expC>

SET() returns information about the SET command you specify with the character expression <expC>. The current setting of the specified command is returned as a character or numeric string.

1

Additional information about a SET command can be returned if the 1 argument is included. Including the optional argument 1 doesn't return additional information for all the SET commands, just those listed with a 1 in the table at the end of this topic.

Example

The following program example uses SET() to save the TALK setting without creating talk on the screen or in a window.

```
IF SET('TALK') = 'ON'      && TALK handled as a special case.
    SET TALK OFF          && Turn TALK OFF
    savetalk = 'ON'       && TALK was ON, save the setting
ELSE                      && TALK is OFF
    savetalk = 'OFF'      && TALK was OFF, save the setting
ENDIF
SET TALK OFF
SET TALK &savetalk        && Restore original TALK setting
```

SET Command	Value Returned
ALTERNATE	ON or OFF
ALTERNATE, 1	Alternate file
ANSI	ON or OFF
AUTOSAVE	ON or OFF
ANSI	ON or OFF
AUTOSAVE	ON or OFF
BELL	ON or OFF
BLINK	ON or OFF
BLOCKSIZE	<expN>
BRSTATUS	ON or OFF
CARRY	ON or OFF
CENTURY	ON or OFF
CLEAR	ON or OFF
CLOCK	ON or OFF
CLOCK, 1	<row> and <column>
COLOR	Color attribute
COMPATIBLE	ON or OFF
CONFIRM	ON or OFF
CONSOLE	ON or OFF
CURRENCY	LEFT or RIGHT
CURRENCY, 1	<expC>
CURSOR	ON or OFF
DATE	<expC>
DEBUG	ON or OFF
DECIMALS	0 through 18
DEFAULT	Default drive; use SYS(2003) for directory
DELETED	ON or OFF
DELIMITERS	ON or OFF
DELIMITERS, 1	<expC>
DEVELOPMENT	ON or OFF
DEVICE	PRINTER or SCREEN
DISPLAY	Not supported, use SYS(2006)
DOHISTORY	ON or OFF
ECHO	ON or OFF
ESCAPE	ON or OFF
EXACT	ON or OFF
EXCLUSIVE	ON or OFF
FIELDS	ON or OFF
FIELDS, 1	<field1>, <field2> ...
FILTER	ON or OFF
FIXED	ON or OFF
FULLPATH	ON or OFF
HEADING	ON or OFF
HELP	ON or OFF
HELP, 1	<file>
HISTORY	ON or OFF
HOURS	12 or 24
INTENSITY	ON or OFF

SET Command	Value Returned
KEYCOMP	DOS or WINDOWS
LOCK	ON or OFF
LOGERRORS	ON or OFF
MACKEY	<expC>
MARGIN	0 to 254
MARK	Mark character
MEMOWIDTH	8 to 256
MENU	ON or OFF
MESSAGE	<expN>
MESSAGE, 1	<expC>
MOUSE	ON or OFF
MULTILOCKS	ON or OFF
NEAR	ON or OFF
NOTIFY	ON or OFF
ODOMETER	1 to 32,767
OPTIMIZE	ON or OFF
ORDER	Index file or index tag (with ASCENDING or DESCENDING clause if included)
PALETTE	ON or OFF
PATH	<path>
POINT	<expC>
PRINTER	ON or OFF
PRINTER, 1	<file> or <port>
PROCEDURE	<file>
REFRESH	0 through 3,600
REPROCESS	-2 to 32,000
RESOURCE	ON or OFF
RESOURCE, 1	<file>
SAFETY	ON or OFF
SCOREBOARD	ON or OFF
SEPARATOR	<expC>
SHADOWS	ON or OFF
SPACE	ON or OFF
STATUS	ON or OFF
STATUS BAR	ON or OFF
STEP	ON or OFF
STICKY	ON or OFF
SYSMENU	ON, OFF or AUTOMATIC
TALK	ON or OFF
TALK, 1	WINDOW, NOWINDOW or window name
TEXTMERGE	ON or OFF
TEXTMERGE, 1	Textmerge delimiters
TOPIC	<expC> or <expL>
TYPEAHEAD	0 to 32,000
UDFPARMS	VALUE or REFERENCE
UNIQUE	ON or OFF
VIEW	Not supported

SHOW GET

Purpose	Redisplays the @ ... GET object that represents the specified memory variable
Syntax	SHOW GET <var> [, <expN1> [PROMPT <expC>]] [ENABLE DISABLE] [LEVEL <expN2>] [COLOR SCHEME <expN3> COLOR <color pair list>]
Remarks	RGB color pairs — FoxPro for Windows only
See Also	@ ... GET, CLEAR READ, _CUROBJ, OBJNUM(), RDLEVEL(), READ, SHOW GETS, SHOW OBJECT

Description SHOW GET redisplay a single control (a check box, a field, an invisible button, a push or radio button, a list, a popup, a spinner or a text editing region). When the value in an @ ... GET field changes (the cursor moves to a new record, for example), SHOW GET updates the value displayed in the @ ... GET field. When a control is redisplayed, it can be enabled or disabled. SHOW GET can also be used to redisplay an individual button within a set of invisible, radio or push buttons. You can also change the prompts of check boxes and buttons with SHOW GET.

SHOW GET is typically used in a routine executed in a control level VALID or WHEN clause or a READ-level ACTIVATE or DEACTIVATE clause.

SHOW GET vs. SHOW GETS and SHOW OBJECT

Individual controls are redisplayed with SHOW GET or SHOW OBJECT. All controls created with @ ... GET are redisplayed with SHOW GETS. SHOW GETS executes the READ LEVEL SHOW routine. SHOW GET and SHOW OBJECT don't.

SHOW OBJECT is similar to SHOW GET except that SHOW OBJECT references controls by their object number, and SHOW GET references controls by <var>, the memory variable, array element or field specified when the control is created.

See OBJNUM() or _CUROBJ for more information on control numbering.

Clauses **<var>**

<var> is the memory variable, array element or field specified when the control was created.

Example

In the following example, three radio buttons with the prompts Apples, Oranges and Lemons are created. In this example, <var> is a memory variable named FRUIT that initializes the radio buttons (since FRUIT equals 2, the second button is initially selected). The button number is stored to FRUIT when you choose a button.

```
CLEAR
STORE 2 TO fruit
@ 4,2 GET fruit PICTURE '@*R Apples;Oranges;Lemons'
READ CYCLE
```

<expN1>

<expN1> is the number of the button you want to redisplay within a set of invisible, radio, or push buttons. A button's number is determined by the order in which the button prompts are created. In the example above, Apples is created first and is button number one. The second button created is Oranges; it is button number two. The last button, Lemons, is button three.

The following program creates the same buttons as in the previous example. However, when one of the radio buttons is chosen, the NOORANGE VALID routine is executed. The routine disables the Oranges button with SHOW GET and DISABLE.

```
CLEAR
STORE 2 TO fruit
@ 4,2 GET fruit PICTURE '@*R Apples;Oranges;Lemons' VALID noorange( )
READ CYCLE
```

```
FUNCTION noorange
SHOW GET fruit,2 DISABLE  && Disables the second button
_CUROBJ = 1
RETURN .T.
```

PROMPT <expC>

You can replace the prompt for an individual push button, radio button or check box with another prompt by including PROMPT <expC>. <expC> replaces the original button or check box prompt. You can also change the button or check box attributes (which determine whether the control is enabled or disabled, what hot keys are assigned, and whether it is the default or Escape choice) by specifying the appropriate special characters with <expC>.

Example

In the following example, a check box is created. When the check box is chosen, the VALID clause routine Newprompt is executed. The routine changes the check box prompt and the hot key.

```
CLEAR
STORE 1 TO check
@ 4,2 GET check FUNCTION '*C \<Checked' SIZE 1,13 VALID NEWPROMPT( )
READ CYCLE

FUNCTION newprompt
IF check = 0
    SHOW GET check,1 PROMPT '\<Unchecked'
ELSE
    SHOW GET check,1 PROMPT '\<Checked'
ENDIF
RETURN .T.
```

ENABLE | DISABLE

Including **ENABLE** allows the specified control to be selected or chosen. Including **DISABLE** prevents the control from being selected or chosen and displays the control in disabled colors.

LEVEL <expN2>

You can nest **READS** by issuing @ ... **GET** and a **READ** in a routine called in an active **READ**. **READS** can be nested to five levels.

If you don't include a **LEVEL** clause, **SHOW GET** defaults to the current **READ** level. Include a **LEVEL** clause to redisplay a control on a level other than the current level. The numeric expression <expN2> can assume the value 1, 2, 3, 4 or 5 corresponding to the **READ** level of the control to redisplay. Use **RDLEVEL()** to return the current **READ** level.

COLOR SCHEME <expN3>**| COLOR <color pair list>**

Controls can be redisplayed in the colors you specify. You can specify a color scheme or a color pair list. Refer to @ ... **GET** and the Controls section in this chapter for information about control colors.

SHOW GETS

Purpose	Redisplays all @ ... GET controls
Syntax	SHOW GETS [ENABLE DISABLE] [LEVEL <expN1>] [OFF ONLY] [WINDOW <window name>] [LOCK] [COLOR SCHEME <expN2> COLOR <color pair list>]
Remarks	RGB color pairs — FoxPro for Windows only
See Also	@ ... GET, CLEAR READ, _CUROBJ, OBJNUM(), RDLEVEL(), READ, SHOW GET, SHOW OBJECT, SYS(2016)

Description SHOW GETS redisplayes all controls (check boxes; fields; invisible; push or radio buttons; lists; popups; spinners or text editing regions). When values in @ ... GET fields change (the cursor moves to a new record, for example), SHOW GETS updates the values of the @ ... GET fields. Controls can be enabled or disabled when they are redisplayed.

A routine containing SHOW GETS can be used to redisplay @ ... GET controls. A GET LEVEL VALID or WHEN, or a READ LEVEL ACTIVATE or DEACTIVATE routine can contain SHOW GETS.

READ supports a SHOW clause that executes a routine whenever SHOW GETS is issued. A READ SHOW routine is often used to refresh output of @ ... SAYS. When SHOW GETS is issued, the SHOW routine is executed all the controls are redisplayed.

Issuing SHOW GETS resets UPDATED() to false (.F.).

SHOW GETS vs. SHOW GET and SHOW OBJECT

All controls created with @ ... GET are redisplayed with SHOW GETS. Individual controls are redisplayed with SHOW GET or SHOW OBJECT. SHOW GETS executes the READ-level SHOW routine. SHOW GET and SHOW OBJECT don't.

SHOW OBJECT is similar to SHOW GET except that SHOW OBJECT references controls by their object number, and SHOW GET references controls by <var>, the memory variable, array element or field specified when the control is created. See OBJNUM() or _CUROBJ for more information on control numbering.

Clauses ENABLE | DISABLE

Including **ENABLE** allows all controls to be selected or chosen. Including **DISABLE** prevents all controls from being selected or chosen and redisplay them in disabled colors.

LEVEL <expN1>

You can nest **READs** by issuing @ ... **GETs** and a **READ** in a routine called in an active **READ**. **READs** can be nested to five levels.

If you don't include a **LEVEL** clause, **SHOW GETS** defaults to the current **READ** level. Include a **LEVEL** clause to redisplay controls on a level other than the current level. The numeric expression <expN2> can assume the value 1, 2, 3, 4 or 5 corresponding to the **READ** level of the controls to redisplay. **RDLEVEL()** returns the current **READ** level.

OFF | ONLY

When **OFF** is included, **SHOW GETS** executes only the **READ SHOW** routine and doesn't refresh any controls. When **ONLY** is included, **SHOW GETS** refreshes only the controls and doesn't execute the **READ SHOW** routine.

WINDOW <window name>

Controls can be placed in different windows and activated with a single **READ**. The **WINDOW** clause lets you redisplay controls in a specific window without redisplaying controls in all windows. Specify the window name for the controls to redisplay with <window name>. **SYS(2016)** returns an asterisk (*) if you issue **SHOW GETS** without a **WINDOW** clause. **SYS(2016)** returns the name of the window if you issue **SHOW GETS** with a **WINDOW** clause.

LOCK

When a **READ** is issued and a table/.DBF is opened for shared access on a network, FoxPro may attempt to lock all records involved in the **READ**. If **READ** or **READ LOCK** is issued, FoxPro attempts to lock all the records involved in the **READ**. If **READ NOLOCK** is issued, FoxPro makes no attempt to lock the records and makes all fields involved in the **READ** read-only.

SHOW GETS assumes the record locking behavior of the **READ**. If **READ** or **READ LOCK** is issued, **SHOW GETS** attempts to lock the records involved in the **READ**. If **READ NOLOCK** is issued, **SHOW GETS** makes no attempt to lock the records, and all fields involved in the **READ** are made read-only.

If LOCK is included in SHOW GETS, SHOW GETS always attempts to lock the records involved in the READ. Issuing SHOW GETS LOCK lets you override READ NOLOCK.

See READ in this chapter for further information on how record locking is attempted during a READ.

COLOR SCHEME <expN2> | COLOR <color pair list>

Controls can be redisplayed in the colors you specify. You can specify a color scheme or a color pair list. Refer to @ ... GET and the Controls section in this chapter for information about control colors.

Example

The following example creates a data entry window and a control window that uses DO to call procedures that move you through CUSTOMER.DBF.

```
CLOSE ALL
SET TALK OFF
DEFINE WINDOW customer FROM 3,3 TO 13,57 ;
    FLOAT SHADOW DOUBLE
DEFINE WINDOW panel2 FROM 2, 61 TO 14,74 ;
    FLOAT SHADOW DOUBLE COLOR SCHEME 5
USE customer
ACTIVATE WINDOW customer NOSHOW
@ 3,3 SAY 'Company'
@ 3,14 GET customer.company SIZE 1,35
@ 5,3 SAY 'Contact'
@ 5,14 GET customer.contact SIZE 1,35
@ 7,3 SAY 'City/State'
@ 7,14 GET customer.city SIZE 1,21
@ 7,36 GET customer.state
@ 7,39 GET customer.zip
@ 0,3 SAY 'Customer #'
@ 0,15 SAY cno SIZE 1,6
@ 0,36 SAY 'Record #'
@ 0,46 SAY RECNO( ) SIZE 1,3
ACTIVATE WINDOW panel2
@ 1,2 GET act PICTURE;
    '@*VN Top;Prior;Next;Bottom;\?Quit' ;
    SIZE 1,8,1 DEFAULT 0 VALID actvalid( )
READ CYCLE SHOW readshow( ) COLOR ,R/BG
RELEASE WINDOWS customer,panel2
```

SHOW GETS

```
FUNCTION actvalid
DO CASE
CASE act = 1
    GO TOP
CASE ACT = 2
    DO go_prev
CASE ACT = 3
    DO go_next
CASE act = 4
    GO BOTTOM
CASE act = 5
    CLEAR READ
ENDCASE
SHOW GETS
RETURN 0
```

```
FUNCTION readshow
STORE WOUTPUT( ) TO currwind
IF SYS(2016) = 'customer' OR SYS(2016) = '*'
    ACTIVATE WINDOW customer SAME
    @ 0,15 SAY cno SIZE 1,6
    @ 0,46 SAY RECNO( ) SIZE 1,3
ENDIF
IF NOT EMPTY(currwind)
    ACTIVATE WINDOW (currwind) SAME
ENDIF
RETURN .T.
PROCEDURE go_prev
SKIP -1
IF BOF( )
    GO TOP
ENDIF
RETURN

PROCEDURE go_next
SKIP 1
IF EOF( )
    GO BOTTOM
ENDIF
RETURN
```

SHOW MENU

Purpose	Displays a user-defined menu bar without activating it
Syntax	SHOW MENU <menu name1> [, <menu name2> ...] ALL [PAD <pad name>] [SAVE]
See Also	ACTIVATE MENU, CREATE MENU, DEFINE MENU

Description SHOW MENU displays a menu bar or a set of menu bars without activating them. The menu bars are displayed but can't be used. Before they can be displayed, the menu bars must first be created with DEFINE MENU.

Clauses <menu name1> [,<menu name2> ...]
Specify the name of one or more menu bars to display with <menu name1>, <menu name2>, and so on.

ALL
Include ALL to display all currently defined menu bars.

PAD <pad name>
Specify the name of a pad to be highlighted on a menu bar with <pad name>.

SAVE
Include SAVE to retain an image of the specified menu bars without activating the menu bars. The menu bar images can be cleared with CLEAR.

Example

```
CLEAR

DEFINE MENU example BAR AT LINE 2

DEFINE PAD convpad OF example PROMPT '\<Conversions' COLOR SCHEME 3 ;
KEY ALT+C, ''
DEFINE PAD cardpad OF example PROMPT 'Card \<Info' COLOR SCHEME 3 ;
KEY ALT+I, ''

SHOW MENU example
```

SHOW OBJECT

Purpose	Redisplays the @ ... GET control that has the specified object number
Syntax	SHOW OBJECT <expN1> [ENABLE DISABLE] [LEVEL <expN2>] [PROMPT <expC>] [COLOR SCHEME <expN3> COLOR <color pair list>]
Remarks	RGB color pairs — FoxPro for Windows only
See Also	@ ... GET, CLEAR READ, _CUROBJ, OBJNUM(), RDLEVEL(), READ, SHOW GET, SHOW GETS, SYS(2016)

Description SHOW OBJECT redisplay a single control (a check box, a field, an invisible, push or radio button, a list, a popup, a spinner or a text editing region). When the value in an @ ... GET field changes (the cursor moves to a new record, for example), SHOW OBJECT updates the value displayed in the @ ... GET field. When a control is redisplayed, it can be enabled or disabled. SHOW OBJECT can also be used to redisplay an individual button within a set of invisible, radio or push buttons change the prompts of check boxes and buttons.

SHOW OBJECT is typically used in a routine executed in a control-level VALID or WHEN clause or a READ-level ACTIVATE or DEACTIVATE clause.

SHOW GET vs. SHOW GETS and SHOW OBJECT

Individual controls can be redisplayed with SHOW OBJECT or SHOW GET. All controls can be redisplayed with SHOW GETS. SHOW GETS executes a READ-level SHOW routine. SHOW GET and SHOW OBJECT don't.

SHOW OBJECT is similar to SHOW GET except SHOW OBJECT references controls by their object number, and SHOW GET references controls by a memory variable, array element or field.

See OBJNUM() or _CUROBJ for more information on control numbering.

Clauses **<expN1>**

Every control has an associated control number (<expN1>), which is determined by the order in which that control was created. Each button within a set of push, radio and invisible buttons is considered a separate control. A field, check box, popup, list, spinner or text editing region is also a single control.

ENABLE | DISABLE

Including ENABLE allows the specified control to be selected or chosen. Including DISABLE prevents the control from being selected or chosen.

LEVEL <expN2>

You can nest READs by issuing @ ... GETs and a READ in a routine called in an active READ. READs can be nested to five levels.

SHOW OBJECT defaults to the current READ level if LEVEL is omitted. Including LEVEL lets you redisplay a control on a level other than the current level. The numeric expression <expN2> can assume the value 1, 2, 3, 4 or 5, corresponding to the READ level of the control to redisplay. RDLEVEL() returns the current READ level.

PROMPT <expC>

You can replace the prompt for an individual push button, radio button or check box with another prompt by including PROMPT <expC>. <expC> replaces the original button or check box prompt. You can also change the button or check box attributes (which determine whether the control is enabled or disabled, which hot keys are assigned, and whether it is the default or Escape choice) by specifying the appropriate special characters with <expC>.

COLOR SCHEME <expN3>**| COLOR <color pair list>**

Controls can be redisplayed in the colors you specify. You can specify a color scheme or a color pair list. Refer to @ ... GET and the Controls section in this chapter for information about control colors.

Example The following example creates a check box. When the check box is chosen, the VALID routine Newprompt is executed. The routine changes the check box prompt and the hot key.

```
CLEAR
STORE 1 TO check
@ 4,2 GET check FUNCTION '*C \<Checked' SIZE 1,13 VALID NEWPROMPT( )
READ CYCLE
```

```
FUNCTION newprompt
IF check = 0
    SHOW OBJECT 1 PROMPT '\<Unchecked'
ELSE
    SHOW OBJECT 1 PROMPT '\<Checked'
ENDIF
RETURN .T.
```

SHOW POPUP

Purpose	Displays a popup defined with DEFINE POPUP without activating it
Syntax	SHOW POPUP <popup name1> [, <popup name2> ...] ALL [SAVE]
See Also	ACTIVATE POPUP, DEFINE POPUP

Description SHOW POPUP displays a popup or a set of popups without activating them. The popups are displayed but can't be used. Before they can be shown, the popups must first be created with DEFINE POPUP.

Clauses <popup name1> [,<popup name2> ...]
Specify the name of one or more popups to display with <popup name1>, <popup name2>, and so on.

ALL
Include ALL to display all currently defined popups.

SAVE
Include SAVE to retain an image of the specified popups without activating them. The popup images can be cleared with CLEAR.

Example SHOW POPUP is used in this example to display an image of a popup when you choose a special menu pad.

The current system menu bar is saved to memory with SET SYSMENU SAVE, and all system menu pads are removed with SET SYSMENU TO. Three new system menu pads are created with DEFINE PAD, and DEFINE POPUP is used to create a popup for the **Show Card Info** menu pad.

When you choose the **Show Card Info** menu pad, SHOW POPUP displays an image of the **Card Info** popup. You can view the popup, but you can't choose options from it. To clear this image, choose the **Clear Card Info** menu pad. Choose the **Exit** option from the **Card Info** popup to restore the system menu bar.

```
*** Name this program SHOW.PRG ***  
  
CLEAR  
  
SET SYSMENU SAVE  
SET SYSMENU TO
```

SHOW POPUP

```
DEFINE PAD cardpad OF _MSYSMENU PROMPT '\<Card Info' COLOR SCHEME 3 ;
    KEY ALT+I, ''
DEFINE PAD showpad OF _MSYSMENU PROMPT '\<Show Card Info' ;
    COLOR SCHEME 3 KEY ALT+I, ''
DEFINE PAD clearpad OF _MSYSMENU PROMPT 'C\<lear Card Info' ;
    COLOR SCHEME 3 KEY ALT+I, ''

ON PAD cardpad OF _MSYSMENU ACTIVATE POPUP cardinfo
ON SELECTION PAD showpad OF _MSYSMENU ;
    DO choice IN show WITH PROMPT( ), POPUP( )
ON SELECTION PAD clearpad OF _MSYSMENU ;
    DO choice IN show WITH PROMPT( ), POPUP( )

DEFINE POPUP cardinfo MARGIN RELATIVE SHADOW COLOR SCHEME 4
DEFINE BAR 1 OF cardinfo PROMPT '\<View Charges' ;
    KEY ALT+V, ''
DEFINE BAR 2 OF cardinfo PROMPT 'View \<Payments' ;
    KEY ALT+P, ''
DEFINE BAR 3 OF cardinfo PROMPT 'Vie\<w Users' ;
    KEY ALT+W, ''
DEFINE BAR 4 OF cardinfo PROMPT '\-'
DEFINE BAR 5 OF cardinfo PROMPT '\<Charges '
DEFINE BAR 6 OF cardinfo PROMPT '\-'
DEFINE BAR 7 OF cardinfo PROMPT 'E\<xit '

ON SELECTION POPUP cardinfo;
    DO choice IN show WITH PROMPT( ), POPUP( )

PROCEDURE choice
PARAMETERS mprompt, mpopup

IF mprompt = 'Show Card Info'
    SHOW POPUP cardinfo
    RETURN
ENDIF

IF mprompt = 'Clear Card Info'
    CLEAR
    RETURN
ENDIF

WAIT WINDOW 'You chose ' + mprompt + ;
    ' from popup ' + mpopup NOWAIT

IF mprompt = 'Exit'
    SET SYSMENU TO DEFAULT
ENDIF
```

SHOW WINDOW

Purpose	Displays one or more user-defined windows or FoxPro system windows without activating them
Syntax	SHOW WINDOW <window name1> [, <window name2> ...] ALL [IN [WINDOW] <window name3> IN SCREEN] [REFRESH] [TOP BOTTOM SAME] [SAVE]
See Also	ACTIVATE WINDOW, DEFINE WINDOW

Description SHOW WINDOW controls the display and front-to-back screen placement of windows. If windows are defined as hidden or haven't been activated, SHOW WINDOW displays the windows without activating them.

You can also display system windows, such as the Command window, the Calculator, the Calendar/Diary, and so on. Desk accessories (the Filer, the Calculator, and so on) must be active before they can be displayed. If one or more windows are currently displayed, SHOW WINDOW lets you change the front-to-back order of the windows.

You can't specify where output is directed with SHOW WINDOW. Use ACTIVATE WINDOW to direct output to a window.

Clauses <window name1> [, <window name2> ...]
Specify the name of one or more windows to display with <window name1>, <window name2>, and so on.

ALL

Include ALL to display all windows.

IN [WINDOW] <window name3>

If IN WINDOW <window name3> is included, the window is displayed inside a parent window without assuming the characteristics of the parent window. A window activated inside a parent window can't be moved outside the parent window. If the parent window is moved, the child window moves with it.

The parent window specified with <window name3> must first be created with DEFINE WINDOW.

IN SCREEN

By including **IN SCREEN**, you can explicitly place a window on the desktop in FoxPro for MS-DOS or the main FoxPro window in FoxPro for Windows instead of in another window. Windows are placed on the desktop or main FoxPro window by default.

REFRESH

You can include the **REFRESH** clause to redraw a Browse window. This is useful on a network to ensure that you are browsing the most current version of a table/.DBF. The work area for the Browse window table/.DBF is selected.

Memo editing windows are refreshed with changes made to the memo field by other users on a network. **SET REFRESH** determines the interval between memo editing window refreshes. Refer to **SET REFRESH** in this manual for additional information on how data is refreshed in tables/.DBFs opened for shared use on a network.

TOP

Include **TOP** to place the specified window in front of all other windows.

BOTTOM

Include **BOTTOM** to place the specified window behind all other windows.

SAME

SAME affects only windows that have been previously displayed or activated and then cleared from the desktop or main FoxPro window with **DEACTIVATE WINDOW**. Issuing **SHOW WINDOW SAME** places the specified window back into a stack of windows in the same position the window occupied before it was deactivated.

SAVE — FoxPro for MS-DOS only

Including **SAVE** keeps an image of a window on the desktop or in another window after the window has been hidden. Normally, windows are removed from the desktop after they are hidden. The window image can be cleared from the desktop or a window with **CLEAR**. You can't save an image of the current output window.

SAVE is ignored in FoxPro for Windows.

Example In this example, a window named Output is created and displayed. Since SHOW WINDOW is used to display the window, output can't be directed to the window until it is activated.

```
CLEAR  
DEFINE WINDOW output FROM 2,1 TO 13,75 TITLE 'Output' ;  
    CLOSE FLOAT GROW SHADOW ZOOM  
SHOW WINDOW output
```

SIGN()

Purpose	Returns a numeric value of 1, -1 or 0 depending on the sign of a specified numeric expression
Syntax	SIGN(<expN>)
Returns	Numeric
See Also	ABS()

Description SIGN() returns a numeric value of 1, -1 or 0 depending on whether the specified numeric expression evaluates to a positive, negative, or 0 value.

Parameter <expN>
SIGN() returns 1 if <expN> evaluates to a positive number, -1 if <expN> is evaluates to a negative number, and 0 if <expN> evaluates to 0.

Examples

```
STORE 10 to num1
STORE -10 TO num2
STORE 0 TO zero
? SIGN(num1)
1

? SIGN(num2)
-1

? SIGN(zero)
0
```


SIN()

Purpose	Returns the sine of an angle
Syntax	SIN(<expN>)
Returns	Numeric
See Also	ACOS(), COS(), DTOR(), RTOD(), SET DECIMALS

Description This trigonometric function returns the sine of a numeric expression.

Parameter **<expN>**

SIN() returns the sine of the angle specified with <expN>. The numeric expression <expN> can assume any value and the value returned by SIN() ranges between -1 and 1.

The numeric expression <expN> is given in radians. Use DTOR() to convert an angle from degrees to radians. The number of decimal places displayed by SIN() can be specified with SET DECIMALS.

Examples

```
? SIN(0)
0.00

? SIN(PI( )/2)
1.00

? SIN(DTOR(90))
1.00
```

SIZE POPUP

Purpose	Changes the size of a popup defined with DEFINE POPUP
Syntax	SIZE POPUP <popup name> TO <expN1>, <expN2> BY <expN3>, <expN4>
See Also	ACTIVATE POPUP

Description SIZE POPUP changes the size of a user-defined popup. The popup can be changed to a specific size relative to its current size. If a user-defined popup has been created, its size can be changed; it doesn't have to be active or visible.

Clauses **<popup name>**
Specify the name of the popup whose size you want to change with <popup name>.

TO <expN1>, <expN2>
Use the TO clause to change the size of a popup to a specific size. <expN1> and <expN2> specify the new row and column coordinates, respectively, of the lower-right corner of the popup.

BY <expN3>, <expN4>
To change the size of a popup relative to its current size, use the BY clause. The numeric expressions <expN3> and <expN4> specify the change in size of the popup in rows and columns, relative to the current row and column coordinates of the lower-right corner of the popup.

Example This example creates a popup containing files with a .PRG extension and moves, enlarges and shrinks it before closing it.

```
CLEAR
DEFINE POPUP movinpop FROM 2,2 TO 7, 14 PROMPT FILES LIKE *.PRG ;
    TITLE 'Programs'
ACTIVATE POPUP movinpop NOWAIT
=CHRS(2)
MOVE POPUP movinpop BY 5,5 && Move popup down
=CHRS(2)
SIZE POPUP movinpop BY 5,5 && Enlarge the popup
=CHRS(2)
SIZE POPUP movinpop BY -5,-5 && Shrink the popup
=CHRS(2)
MOVE POPUP movinpop BY -5,-5 && Move popup up
=CHRS(2)
DEACTIVATE POPUP movinpop
RELEASE POPUP movinpop
```

SKIP

Purpose	Moves the record pointer forward or backward in the current or specified table/.DBF
Syntax	SKIP [<expN1>] [IN <expN2> <expC>]
See Also	GO, SET SKIP

Description SKIP moves the record pointer in the current or specified table/.DBF.

Clauses <expN1>

<expN1> specifies the number of records to move the record pointer. Issuing SKIP without <expN1> advances the record pointer to the next record. The sequence of the records is the same as the key sequence in the controlling index or index tag if an index is enabled.

The record pointer moves toward the end of the file <expN1> records if <expN1> evaluates to a positive number. The record pointer moves towards the beginning of the file <expN1> records if <expN1> evaluates to a negative number.

If the record pointer is positioned on the last record of a table/.DBF and SKIP with no arguments is executed, RECNO() returns a value one greater than the number of records in the table/.DBF and EOF() returns true (.T.). If the record pointer is positioned on the first record of a table/.DBF and SKIP -1 is executed, RECNO() returns 1 and BOF() returns true (.T.).

IN <expN2> | <expC>

Include a work area number <expN2> or a table/.DBF or work area alias <expC> to move the record pointer in a work area other than the current work area.

Example

```
CLOSE DATABASES
SELECT 2
  USE invoices
SELECT 1
  USE customer
SKIP 4 IN 2
? RECNO(2)
5

GO BOTTOM
SKIP -5
? RECNO( )
496
```

SKPBAR()

Purpose	Determines whether a popup bar is enabled or disabled with SET SKIP OF
Syntax	SKPBAR(<expC>, <expN>)
See Also	DEFINE BAR, SET SKIP OF, SKPPAD()

Description SKPBAR() returns a logical value that indicates whether a popup bar is enabled or disabled. Use SET SKIP OF to enable or disable a popup bar.

SKPPAD() returns true (.T.) if the popup bar is disabled, and false (.F.) if the popup bar is enabled.

Parameters **<expC>**
Specify the name of the popup that contains the bar with <expC>.

<expN>
SKPBAR() returns the enabled or disabled status of the popup bar whose number you specify with <expN>.

SKPPAD()

Purpose	Determines whether a menu pad is enabled or disabled with SET SKIP OF
----------------	---

Syntax	SKPPAD(<expC1>, <expC2>)
---------------	---

See Also	DEFINE BAR, SET SKIP OF, SKPBAR()
-----------------	------------------------------------

Description SKPPAD() returns a logical value that indicates whether a menu pad on a menu bar is enabled or disabled. Use SET SKIP OF to enable or disable a menu pad.

SKPPAD() returns true (.T.) if the menu pad is disabled, and false (.F.) if the menu pad is enabled.

Parameters <expC1>

Specify the name of the menu bar that contains the menu pad with <expC1>.

<expC2>

SKPPAD() returns the enabled or disabled status of the menu pad whose name you specify with <expC2>.

SORT

Purpose	Sorts records in the current table/.DBF and outputs the sorted data to a new table/.DBF
Syntax	SORT TO <file> ON <field1> [/A] [/D] [/C] [, <field2> [/A] [/D] [/C] ...] [ASCENDING DESCENDING] [<scope>] [FOR <expL1>] [WHILE <expL2>] [FIELDS <field list>] [NOOPTIMIZE]
See Also	COPY, DISKSPACE(), INDEX, SYS(2020)

Description SORT sorts the table/.DBF in the current work area and creates a new table/.DBF.

One or more specified fields in the current table/.DBF determine the order in which the records appear in the new table/.DBF.



Be sure you have enough disk space for the new table/.DBF and the temporary work files created during the sort. The disk space needed to perform a sort can be as much as three times the size of the source file. The amount of available disk space can be determined with DISKSPACE() and SYS(2020). If you run out of disk space during a sort, the alert “Not enough disk space” is displayed, and the temporary work files are deleted.

Character type fields that contain numbers and spaces may not sort in the order you expect. Numeric fields fill from right to left, with empty spaces to the left. In contrast, character fields fill from left to right, with empty spaces to the right.

For example, if two records in a table/.DBF contain a character field with 1724 in one record and 18 in the other, and the table/.DBF is sorted on this field in ascending order, the record containing 1724 appears before the record containing 18. This is because FoxPro reads each character in the character fields from left to right, and because 17 (in 1724) is less than 18 (in 18), it puts 1724 first. To avoid this problem, always precede lower numbers with leading zeros (0018) or make the field numeric.

Clauses**<file>**

When a table is sorted, a new table is created. <file> specifies the name of the new table file. FoxPro assumes a .DBF file name extension for tables. A .DBF extension is automatically assigned if the file name you include doesn't have an extension.

ON <field1>

You must include a name of a field (<field1>) from the current table. The contents and data type of the field determine the order of the records in new table. By default, the sort is done in ascending order. You can't sort on memo or general fields.

You can include additional field names (<field2>, <field3>) to further order the new table. The first field <field1> is the primary sort field, the second field <field2> is the secondary sort field, and so on.

Example

Here is an example that sorts a table on three fields. CUSTOMER.DBF is opened and sorted to a new table named TEMP. The records in TEMP are ordered by the CNO field.

```
CLOSE DATABASES
CLEAR
USE customer
LIST FIELDS company, cno NEXT 3
SORT TO temp ON cno
USE temp
LIST FIELDS company, cno NEXT 3
WAIT WINDOW 'Now sorted on CNO' NOWAIT
```

[/A] [/D] [/C]

For each field you include in the sort you can specify an ascending or descending sort order. The /A option specifies an ascending order for the field, and the /D option specifies a descending order. /A or /D can be included with any type of field.

By default, the field sort order for character fields is case sensitive (lower- or upper-case dependent). If you include the /C option after the name of a character field, case is ignored. You can combine the /C option with the /A or /D option. Use one slash (/AC or /DC) if you include the /C option with /A or /D.

In the following example, a new table named CLIENTS is created. INVOICES.DBF is sorted on the IDATE field in ascending order and the ITOTAL field in descending order.

```
USE invoices
SORT TO clients ON idate/A,itotal/D
```

ASCENDING | DESCENDING

You can specify a sort order for all sort fields not followed by the /A or /D options. All sort fields are sorted in ascending order if you include ASCENDING. All sort fields are sorted in descending order if DESCENDING is included. The sort fields default to an ascending order if ASCENDING or DESCENDING isn't included.

<scope>

The scope clauses are: ALL, NEXT <expN>, RECORD <expN>, and REST. These are explained in the Overview of the FoxPro Language chapter of this manual. Commands which include <scope> operate only on the table/.DBF in the active work area.

You can specify a scope of records to sort. Only the records that fall within the range of records specified by the scope are sorted.

The default scope for SORT is ALL records.

FOR <expL1>

If the FOR clause is included, only the records in the current table/.DBF for which the logical condition <expL1> evaluates to true (.T.) are included in the sort. Including FOR lets you conditionally sort records, filtering out undesired records.

Rushmore optimizes a SORT ... FOR command if <expL1> is an optimizable expression. For best performance, use an optimizable expression in the FOR clause. A discussion of expressions Rushmore can optimize appears in the Optimizing Your Application chapter in the FoxPro *Developer's Guide*.

WHILE <expL2>

If the WHILE clause is included, records from the current table/.DBF are included in the sort for as long as the logical expression <expL2> evaluates to true (.T.).

FIELDS <fields list>

The new table that SORT creates can contain a subset of fields from the original table. All fields from the original table are included in the new table if the FIELDS clause isn't included. Include FIELDS and a <fields list> to include specific fields from the original table.

NOOPTIMIZE

To cause Rushmore to not optimize SORT, include NOOPTIMIZE. For further information on NOOPTIMIZE, see SET OPTIMIZE in this manual, and the discussion of Rushmore in the Optimizing Your Application chapter in the *FoxPro Developer's Guide*.

SOUNDEX()

Purpose	Returns a phonetic representation of the specified character expression
Syntax	SOUNDEX(<expC>)
Returns	Character
See Also	DIFFERENCE()

Description SOUNDEX() returns a phonetic representation of a character expression. A four-character string is returned. By comparing the results SOUNDEX() returns for two character expressions, you can determine if the two expressions are phonetically similar, indicating that they sound alike. This can be useful when searching for duplicate records in a table/.DBF.

SOUNDEX() isn't case sensitive and generally disregards vowels.

Parameter **<expC>**
Specify the character expression SOUNDEX() evaluates with <expC>.

Example ? SOUNDEX('Smith')=SOUNDEX('Smyth')
 .T.

 ? SOUNDEX('Computer')
 C513

 USE customer
 INDEX ON SOUNDEX(company) TO newcus
 SET INDEX TO newcus
 SEEK SOUNDEX('Furst')
 ? company
 First Computer

SPACE()

Purpose	Returns a character string composed of a specified number of spaces
Syntax	SPACE(<expN>)
Returns	Character
See Also	PADC(), PADL(), PADR(), REPLICATE()

Description SPACE() returns a character string made up of spaces (CHR(32)).

Parameter <expN>
Specify the number of spaces returned by SPACE() with <expN>. The maximum value for <expN> is 65,504 in the Standard version of FoxPro for MS-DOS. The maximum value of <expN> is limited only by memory in the 32-bit Extended version of FoxPro for MS-DOS and in FoxPro for Windows.

Example
STORE SPACE(15) TO blank
? LEN(blank)
15

SQRT()

Purpose	Returns the square root of the specified numeric expression
Syntax	SQRT(<expN>)
Returns	Numeric
See Also	SET DECIMAL

Description SQRT() returns the square root of a numeric expression. The number of decimal places in the value returned by SQRT() is the larger of the current decimal place setting and the number of decimal places contained in <expN>. The current decimal place setting is specified with SET DECIMALS.

Parameter <expN>
Specify the numeric expression to be evaluated by SQRT() with <expN> . <expN> can't be a negative number.

Examples ? SQRT(4)
2.00

? SQRT(57.6*14.3)
28.70

SROWS()

Purpose	Returns the number of rows available on the desktop or main FoxPro window
Syntax	SROWS()
Returns	Numeric
See Also	COL(), ROW(), SCOLS(), SET DISPLAY, WCOLS(), WROWS()
Description	This function returns the number of rows available on the desktop in FoxPro for MS-DOS or the main FoxPro window in FoxPro for Windows. The value returned by SROWS() depends on the current display mode. The display mode can be changed in FoxPro for MS-DOS with SET DISPLAY. In FoxPro for Windows you can change the display mode with Windows Setup.
Examples	? SROWS()

STORE

Purpose	Stores data to a memory variable stores data to an array
Syntax	STORE <expr> TO <memvar list> <array list> <memvar> <array> = <expr>
See Also	DIMENSION, SET COMPATIBLE

Description STORE sets the value of a memory variable or an existing array. An alternative to STORE is the equal sign (=). The memory variable or array must be on the left side of the equal sign and the value on the right side.

Dates can be stored directly to memory variables or arrays by using braces:

```
STORE {12/25/92} TO XMAS
```

Clauses **<expr>**
The value of the expression <expr> is stored to the memory variable or the array. If the memory variable doesn't exist, it is created and initialized to <expr>. An array must be previously defined with DIMENSION. STORE replaces the value in an existing memory variable or array with the new value.

<memvar list>
Include in <memvar list> a memory variable name or array element, or a list of memory variables names or array elements. Separate memory variable names or array elements with a comma.

<array list>
Include the name of an existing array name or a list of names of existing arrays in <array list>. Separate array names with commas.

STORE initializes every element in the arrays to the specified value if SET COMPATIBLE is OFF. STORE stores the specified value to memory variables of the specified names if SET COMPATIBLE is ON, overwriting any existing arrays that have those names.

Example

```
STORE DATE( ) TO mdate  
STORE 50 TO mnum  
STORE 'Hello' TO mchar  
STORE .T. TO mlog  
DISPLAY MEMORY LIKE m*
```

```
DIMENSION x(2,2)  
STORE 2 TO x  
DISPLAY MEMORY LIKE x
```

STR()

Purpose	Returns the character string equivalent to a specified numeric expression
Syntax	STR(<expN1> [, <expN2> [, <expN3>]])
Parameters	<expN1> The expression to evaluate <expN2> The length of the character string to return <expN3> The number of decimal places in the returned string
Returns	Character
See Also	VAL()

Description STR() returns a character string from a numeric expression.

Parameters **<expN1>**
The numeric expression evaluated by STR() is specified by <expN1>.

<expN2>
The length of the character string returned by STR() is specified with <expN2>. The length includes one character for the decimal point and on character for each digit to the right of the decimal point.

STR() pads the character string with leading spaces if you specify a length larger than the number of digits left of the decimal point. STR() returns a string of asterisks, indicating numeric overflow, if you specify a length less than the number of digits left of the decimal point.

<expN3>
The number of decimal places in the character string returned by STR() is specified by the numeric expression <expN3>. You must include <expN2> to specify the number of decimal places.

If you specify fewer decimal places than are in the numeric expression <expN1>, the extra digits are truncated.

? STR(123.456,10,4)
123.4560

STRTRAN()

Purpose	Searches a character expression or memo field for occurrences of a second character expression or memo field, and then replaces each occurrence with a third character expression or memo field
Syntax	STRTRAN(<expC1>, <expC2> [, <expC3>] [, <expN1>] [, <expN2>])
Parameters	<expC1> <expC2> <expC3> <expN1> <expN2> The character expression to search The character expression in <expC1> to replace The character expression to replace <expC2> The occurrence of <expC2> where replacement begins The number of occurrences to replace
Returns	Character
See Also	STUFF()

Description This function searches a character expression for occurrences of another character expression and replaces the occurrences with a third character expression. The resulting character string is returned.

You can also specify where the replacement begins and how many replacements are made.

Parameters **<expC1>**
The character expression that's searched is specified with <expC1>. <expC1> can be a memo field.

<expC2>
The character expression that's searched for in <expC1> is specified with <expC2>. The search is case-sensitive (lower-case and upper-case are respected). <expC2> can be a memo field.

<expC3>
The search character expression <expC2> is replaced in <expC1> by the character expression <expC3>. If <expC3> is omitted, <expC2> is replaced with the null string.

<expN1>

The optional numeric expression <expN1> specifies which occurrence of <expC2> is the first to be replaced. For example, if <expN1> is 4, replacement begins with the fourth occurrence of <expC2> in <expC1>, counting from the left, and the first three occurrences of <expC2> remain unchanged. The occurrence where replacement begins defaults to 1 if <expN1> is omitted.

<expN2>

<expN2> specifies the number of occurrences of <expC2> to replace. If <expN2> is omitted, all occurrences of <expC2>, starting with the occurrence specified in <expN1>, are replaced.

Example

```
STORE 'abracadabra' TO mystring
```

```
?STRTRAN(mystring, 'a', 'z')
```

```
zbrzczdzbrz
```

```
?STRTRAN(mystring, 'a', 'q', 1, 2)
```

```
qbrqcadabra
```

```
CLOSE DATABASES
```

```
USE parts
```

```
REPLACE descript WITH STRTRAN(descript, 'Abashed', 'Abrasive')
```

```
WAIT WINDOW 'Now you see Abrasive Goat in first descript' NOWAIT
```

```
BROWSE TIMEOUT 5
```

```
WAIT WINDOW 'Now you see Abashed Goat in first descript' NOWAIT
```

```
REPLACE descript WITH STRTRAN(descript, 'Abrasive', 'Abashed')
```

```
BROWSE TIMEOUT 5
```

```
WAIT CLEAR
```

STUFF()

Purpose	Returns a character string created by replacing a specified number of characters in a character expression
Syntax	STUFF(<expC1>, <expN1>, <expN2>, <expC2>)
Parameters	<expC1> The character expression in which replacement occurs <expN1> The position in <expC1> where replacement begins <expN2> The number of characters to replace <expC2> The character expression to place in <expC1>
Returns	Character
See Also	LEFT(), PADC(), PADL(), PADR(), RIGHT(), STRTRAN(), SUBSTR()

Description STUFF() returns a character string created by replacing a specified number of characters in a character expression with another character expression.

Parameters **<expC1>**

Specify the character expression <expC1> in which the replacement occurs.

<expN1>

<expN1> specifies the position in <expC1> where the replacement begins.

<expN2>

The number of characters to be replaced is specified by <expN2>. If <expN2> is 0, the replacement string <expC2> is inserted into <expC1>.

<expC2>

The replacement character expression is specified by <expC2>. If <expC2> is the null string, the number of characters specified by <expN2> are removed from <expC1>.

Examples

```
STORE 'abcdefghijklm' TO str1  
STORE '12345' TO str2
```

```
? STUFF(str1, 4, 0, str2)      && insert  
abc12345defghijklm
```

```
? STUFF(str1, 4, 3, str2)      && replace  
abc12345ghijklm
```

```
? STUFF(str1, 4, 6, '')        && delete  
abcjklm
```

```
? STUFF(str1, 4, 1, str2)      && replace and insert  
abc12345efghijklm
```

```
? STUFF(str1, 4, 4, str2)      && replace and delete  
abc12345hijklm
```

```
? STUFF(str1, 4, LEN(str1), str2) && replace and delete rest  
abc12345
```

SUBSTR()

Purpose	Returns a specified number of characters from the given character expression or memo field
Syntax	SUBSTR(<expC>, <expN1> [, <expN2>])
Parameters	<expC> The character expression from which characters are extracted <expN1> The position in <expC> where character extraction begins <expN2> The number of characters to extract
Returns	Character
See Also	AT(), LEFT(), RIGHT(), STRTRAN(), STUFF()

Description SUBSTR() returns a specified number of characters from a character expression.

This function extracts and returns characters from a character expression, starting at a specified position in the character expression and continuing for a specified number of characters.

Parameters **<expC>**
The character expression from which characters are extracted is specified by <expC>. The expression <expC> can be a memo field.

<expN1>
The position in the character expression <expC> where character extraction begins is specified by <expN1>. The first character of <expC> is position 1.

<expN2>
<expN2> specifies the number of characters to extract from <expC>. If <expN2> is omitted, characters are extracted until the end of the character expression is reached.

Examples

```
STORE 'abcdefghijklm' TO mystring
? SUBSTR(mystring, 1, 5)
abcde

? SUBSTR(mystring, 6)
fghijklm
```

SUM

Purpose	Totals all or specified numeric field values in the current table/.DBF
Syntax	SUM [<expr list>] [<scope>] [FOR <expL1>] [WHILE <expL2>] [TO <memvar list> TO ARRAY <array>] [NOOPTIMIZE]
See Also	AVERAGE, CALCULATE, COUNT

Description SUM totals numeric fields in the current table/.DBF.

Clauses **<expr list>**
One or more fields or field expressions to total. All numeric fields are totaled if the expression list is omitted.

<scope>
The scope clauses are: ALL, NEXT **<expN>**, RECORD **<expN>**, and REST. These are explained in the Overview of the FoxPro Language chapter of this manual. Commands which include **<scope>** operate only on the table/.DBF in the currently active work area.

You can specify a scope of records used in the total. Only the records that fall within the range of records specified by the scope are included in the total.

The default scope for SUM is ALL records.

FOR <expL1>
If the FOR clause is included, only the records for which the logical condition **<expL1>** evaluates to true (.T.) are included in the total. Including FOR lets you conditionally total records, filtering out undesired records.

Rushmore optimizes a SUM ... FOR command if **<expL1>** is an optimizable expression. For best performance, use an optimizable expression in the FOR clause. A discussion of expressions Rushmore can optimize appears in the Optimizing Your Application chapter in the FoxPro *Developer's Guide*.

WHILE <expl2>

If the WHILE clause is included, records from the current table/.DBF are included in the total for as long as the logical expression <expl2> evaluates to true (.T.).

TO <memvar list>

Use the TO <memvar list> clause to store each total to a memory variable. If you specify a memory variable name in <memvar list> that doesn't exist, FoxPro automatically creates it. Separate the memory variable names in the list with a comma.

TO ARRAY <array>

The TO ARRAY clause causes SUM to store totals to a memory variable array. If the array you specify in SUM doesn't exist, FoxPro automatically creates it. If the array exists and is too small to contain all the totals, the size of the array is automatically increased to accommodate the totals.

NOOPTIMIZE

Include NOOPTIMIZE to cause Rushmore to not optimize SUM. For further information on NOOPTIMIZE, see SET OPTIMIZE in this manual, and the discussion of Rushmore in the *Optimizing Your Application* chapter in the *FoxPro Developer's Guide*.

Example

The following example sums and displays the totals of the ONHAND and ONORDER fields in PARTS.DBF, along with a sum of the two totals.

```
CLOSE DATABASES
USE parts
SUM onhand, onorder, onhand+onorder ;
    TO tonhand, tonorder, tunits
?
? 'Total on hand : ', tonhand
? 'Total on order : ', tonorder
? 'Total # units : ', tunits
```

SUSPEND

Purpose	Suspend program execution
Syntax	SUSPEND
See Also	CANCEL, RESUME

Description Use SUSPEND to pause program execution and return to interactive FoxPro. While a program is paused, you can execute intervening commands, check memory variable values, open the Trace and Debug windows and so on.

All memory variables created while the program is paused are PRIVATE.

Issue RESUME to restart execution of a suspended program. Program execution continues with the line following the line containing SUSPEND.

SYSMETRIC()

Purpose	Returns the size of a Windows-style display element
Syntax	SYSMETRIC(<expN>)
Returns	Numeric
Remarks	FoxPro for Windows only
See Also	FONTMETRIC(), GETFONT(), TXTWIDTH(), WFONT()

Description SYSMETRIC() returns the size of a Windows-style display element. Display elements include the main FoxPro window, menus, windows, window controls and the cursor. Values are returned in pixels unless otherwise noted and can vary for different display drivers and video hardware. For further information on Windows-style display elements, refer to the GetSystemMetrics function in the *Microsoft Windows Programmer's Reference*.

FoxPro for Windows uses the same display element sizes for its system menus and windows and for the user-defined menus and windows you create with DEFINE WINDOW and DEFINE MENU. SYSMETRIC() enables you precisely to determine the size of menus, windows and window controls you create in FoxPro for Windows.

Parameter <expN>

Use <expN> to specify a Windows-style screen element. The following table shows values for <expN> and the corresponding screen element.

<expN>	Display Element	<expN>	Display Element
1	Windows screen width	17	Minimized window icon height
2	Windows screen height	18	Maximum cursor width
3	Width of sizable window frame	19	Maximum cursor height
4	Height of sizable window frame	20	Single line menu bar height
5	Width of scroll arrows on vertical scroll bar	21	Maximized window width
6	Height of scroll arrows on vertical scroll bar	22	Maximized window height
7	Width of scroll arrows on horizontal scroll bar	23	Kanji window height
8	Height of scroll arrows on horizontal scroll bar	24	Minimum sizable window width
9	Height of window title	25	Minimum sizable window height
10	Width of non-sizable window frame	26	Minimum window width
11	Height of non-sizable window frame	27	Minimum window height
12	Width of DOUBLE or PANEL window frame	28	Window controls width
13	Height of DOUBLE or PANEL window frame	29	Window controls height
14	Thumb width on horizontal scroll bar in text editing windows	30	1 if mouse hardware present; 0 otherwise
15	Thumb height on vertical scroll bar in text editing windows	31	1 for Windows debugging version; 0 otherwise
16	Minimized window icon width	32	1 if mouse buttons swapped; 0 otherwise

SYS()

Purpose	Returns FoxPro system information
Syntax	SYS()
Returns	Character
See Also	See individual descriptions below

Description The various FoxPro SYS() functions return character values that contain useful system information. The following table lists the SYS() functions and includes a brief description of the information returned by each function.

SYS() Functions	
Function	Information Returned
SYS(0)	Network machine number
SYS(1)	Julian system date
SYS(2)	Seconds since midnight
SYS(3)	Unique file name
SYS(5)	Default drive
SYS(6)	Current printer device
SYS(7)	Current format file
SYS(9)	FoxPro serial number
SYS(10)	String from day number
SYS(11)	Julian day number
SYS(12)	Available memory in bytes
SYS(13)	Printer status
SYS(14)	Index expression
SYS(15)	Character translation
SYS(16)	Executing program file name
SYS(17)	Processor in use
SYS(18)	Current @ ... GET field or object
SYS(20)	Transform German text

SYS() Functions	
Function	Information Returned
SYS(21)	Controlling index number
SYS(22)	Controlling tag or index name
SYS(23)	FoxPro EMS memory usage
SYS(24)	EMS memory limit
SYS(100)	Prior console setting
SYS(101)	Prior device setting
SYS(102)	Prior printer setting
SYS(103)	Prior talk setting
SYS(1001)	FoxPro memory
SYS(1016)	User object memory use
SYS(2000)	File name wildcard match
SYS(2001)	SET ... command status
SYS(2002)	Turn cursor on or off
SYS(2003)	Current directory
SYS(2004)	FoxPro start directory
SYS(2005)	Current resource file
SYS(2006)	Current graphics card
SYS(2007)	Check sum value
SYS(2008)	Cursor shape
SYS(2009)	Swap cursor shape
SYS(2010)	CONFIG.SYS file settings
SYS(2011)	Current lock status
SYS(2012)	Memo field blocksize
SYS(2013)	System menu name string
SYS(2014)	Minimum path
SYS(2015)	Unique procedure name
SYS(2016)	SHOW GETS WINDOW name
SYS(2017)	Display sign-on screen
SYS(2018)	Error message parameter

SYS() Functions	
Function	Information Returned
SYS(2019)	Configuration file name and location
SYS(2020)	Default disk size
SYS(2021)	Filtered index expression
SYS(2022)	Disk cluster size
SYS(2023)	Temporary file drive

SYS(0)

SYS(0) returns the machine number and machine name when using a network version of FoxPro. A machine number and name must first be assigned by the network software and the network shell must be loaded.

If a machine number or name haven't been assigned or the network shell hasn't been loaded, SYS(0) returns a character string consisting of 15 spaces, a pound sign (#) followed by another space and then 0. Consult your network manual for further information on defining a machine number and name.

SYS(0) returns 1 when using single-user FoxPro for MS-DOS.

SYS(1)

This function returns a Julian day number from the current system date. The value returned by SYS(1) is valid in the U.S.A. FoxPro versions for any system date after September 14, 1752 and before December 31, 9999.

Examples:

```
? SYS(1)
2448905
? SYS(10,VAL(SYS(1)))
10/09/92
```

SYS(2)

This function returns the number of seconds elapsed since midnight.

Examples:

```
? TIME( )
10:19:41 am
? SYS(2)
37182
```

SYS()

SYS(3) This function returns a unique file name which can be used to create temporary files. A different file name is returned each time SYS(3) executes.

Example:

```
USE offices
COPY TO (SYS(3) + '.DBF')
```

SYS(5) This function returns the current SET DEFAULT setting.

Example:

```
SET DEFAULT TO C:
? SYS(5)
C:
```

SYS(6) This function returns the current SET PRINTER TO setting.

Examples:

```
? SYS(6)
PRN:
SET PRINTER TO output.txt
? SYS(6)
OUTPUT.TXT
```

SYS(7 [, <expN>])

This function returns the name of the current format file. A format file is opened with SET FORMAT. The null string is returned if a format file isn't open for the specified work area.

The format file for the current work area is returned if you don't include a work area number <expN>.

SYS(9) SYS(9) returns your FoxPro serial number.

SYS(10, <expN>)

SYS(10) returns a character type date from a Julian day number <expN>. The value returned by SYS(10) is valid in U.S.A. FoxPro versions for any Julian day number after September 14, 1752 and before December 31, 9999 (2,361,222 to 5,373,484).

Examples:

```
? SYS(1)
2448905
? SYS(10, VAL(SYS(1)))
10/09/92
```


SYS(11, <expD> | <expC>)

SYS(11) returns a Julian day number from a date expression <expD> or a character expression in a date format <expC>. The day number is returned as a character string.

Examples:

```
? SYS(11,{10/31/92})
2448927
? SYS(11,'10/31/92')
2448927
```

SYS(12)

In FoxPro for MS-DOS, SYS(12) returns the amount of memory below 640KB which is available to execute an external program. Any available expanded memory isn't included in the value returned by this function, and the value can vary throughout a FoxPro for MS-DOS session.

In FoxPro for Windows, SYS(12) always returns 655,360.

SYS(12) is similar to MEMORY() with two exceptions:

- SYS(12) returns the amount of available memory in bytes. MEMORY() returns available memory in kilobytes.
- SYS(12) returns a character string. MEMORY() returns a numeric value.

SYS(13)

This function returns the status of the printer. OFFLINE is returned if the printer isn't ready. READY is returned if the printer is ready.

If the printer is connected to a COM port, SYS(13) returns READY if the printer returns Clear To Send Data or Data Set Ready to FoxPro.

If the printer is connected to a parallel port, SYS(13) returns OFFLINE if the printer returns Out of paper, I/O error, Time out, Printer Busy or Printer Not Selected to FoxPro.

Example:

```
IF SYS(13) = 'OFFLINE'
  WAIT WINDOW 'Printer is offline'
ENDIF
```

SYS(14, <expN1> [, <expN2 | <expC>])

SYS(14) returns the index expression of an open, single-entry index file or of a tag in a compound index file. An index expression is specified when an index file or tag is created with INDEX. The index expression determines how a table/.DBF is displayed and accessed when an index file or tag is used to order the table/.DBF.

See INDEX in this chapter for additional information on index expressions and creating index files and tags. SYS(14) is similar to the KEY() function.

USE and SET INDEX both support an index file name list that lets you open index files for a table/.DBF. Any combination of single-entry index files, structural compound or independent compound index file names can be included in the index file list.

The numeric expression <expN1> specifies which index expression to return from the open index files or tags. SYS(14) returns index expressions from open index files and tags in the following order:

- Index expressions from single-entry index files (if any are open) are returned first. The order the single-entry index files are included in USE or SET INDEX determines the order the index expressions are returned.
- Index expressions for each tag in the structural compound index (if one is present) are returned next. The index expressions are returned from the tags in the order the tags are created in the structural index.
- Index expressions for each tag in any open independent compound indexes are returned last. The index expressions are returned from the tags in the order the tags are created in the independent compound indexes.

The null string is returned if <expN1> is greater than the total number of open single-entry files and structural compound and independent compound index tags.

Index expressions are returned from index files open in the current work area unless you include a work area number <expN2> or a table/.DBF or work area alias <expC>.

If a table/.DBF doesn't have the alias you specify, the message "Alias not found" is displayed.

Example:

```

CLOSE DATABASES
USE customer
INDEX ON cno TO cus_cno
INDEX ON company TO cus_com
SET INDEX TO cus_cno, cus_com
? SYS(14,1)
CNO
? SYS(14,2,'customer')
COMPANY
USE

```

See also: KEY()

SYS(15, <expC1>, <expC2>)

This function is primarily intended for the convenience of European users who must use letters with diacritical marks. Since there are different versions of most vowels (with different diacritical marks), indexing on fields containing such characters doesn't preserve normal alphabetical order.

This function takes each character in the character expression <expC2> and uses the character's numeric value as a subscript into the table/.DBF <expC1> and replaces the character in <expC2> with the character found at that position in the table/.DBF.

If table/.DBF <expC1> doesn't have a character corresponding to a character from <expC2>, those characters in <expC2> aren't translated.

We have included a sample translation table in a memory variable file called EUROPEAN.MEM. Stored in this memory variable file is a character variable called EUROPEAN. This table translates a character with a diacritical mark to the corresponding character without the diacritical mark.

As an example, the following command can be used to index a table/.DBF on a field containing characters with diacritical marks while preserving normal alphabetical order:

Example:

```
INDEX ON SYS(15,EUROPEAN,field) TO file
```

SYS(16 [, <expN>])

This function returns the file name of the currently executing program. This is useful when recovering from errors. SYS(16) is similar to PROGRAM(). However SYS(16) returns a path with the program name; PROGRAM() returns just the program name. If the program being executed is a procedure or function, SYS(16) also returns the name of the file containing the procedure or function.

When the executing program is part of an application (APP), SYS(16) only returns the name of the program — the path isn't returned with the program name.

<expN> indicates from how many levels back the program name is fetched. This value can range from 1 to the depth programs are nested.

If <expN> is 1 or 0, SYS(16) returns the name of the master program (the program first executed). The name of the currently executing program is returned if <expN> is omitted. The null string is returned if <expN> is greater than the program nesting depth.

Example:

The program nesting can be returned with the following short program example:

```
STORE 1 TO i
DO WHILE LEN(SYS(16,i)) <> 0
    ? SYS(16,i)
    STORE i+1 TO i
ENDDO
```

SYS(17)

This function returns the type of central processing unit you are using: 8086/88, 80286, 80386 or 80486.

Example:

```
? 'Processor in use:',SYS(17)
Processor in use: 80386
```

SYS(18)

SYS(18) returns the name in upper-case of the memory variable, array element or field used to create the current @ ... GET control. Controls are check boxes, fields, invisible, push or radio buttons, lists, popups, spinners or text editing regions. If a Browse, Change or Edit window is active, the name of the current field is returned.

SYS(18) is identical to VARREAD(). Both can be used to pass the name of the current control to a procedure.

SYS(20, <expC>, <expN>)

SYS(20) is used to order fields containing German words and names. The fields are ordered to resemble the ordering in a German phone book — upper- and lower-case are intermixed, eszets sort immediately after double s and characters with umlauts sort immediately after the unadorned character followed by the letter e. The following table lists the sort order of a sample set of characters:

A	C	T
a	c	t
AD	...	U
AE	r	u
ae	S	ua
Ä	s	ue
ä	sa	Ü
af	ss	ü
B	ß	V
b	sz	...

SYS(15) can also be used for indexing and sorting fields that contain diacritical marks.

SYS(20) transforms the character expression <expC>. The numeric expression <expN> specifies the number of characters in <expC> to transform, starting with the first character in <expC>. Any additional characters after the position specified by <expN> are ignored. The length of character string returned by SYS(20) is 2 * <expN> + 10 characters, with a maximum length of 254 characters.

In the following example, the CONTACT field contains the names of customers. This field is indexed using SYS(20) and the first 20 characters in the field. An structural index tag named GCONTACT is created.

```
USE CUSTOMER
INDEX ON SYS(20, contact, 20) TAG gcontact
```

SYS(21)

This function returns the index position number of the controlling index for the current table/DBF. The controlling index is the first index specified in USE or SET INDEX. SET ORDER can also be used to specify which index is the controlling index. See USE and SET ORDER in this chapter for more information about the controlling index.

SYS(22 [, <expN>])

This function returns the name of the controlling tag or index for a table/DBF. The controlling tag or index is returned for the current work area unless another work area number <expN> is included.

SYS(23)

SYS(23) returns the amount of EMS memory (in 16K segments) that is currently being used by FoxPro. 0 is returned if no EMS memory is in use.

SYS(23) always returns 0 in the 32-bit Extended version of FoxPro for MS-DOS and FoxPro for Windows.

SYS(24)

This function returns the EMS limit set in your FoxPro configuration file. SYS(24) returns 0 if an EMS limit was not included in your configuration file. SYS(23) always returns 0 in the 32-bit Extended version of FoxPro for MS-DOS and FoxPro for Windows.

See the Customizing FoxPro chapter in the *FoxPro Installation and Configuration* manual for complete information on this configuration setting.

SYS(100)

This function returns the current SET CONSOLE setting.

Example:

```
IF SYS(100) <> 'ON'
    SET CONSOLE ON
ENDIF
```

SYS(101)

This function returns the current SET DEVICE setting.

Example:

```
IF SYS(101) <> 'SCREEN'
    SET DEVICE TO SCREEN
ENDIF
```

SYS(102)

This function returns the current SET PRINTER setting.

Example:

```
IF SYS(102) <> 'OFF'
    SET PRINTER OFF
ENDIF
```

SYS(103) This function returns the current SET TALK setting.

Example:

```
IF SYS(103) <> 'ON'
    SET TALK ON
ENDIF
```

SYS(1001) SYS(1001) returns the total amount of memory available to the FoxPro memory manager including high memory between 640K and 1MB that is available to MS-DOS. If you have 64K of LIM 4.0 compatible expanded memory available, this is also included. SYS(12) and MEMORY() return only memory below 640KB that's available when you execute an external program.

SYS(1016) SYS(1016) returns the amount of memory being used by objects you define — user-defined windows, menu bars, popups, memory variables, arrays, open tables/.DBFs, files opened with low-level file functions and so on.

SYS(2000, <expC> [, 1])

This function returns the first file name that matches the skeleton <expC>. The name of the next matching file is returned if 1 is included. The null string is returned if a matching file can't be found. The file skeleton can contain the wildcard characters ? and *.

Examples:

```
? SYS(2000, 'CUSTOMER.*')
CUSTOMER.DBF
? SYS(2000, 'CUSTOMER.*', 1)
CUSTOMER.FPT
```

SYS(2001, <expC> [, 1])

SYS(2001) returns the current status of the specified SET ... ON/OFF or SET ... TO commands. This function is identical to SET().

Some SET commands have two settings (for example, SET PRINTER ON | OFF and SET PRINTER TO <file>). Use SYS(2001) without 1 to return the ON | OFF switch setting. Use SYS(2001) with 1 to return the an additional setting.

Examples:

```
? SYS(2001, 'PRINTER')
OFF
? SYS(2001, 'PRINTER', 1)
PRN:
```

SYS(2002 [, 1])

This function turns the cursor on or off. SYS(2002) turns the cursor off. SYS(2002, 1) turns the cursor on. See SET CURSOR for additional information about turning the cursor on or off.

SYS(2003) This function returns the name of the current directory on the default drive. You can use SYS(5) to determine the current drive.

Example:

```
? 'Current directory:', SYS(2003)
Current directory: \FOXPRO25
```

SYS(2004) This function returns the name of the directory that FoxPro was started from.

Example:

```
? 'FoxPro launch directory: ', SYS(2004)
FoxPro launch directory: C:\FOXPRO\
```

SYS(2005) This function returns the name of the current resource file. The resource file defaults to FOXUSER.DBF.

Example:

```
? 'Current resource file:', SYS(2005)
Current resource file: C:\FOXPRO25\FOXUSER.DBF
```


SYS(2006) This function returns as a character string the type of graphics card and monitor you are using.

Example:

```
? 'Current graphics card/monitor:', SYS(2006)
Current graphics card/monitor: VGA/Color
```

SYS(2007, <expC>)

This function returns the check sum value of the character expression <expC>. A check sum can be used to test the validity of data or to compare two character expressions.

SYS(2008 [, <expC> [, <expN>]]) — FoxPro for MS-DOS only

You can specify the shape of the insert and overwrite mode cursors in FoxPro for MS-DOS with this function.

SYS(2008) is ignored in FoxPro for Windows.

The default setting for the insert cursor is an underline and for the overwrite cursor is a block.

Use <expC> to specify the cursor to change — include an I to change the insert cursor or O to change the overwrite cursor. You can specify the cursor shape with <expN> — use 0 for underline, 1 for a block and 2 for a half size block.

The insert or overwrite cursor is reset to their default settings if you omit the optional <expN> but include <expC>. Both the insert and overwrite cursors are reset to their default settings if you omit <expN> and <expC>.

Examples:

```
=SYS(2008, 'I', 2)
```

```
=SYS(2008, 'I')
```

```
=SYS(2008)
```

The insert mode cursor is set to a half size block in the first example. The insert cursor is reset to its default shape, an underline in the second example. Both the insert and overwrite cursors are reset to their default values in the third example.

See also: INSMODE(), SYS(2009)

SYS(2009) — FoxPro for MS-DOS only

This function swaps the shape of the insert and overwrite mode cursors. For example, if the insert cursor is a block and the overwrite cursor is an underline, issuing `= SYS(2009)` changes the insert cursor to an underline and the overwrite cursor to a block.

The insert and overwrite cursor shapes can be set with `SYS(2008)`.

`SYS(2008)` is ignored in FoxPro for Windows.

See also: `INSMODE( )`, `SYS(2008)`

SYS(2010) In FoxPro for MS-DOS, `SYS(2010)` returns as a character string the files setting in your `CONFIG.SYS` configuration file.

In FoxPro for Windows, `SYS(2010)` always returns 255.

Most `CONFIG.SYS` files contain a line specifying the maximum number of files that can be opened at the same time under MS-DOS. This line usually reads `FILES=NNN` where `NNN` is a number. `SYS(2010)` returns this number.

A `CONFIG.SYS` configuration file doesn't have to have a files setting and you may not have a `CONFIG.SYS` file. `SYS(2010)` returns the default MS-DOS files setting in either case.

The number returned by `SYS(2010)` isn't the number of files you can open in FoxPro. MS-DOS opens files for its own use. FoxPro also open files for its own internal use and the number of these open files can vary throughout a FoxPro session. The file setting in your `CONFIG.SYS` file must be somewhat greater than the number of files you would like to open in FoxPro.

For more information on the `CONFIG.SYS` configuration file consult your MS-DOS manual.

SYS(2011) Environment: Networks

`SYS(2011)` returns the current record- or file-lock status for the current work area. Unlike the `FLOCK( )`, `LOCK( )` and `RLOCK( )` functions, `SYS(2011)` doesn't attempt to lock the file or record.

The character string returned by `SYS(2011)` is identical to the message displayed in the status bar (Exclusive, Record Unlocked, Record Locked ...).

`SYS(2011)` returns Exclusive only at the workstation that opened the table/.DBF exclusively and Record Locked only at the workstation that applied the record lock.

See also: `FLOCK( )`, `LOCK( )`, `RLOCK( )`

SYS(2012, [<expN> | <expC>])

SYS(2012) returns the memo field blocksize for a table/.DBF. See SET BLOCKSIZE for more information on specifying the memo field blocksize for a table/.DBF.

The memo field blocksize is returned for the current work area. To return the blocksize from a table/.DBF open in another work area, include the optional work area number <expN> or the table/.DBF alias <expC>.

SYS(2012) returns 0 if a table/.DBF isn't open in the specified work area or the table/.DBF doesn't have a memo field.

See also: SET BLOCKSIZE

SYS(2013) SYS(2013) facilitates utilization of the system menu bar and system menu popups. SYS(2013) returns a space delimited character string containing the names of the system menu bar, each pad in the system menu bar, system menu popups and each option in the menu popups.

See also: DEFINE BAR, DEFINE MENU, DEFINE POPUP

SYS(2014, <expC1> [, <expC2>])

SYS(2014) returns the minimum path relative to the current or specified directory for a specified file.

SYS(2014), when used with FULLPATH(), can be used to make applications portable. That is, given the location of the current program and the location of other files used by the program, FULLPATH() and SYS(2014) are used to obtain the proper paths of these files. Once the files' paths are determined, the files can be accessed by the program.

To return the minimum path between a file and the current directory, include the name of the file in the character expression <expC1>. The current directory can be changed with SET DEFAULT.

You can also return the relative minimum path between a file and a specified directory by including the file name in <expC1> and the directory specification in <expC2>.

See FULLPATH() in this help facility for additional information and an example of the use of SYS(2014) and FULLPATH().

See also: FULLPATH(), SET DEFAULT, SET PATH

SYS(2015) SYS(2015) returns a unique 10 character string that begins with an underscore followed by letters and numbers. This character name can be used to create unique procedure or function names.

The string returned is created from the system date and system time (with a millisecond resolution). Calling SYS(2015) more than once during the same millisecond interval still returns a unique character string.

See also: SYS(3)

SYS(2016) SHOW GETS supports an optional WINDOW clause that lets you refresh controls (check boxes, fields, invisible, push or radio buttons, lists, popups, spinners or text editing regions) in a specific window. SYS(2016) returns the window name included in the most recent SHOW GETS WINDOW. SYS(2016) only returns a value in a READ SHOW routine.

SYS(2016) returns an asterisk (*) if you issue SHOW GETS without a WINDOW clause (refreshing @ ... GETs in all windows). Use SHOW GETS WINDOW "" to refresh only controls on the desktop in FoxPro for MS-DOS or the main FoxPro window in FoxPro for Windows. SYS(2016) returns the null string when you refresh controls on the desktop or main FoxPro window.

See also: READ, SHOW GETS

SYS(2017) SYS(2017) clears the desktop in FoxPro for MS-DOS or the main FoxPro window in FoxPro for Windows and displays the FoxPro sign-on screen. Your FoxPro serial number and version number is shown in the sign-on screen. Either of these commands display the sign-on screen:

=SYS(2017)

? SYS(2017)

SYS(2018) This function returns the error message parameter for the most recent error. Certain error messages return additional information about the cause of the error. For example, if you reference a memory variable that doesn't exist, the name of the memory variable is included in the error message. SYS(2018) returns this additional information, called the error message parameter. Some examples of error message parameters are memory variable names and files names.

Suppose you try to run a program named REPORTS, and the program doesn't exist:

```
DO REPORTS
```

The text of the error message displayed is

File REPORTS doesn't exist.

REPORTS is the error message parameter, and it is returned by SYS(2018):

```
? SYS(2018)
REPORTS
```

See Also: ERROR(), MESSAGE(), ON ERROR

SYS(2019) SYS(2019) returns the name and location of your FoxPro configuration file. The name of the FoxPro configuration file is CONFIG.FP in FoxPro for MS-DOS and CONFIG.FPW in FoxPro for Windows. If a configuration file can't be located, the null string is returned.

The FoxPro configuration file is typically located in the directory where FoxPro is started, but can be located elsewhere. FoxPro first looks for the FoxPro configuration file in the directory where FoxPro is started. If the FoxPro configuration file isn't located here, FoxPro then searches the MS-DOS path.

Two options let you tell FoxPro where your configuration file is located and its name (which may differ from CONFIG.FP or CONFIG.FPW). An MS-DOS environmental variable named FOXPROCFG can be created with the MS-DOS SET command. You can store the location and name of your configuration file in FOXPROCFG.

You can also use the -C switch when starting FoxPro to designate a configuration file. Include the location and name of the configuration file immediately after -C.

See the Customizing FoxPro chapter in the *FoxPro Installation and Configuration* manual for further information on the FoxPro configuration file.

SYS(2020) SYS(2020) returns the total size in bytes of the default disk. The default disk can be specified with SET DEFAULT.

SYS(2021, <expN1> [, <expN2> | <expC>])

You can create filtered indexes in FoxPro. If you include the optional FOR clause in INDEX, the index file acts as a filter on the table/.DBF. Only records that match the filter expression <expL> are available for display and access — index keys are created in the index file for just those records matching the filter expression.

SYS(2021) returns the index filter expression of an open single-entry index file or a tag in a compound index file. The null string is returned if the index is created without a FOR clause.

USE and SET INDEX both support an index file name list that lets you open index files for a table/.DBF. Any combination of single-entry index file names, structural compound or independent compound index file names may be included in the index file name list.

The numeric expression <expN1> specifies which filter expression to return from the open index files or tags. SYS(2021) returns filter expressions from open index files and tags in the following order:

- Filter expressions from single-entry index files (if any are open) are returned first. The order the single-entry index files are included in USE or SET INDEX determines the order the filter expressions are returned.
- Filter expressions for each tag in the structural compound index (if one is present) are returned next. The filter expressions are returned from the tags in the order the tags are created in the structural index.
- Filter expressions for each tag in any open independent compound indexes are returned last. The filter expressions are returned from the tags in the order the tags are created in the independent compound indexes.

The null string is returned if <expN1> is greater than the total number of open single-entry files and structural compound and independent compound index tags.

Filter expressions are returned from index files open in the current work area unless you include an optional work area or alias. To return filter expressions from index files open in a specific work area, include the work area number <expN2> or the table/.DBF alias <expC>. If a table/.DBF doesn't have the alias you specify, the message "Alias not found" is displayed.

SYS(2022 [, <drive>])

SYS(2022) returns the cluster size in bytes of the specified disk. To return the cluster size of a disk other than the current default disk, include the disk drive letter (A, B, C, ...) in <drive>.

SYS(2023) This function returns the drive and directory in which the FoxPro temporary files are stored. You can specify the drive where temporary files are stored by including the special TMPFILES configuration item in your FoxPro configuration file. See the Customizing FoxPro chapter in the FoxPro *Installation and Configuration* manual for additional information about the special TMPFILES configuration item.

TAG()

Purpose	Returns a tag name from a .CDX compound index file or the name of an .IDX single-entry index file
Syntax	TAG([<cdx file>], <expN1> [, <expN2> <expC>])
Parameters	<cdx file> The name of the compound index file <expN1> A number indicating the order in which the desired tag was created <expN2> The work area number for a table/.DBF <expC> The table/.DBF alias
Returns	Character
See Also	CDX(), INDEX, MDX(), NDX(), SET INDEX, SYS(14), USE

Description TAG() returns a tag name from an open multiple-entry compound index file or the name of an open single-entry index file. Tags are index entries in compound index files created with INDEX.



Index files can be opened for a table/.DBF with the INDEX clause of USE or with SET INDEX. A structural compound index file is automatically opened with its table/.DBF.

Parameters <cdx file>

<cdx file> is the name of the compound index file from which you want to return a tag name.

<expN1>

The tag names in the compound index file are returned in the order in which the tags were created. For example, if <expN1> is 1, the name of the first tag created in the compound index file is returned. If <expN1> is 2, the name of the second tag created is returned, and so on. The null string is returned when <expN1> exceeds the number of tag names.

If you omit <cdx file>, TAG() returns compound index file tag names and single-entry index file names in a specific order. First, the names of single-entry index files are returned based on their order in the INDEX clause of USE or SET INDEX. Next, tag names from the structural compound index file (if one exists for the table/.DBF) are returned. Finally, tag names from other open compound index files are returned, again in the order in which the tags were created in the compound index files and in the order in which the compound index files are specified in the INDEX clause of USE or SET INDEX.

<expN2> | <expC>

Tag names and index file names are returned from files in the current work area if <expN2> or <expC> isn't included.

You can return tag names and index file names from files open in other work areas by specifying the work area number with <expN2> or the table/.DBF alias with <expC>.

Example

The following example creates an array and stores 'Record #' in the first element. The names of all the index files tags for CUSTOMER.DBF are returned and stored in the array. The size of the array is adjusted to accommodate the names of the indexes. The number and names of the indexes are then displayed.

```
CLOSE DATABASES
USE customer
DIMENSION dbftags(256)
dbftags(1) = 'Record #'
FOR i = 2 TO 256
    IF EMPTY(TAG(i-1))
        i = i - 1
        DIMENSION dbftags(i)
        EXIT
    ELSE
        dbftags(i) = TAG(i-1)
    ENDIF
ENDFOR
DISPLAY MEMORY LIKE dbftags
```

TAN()

Purpose	Returns the tangent of an angle
Syntax	TAN(<expN>)
Returns	Numeric
See Also	COS(), DTOR(), RTOD(), SET DECIMALS, SIN()

Description This trigonometric function returns the tangent of an angle.

Parameter <expN>
TAN() returns the tangent of the angle specified with <expN>, which is given in radians. To convert an angle from degrees to radians, use DTOR(). The number of decimal places returned by TAN() can be specified with SET DECIMALS.

Examples

```
? TAN(0)
0.00

? TAN(PI( )/4)
1.00

? TAN(PI( )*3/4)
-1.00
```

TARGET()

Purpose	Returns the alias of a table/.DBF that is the target for a relation
Syntax	TARGET(<expN1> [, <expN2> <expC>])
Parameters	<expN1> A number <expN2> The work area number for a table/.DBF <expC> The table/.DBF alias
Returns	Character
See Also	RELATION(), SET RELATION

Description This function returns the alias of the table/.DBF that is the target for a relation as specified in the INTO clause of SET RELATION.

Parameter <expN1>
Include a number from 1 to the number of relationships out of the specified work area. The alias of the target databases are returned (in no specific order) until <expN1> is greater than the number of relationships. When <expN1> is greater than number of relationships, the null string is returned.

<expN2> | <expC>
The alias of the target table/.DBF from the current work area is returned if you don't specify the work area or alias. Specify a work area number with <expN2> or table/.DBF alias with <expC> to return the alias of the target table/.DBF in another work area.

Example In this example, a relation is set on the field ONO from SALESMAN.DBF into CUSTOMER.DBF, and another is set on the same field into OFFICES.DBF. The ADDITIVE key word is used with the second relation to add the relation to the existing relation without replacing it. The TARGET() function is then used to return the alias of the target table.

TARGET()

```
CLOSE DATABASES
USE salesman IN 0 ORDER ono
USE customer IN 0 ORDER ono
USE offices IN 0 ORDER ono
SELECT salesman
SET RELATION TO ono INTO customer
SET RELATION TO ono INTO offices ADDITIVE
```

? TARGET(1)

? TARGET(2)

? TARGET(3) && null string

TEXT ... ENDTEXT

Purpose	Outputs lines of text, the results of expressions and functions, and the contents of memory variables
Syntax	TEXT <text lines> ENDTEXT
See Also	FOPEN(), _PRETEXT, SET TEXTMERGE, SET TEXTMERGE DELIMITERS, _TEXT

Description This structured programming command sends text lines placed between TEXT and ENDTEXT to the desktop in FoxPro for MS-DOS or the main FoxPro window in FoxPro for Windows, a user-defined window, a printer, a text file or a low-level file.

TEXT sends the text lines to the current output device. This continues until an ENDTEXT statement is encountered or until the program ends.

By default, output from TEXT ... ENDTEXT is sent to the desktop, the main FoxPro window or the active window. Issue SET CONSOLE OFF to suppress output to the desktop, the main FoxPro window or the active window. Issue SET PRINTER to send output to a printer or a text file.

Output from TEXT ... ENDTEXT can also be sent to a low-level file created or opened with FCREATE() or FOPEN(). If a file handle returned by FCREATE() or FOPEN() is stored to the _TEXT system memory variable, output is directed to the corresponding low-level file.

Clause <text lines>

<text lines> can consist of text, memory variables, array elements, expressions, functions or any combination of these.

Expressions, functions, memory variables and array elements specified with <text lines> are evaluated only if SET TEXTMERGE is ON and must be enclosed in the delimiters specified by SET TEXTMERGE DELIMITERS. If SET TEXTMERGE is OFF, the expressions, functions, memory variables and array elements are output as literals along with their delimiters.

TEXT ... ENDTEXT

For example, today's date is output if SET TEXTMERGE is ON and a text line contains <<DATE(>>. If SET TEXTMERGE is OFF, <<DATE(>> is output literally.

If you place comments within TEXT and ENDTEXT or after \ or \\\, the comments are also output.

Example The following example demonstrates how you can use SET TEXTMERGE, SET TEXTMERGE DELIMITERS, TEXT ... ENDTEXT and the _TEXT system memory variable.

A low-level file named NAMES.TXT is created and its file handle is stored in the _TEXT system memory variable. The program is exited if NAMES.TXT can't be created. SALESMAN.DBF is opened, and the names of the first 10 salemen are output to NAMES.TXT. Text and the results of functions are output to the text file.

The text file containing the names is then opened with MODIFY FILE.

```
CLEAR
CLOSE DATABASES
SET TALK OFF
SET TEXTMERGE ON      && Enable embedding of expressions and functions
STORE FCREATE('names.txt') TO _TEXT  && Create low-level file
IF _TEXT = -1          && Can't create low-level file then exit program
    WAIT WINDOW 'Cannot create an output file, press a key to exit'
    CANCEL
ENDIF
SET HELP OFF
USE salesman
TEXT
    SALES FORCE NAMES

    <<DATE(>>          <<TIME(>>

ENDTEXT

WAIT WINDOW 'Press a key to generate the first ten names'
SCAN NEXT 10
    TEXT
        <<name>>
    ENDTEXT
ENDSCAN

CLOSE ALL      && close the text file the table/.DBF
MODIFY FILE names.txt
ERASE names.txt
```

TIME()

Purpose Returns the current system time

Syntax **TIME**([<expN>])

Returns Character

See Also SECONDS(), SYS(2)

Description TIME() returns the current system time in 24-hour, eight-character string (HH:MM:SS) format.

Parameter <expN>

The time returned includes hundredths of a second if <expN> is included. The numeric expression <expN> can be any value. However, the actual maximum resolution is about 1/18 second. Use SECONDS() for greater resolution.

TOTAL

Purpose	Creates summary records in a new table/.DBF with one record for each group of identically keyed records in the current table/.DBF.
Syntax	TOTAL TO <file> ON <expr> [FIELDS <field list>] [<scope>] [FOR <expL1>] [WHILE <expL2>] [NOOPTIMIZE]
See Also	AVERAGE, CALCULATE, INDEX, SORT, SUM

Description TOTAL computes totals for numeric fields in the current table/.DBF. The table/.DBF must be sorted or indexed. A separate total is calculated for each set of records with a unique index key value. The results are placed into records in a second table/.DBF. One record is created in the second table/.DBF for each unique index key value.

Numeric overflow can occur if the numeric fields in the second table/.DBF aren't wide enough to contain the totals. FoxPro conserves the most significant portions of the totals when numeric overflow occurs. When a field is too small to accept a total:

- Decimal places are truncated, and the remaining decimal portion of the total is rounded.
- If the total still doesn't fit, scientific notation is used if the total field contains seven or more digits.
- Finally, asterisks replace the field contents.

Clauses **<file>**
<file> is the name of the output table. If the specified table doesn't exist, FoxPro creates it. If the output table does exist and SET SAFETY is ON, FoxPro asks if you would like to overwrite the existing table. If SET SAFETY is OFF, you aren't prompted, and the output table is overwritten.

<expr>
<expr> is the expression on which the source table is indexed or sorted.

FIELDS <field list>

By default, all numeric fields are totaled unless **FIELDS <field list>** is included. If you do include this clause, only the fields specified with <field list> are totaled. Separate the field names in the list with commas.

<scope>

The scope clauses are: **ALL**, **NEXT <expN>**, **RECORD <expN>**, and **REST**. These are explained in the Overview of the FoxPro Language chapter of this manual. Commands which include <scope> operate only on the table/.DBF in the active work area.

You can specify a scope of records to total. Only the records that fall within the range of records specified by the scope are totaled.

The default scope for **TOTAL** is **ALL** records.

FOR <expL1>

Including the **FOR** clause lets you conditionally total records, filtering out undesired records. If the **FOR** clause is included, only the records that satisfy the logical condition <expL1> are included in the totals.

Rushmore optimizes a query created with **TOTAL ... FOR** if <expL1> is an optimizable expression. For best performance, use an optimizable expression in the **FOR** clause. A discussion of Rushmore optimizable expressions appears in the Optimizing Your Application chapter in the *FoxPro Developer's Guide*.

WHILE <expL2>

If the **WHILE** clause is included, records from the current table/.DBF are included in the totals for as long as the logical expression <expL2> evaluates to true (.T.).

NOOPTIMIZE

Include **NOOPTIMIZE** to disable Rushmore optimization of **TOTAL**. For further information on **NOOPTIMIZE**, see **SET OPTIMIZE** in this chapter and the discussion of Rushmore technology in the Optimizing Your Application chapter in the *FoxPro Developer's Guide*.

Example This example creates a new table named CUSTOTAL.DBF containing totals from the CUSTOMER.DBF. A Browse window is opened displaying the totals in the CUSTOTAL.DBF.

```
CLOSE DATABASES
USE customer ORDER ono
TOTAL ON ono TO custotal
USE custotal
WAIT WINDOW 'Here are your totals' NOWAIT
BROWSE FIELDS ono, ytdpurch
CLOSE DATABASES
ERASE custotal.dbf
```

TRANSFORM()

Purpose	Formats character or numeric expressions using PICTURE and FUNCTION codes
Syntax	TRANSFORM(<expr>, <expC>)
Parameters	<expr> The expression to be formatted <expC> The PICTURE or FUNCTION code that specifies how <expr> is formatted
Returns	Character
See Also	@ ... SAY

Description TRANSFORM() lets you format a character or numeric expression with the PICTURE and FUNCTION codes available for @ ... SAY.

TRANSFORM() returns a character string in a format determined by a character expression containing the PICTURE or FUNCTION codes.

Parameters <expr>

Specify the numeric or character expression to format with <expr>.

<expC>

<expC> is one or more PICTURE or FUNCTION codes that specify how the expression is formatted. See @ ... SAY and @ ... GET in this chapter for information about the available PICTURE and FUNCTION codes.

Example

```
STORE 15.89 TO Price
? TRANSFORM (Price, '$$$$$.99')
$15.89
```

TRIM()

Purpose	Returns the specified character expression with all trailing blanks removed
Syntax	TRIM(<expC>)
Returns	Character
See Also	ALLTRIM(), LTRIM(), RTRIM()

Description TRIM() returns a character expression with all trailing blanks removed. TRIM() is identical to RTRIM().

Parameter **<expC>**
<expC> is the character expression from which you want to trim all trailing blanks.

Example

```
CLOSE DATABASES
USE customer
GOTO 3
? 'The contact for ' + TRIM(company) + ' is ' + contact
The contact for 1st Software Systems Ltd. is Rance Sivren
```

TXTWIDTH()

Purpose	Returns the length of a character expression with respect to a font in FoxPro for Windows
Syntax	TXTWIDTH(<expC1> [, <expC2>, <expN> [, <expC3>]])
Returns	Numeric
Remarks	FoxPro for Windows only
See Also	FONTMETRIC(), GETFONT(), SYSMETRIC(), WFONT()

Description TXTWIDTH() determines the length of a character expression with specific font attributes with respect to the font for the active output window. The length is returned as the *number of characters* the character expression occupies in the current font for the active output window.

The value returned by TXTWIDTH() is determined by the current font for the main FoxPro window if output is directed to the main FoxPro window when TXTWIDTH() is issued. If output is being directed to a user-defined window, the value returned is determined by the window's current font.

The number of characters <expC> occupies is based on the average character width of the current font for the main FoxPro window or active output window. Use FONTMETRIC(6) to determine the average character width of a font.

Parameters <expC1>

TXTWIDTH() returns the length of the character expression specified with <expC>. If you include only <expC1>, the length of the character expression is returned for the current font for the active output window. In this case, the value returned is the number of characters in <expC1>. A non-proportional font always returns a value equal to the number of characters in <expC1>.

Include the optional arguments <expC2>, <expN> and <expC3> to determine the length of a character expression with a specific font, font size or font style.

<expC2>

Include <expC2> to return the length of a character expression with a specific font. <expC2> is the name of the font.

<expN>

Include <expN> to return the length of a character expression with a specific font size. <expN> is a number that corresponds to the font size in points.

<expC3>

Include <expC3> to specify a special font style for the character expression. <expC3> is a font style code specified by a character or a set of characters. The normal font style is used if <expC3> isn't included. Here are the characters for each font style:

Character	Font Style
B	Bold
I	Italic
N	Normal
O	Outline
Q	Opaque
S	Shadow
—	Strikeout
T	Transparent
U	Underline

You can include more than one character to specify a combination of font styles. For example, BI specifies a bold italic font style.

The following commands are used to print the length (relative to the current font in the main FoxPro window) of a string literal in 32-point bold italic Script font:

```
ACTIVATE SCREEN    && Direct output to the main FoxPro window
? TXTWIDTH('Greetings from Redmond', 'Script', 32, 'BI')
```

TYPE

Purpose	Displays the contents of an ASCII file
Syntax	TYPE <file1> [AUTO] [WRAP] [TO PRINTER [PROMPT] TO FILE <file2>] [NUMBER]
Remarks	PROMPT clause — FoxPro for Windows only
See Also	SET HEADING

Description TYPE displays the contents of text files. This display can be directed to the desktop in FoxPro for MS-DOS, the main FoxPro window in FoxPro for Windows, the active user-defined window, a printer or another file.

When SET HEADING is ON, FoxPro inserts a form feed, the path and name of the file and the date at the beginning of output produced with TYPE. If SET HEADING is OFF, this information isn't included.

In FoxPro for MS-DOS, if a printer driver setup is loaded and you direct output from TYPE to a file or a printer, TYPE displays the file contents using the settings from the printer driver setup.

Clauses <file1>

<file1> is the name of the file to display. The name must include a file extension.

AUTO

Include the AUTO key word to turn automatic indentation on. When you also include WRAP, TYPE automatically indents wrapped text in each paragraph by the same amount it indents the first line of the paragraph. For example:

- Wrapped lines are aligned with the tab when a paragraph begins with a tab.
- Wrapped lines are aligned with the tab when a paragraph begins with a number (or other text) followed by a tab and text.
- Wrapped lines are indented by the same amount as the first line when the first line of a paragraph is indented with spaces.

WRAP

Include **WRAP** to enable word wrap. When word wrap is enabled, a word that is too long to fit at the end of a line is automatically moved to the next line.

TO PRINTER [PROMPT]

Include **TO PRINTER** to direct output to the printer.

In FoxPro for Windows, you can include the optional **PROMPT** clause to display a dialog before printing starts. Place the **PROMPT** key word immediately after **TO PRINTER**. **PROMPT** is ignored in FoxPro for MS-DOS.

In this dialog you can adjust printer settings. You can specify the number of copies and page numbers to print and direct print output to a file. The Windows Printer Setup dialog can be opened to adjust additional printer settings. The printer settings you can adjust depend on the currently installed printer driver.

TO FILE <file2>

Include **TO FILE <file2>** to direct output to the file specified with **<file2>**.

NUMBER

Include **NUMBER** to place line numbers at the beginning of each line in the output.

TYPE()

Purpose	Returns the data type of the specified memo field or character, numeric, logical or date expression
Syntax	TYPE(<expC>)
Returns	Character
See Also	EVALUATE()

Description TYPE() returns a single character that identifies the data type of an expression. The following table lists the characters TYPE() returns and their corresponding data types:

Data Type	Character Returned
Character	C
Numeric (also float)	N
Date	D
Logical	L
Memo	M
Undefined type of expression	U

Parameter <expC>

<expC> is the name of a memo field or a character expression. TYPE() evaluates the contents of <expC> and returns the appropriate data type.

Examples

```

CLOSE DATABASES
USE salesman
? TYPE('salesman.name')
C
? TYPE('(12 * 3) + 4')
N
? TYPE('DATE( )')
D
? TYPE('.F. OR .T.')
L
? TYPE('ANSWER=42')
U

```

UNLOCK

Purpose	Releases a record lock, multiple record locks or a file lock from the current or specified table/.DBF, or releases all record and file locks from all open table/.DBFs
Syntax	UNLOCK [IN <expN> <expC> ALL]
See Also	FLOCK()

Description UNLOCK releases a record lock, multiple record locks or a file lock. Record and file locks can be removed from a table/.DBF only by the user who issued the locks. UNLOCK cannot unlock a table/.DBF opened for exclusive use.

A record lock (or locks) or a file lock is released from the table/.DBF in the current work area if you issue UNLOCK without arguments.

If relations are established between tables/.DBFs, releasing a record lock (or locks) or a file lock from one of the tables/.DBFs doesn't release locks from the related records or files. You must explicitly release the record or file locks in each of the related files. You can release all locks on any related tables/.DBFs by issuing UNLOCK ALL (this also releases locks in unrelated files).

Clauses **IN <expN> | <expC>**

If you don't include <expN> or <expC>, UNLOCK releases a record lock or locks or a file lock from a table/.DBF in the current work area. Specify a work area number with <expN> or a table/.DBF alias with <expC> to release a record lock or locks or a file lock from a table/.DBF in another work area.

ALL

Include ALL to release all record and file locks in all work areas.

For additional information about record and file locking and sharing tables/.DBFs on a network, see the FoxPro in a Multi-User Environment chapter in the FoxPro *Developer's Guide*.

Example

```
CLOSE DATABASES
SET REPROCESS TO 3 AUTOMATIC
STORE '1,2,3,4' TO mreclist    && Lock records 1 through 4
old_exc = SET('EXCLUSIVE')    && Save the EXCLUSIVE setting
SET EXCLUSIVE OFF              && Allow sharing of tables/.DBFs

SELECT 0
USE salesman

SELECT 0
USE customer

? LOCK('1,2,3,4', 'customer') && Lock the first 4 records in customer
? RLOCK(mreclist, 'salesman') && Lock the first 4 records in salesman

UNLOCK IN customer
UNLOCK IN salesman

SET EXCLUSIVE &old_exc        && Restore original EXCLUSIVE setting
```

UPDATE

Purpose	Updates fields in the current table/.DBF with data from another table/.DBF
Syntax	UPDATE ON <field> FROM <expN> <expC> REPLACE <field1> WITH <expr1> [, <field2> WITH <expr2> ...] [RANDOM]
See Also	INDEX, SORT

Description UPDATE updates fields in the current table/.DBF with data from a table/.DBF open in another work area.

Clauses **ON <field>**

To use UPDATE, the current table/.DBF and the table/.DBF you update from must have a common field. <field> specifies the common field that controls the update. The current table/.DBF must be indexed or sorted in ascending order on the common field. Update performance is improved if the update table/.DBF is also sorted or indexed.

FROM <expN> | <expC>

The current table/.DBF is updated with data from a table/.DBF open in another work area. Specify the work area number (<expN>) or the table/.DBF alias (<expC>) of the table/.DBF open in the other work area.

REPLACE <field1> WITH <expr1> ...

Issuing UPDATE replaces a field (<field1>) in the current table/.DBF with an update expression (<expr1>). You can update more than one field in the current table/.DBF by including a list of fields (<field2>, <field3>, and so on) and corresponding update expressions (<expr2>, <expr3>, and so on).

The update expressions usually are the names of fields from the update table/.DBF. They can also be general expressions or constants.

Note that for each record in the current table/.DBF, there can be multiple matching records in the update table/.DBF. If there are multiple matching records, the record in the current table/.DBF is updated by each of the matching records. If the current table/.DBF contains identical key field records, only the first of the matching records is updated.

RANDOM

You must include the RANDOM key word if the update table/.DBF isn't indexed or sorted in ascending order.



An incorrect number of records may be updated if the update table/.DBF isn't indexed or sorted in ascending order and RANDOM isn't included.

UPDATED()

Purpose	Returns true (.T.) if you changed any data during the most recent READ
Syntax	UPDATED()
Returns	Logical
See Also	@ ... GET, INKEY(), READ, READKEY(), SHOW GETS

Description UPDATED() returns true (.T.) if data in any controls was changed during the most recent READ: It returns true if the data in a field, spinner or text editing region was changed or if a choice was made from a check box; an invisible, push or radio button; a list; or a popup.

UPDATED() returns false (.F.) if the data in a field, spinner or text editing region wasn't changed or no choices were made from a check box; an invisible, push or radio button; a list; or a popup during the most recent READ. UPDATED() returns false if a control was updated from within a program (for example, with REPLACE or SHOW GETS).

UPPER()

Purpose	Returns the specified character expression in upper-case
Syntax	UPPER(<expC>)
Returns	Character
See Also	ISLOWER(), ISUPPER(), LOWER()

Description UPPER() returns an upper-case character string from a character expression. Each lower-case letter (a – z) in the character expression is converted to upper-case (A – Z) in the returned string. All other characters remain unchanged.

Parameter <expC>
<expC> is the character expression you want to convert to upper-case.

Example ? UPPER('abcdefgh')
ABCDEFGH

USE

Purpose	Opens a table/.DBF file and associated index files
Syntax	USE [<file> ?] [IN <expN1>] [AGAIN] [INDEX <index file list> ? [ORDER [<expN2> <idx index file> [TAG] <tag name> [OF <cdx file>] [ASCENDING DESCENDING]]]] [ALIAS <alias>] [EXCLUSIVE] [SHARE] [NOUPDATE]
See Also	CREATE, DBF(), USED()

Description USE opens a table/.DBF file in the current work area.

If USE is issued without a table/.DBF name and a table/.DBF file is open in the current work area, the table/.DBF is closed. A table/.DBF is also closed when another table/.DBF is opened in the same work area.

Clauses <file> | ?

The name of the table/.DBF to open is specified with <file>. The Open dialog appears with a list of available tables/.DBFs to choose from if you include ? instead of a table/.DBF name.

IN <expN1>

You can open a table/.DBF in a non-current work area by specifying the work area number with <expN1>. You can close a table/.DBF in a non-current work area by issuing USE without a file name and specifying the work area number.

The IN clause supports 0 as a work area. Including 0 opens a table/.DBF in the lowest available work area. For example, if tables/.DBFs are open in work areas one through ten, the command

```
USE customer IN 0
```

opens the client table/.DBF in work area 11. This single command is equivalent to issuing these two commands:

```
SELECT 0  
USE customer
```

AGAIN

To open a table/.DBF concurrently in multiple work areas, you can do one of the following:

- Select another work area and issue USE with the table/.DBF name and the AGAIN clause.
- Issue USE with the table/.DBF name and the AGAIN clause, and specify a different work area.

When you use a table/.DBF again in another work area, the table/.DBF in the new work area takes on the attributes of the table/.DBF in the original work area. For example, if a table/.DBF is opened for read-only or exclusive access and is used again in another work area, the table/.DBF is opened for read-only or exclusive access in the new work area.

Index files opened for the original table/.DBF are available for the table/.DBF you open again if you don't open indexes when you reopen the table/.DBF. The index order is set to 0 in the work areas where the table/.DBF is reopened.

You can open indexes that weren't opened with the original table/.DBF. This sets the index order to 0 for the original table/.DBF.

A reopened table/.DBF is assigned the default alias of the work area. You can include an alias every time you open a table/.DBF in multiple work areas as long as the aliases are unique.

INDEX <index file list> | ?

You can open index files with the table/.DBF by including INDEX and specifying a set of indexes with <index file list>. If a table/.DBF has a structural compound index file, the index file is automatically opened with the table/.DBF. You can open a single index file by issuing INDEX ?. The Open dialog appears with a list of available index files.

<index file list> can contain any combination of names of .IDX and .CDX index files. You don't have to include the file name extensions for index files unless an .IDX and a .CDX index file in the index file list have the same name.

The first index file named in the index file list is the controlling index file, which controls how records in the table/.DBF are accessed and displayed. If the first index file is a .CDX compound index file, records in the table/.DBF are displayed and accessed in physical record order.

ORDER [<expN2>]

Include the ORDER clause to specify a controlling index file or tag other than the first index file in <index file list>. <expN2> specifies the controlling tag or index file as it appears in the index file list. .IDX index files are numbered first in the order in which they appear in the index file list. Tags in the structural compound index file (if one exists) are then numbered in the order in which the tags were created. Finally, tags in any independent compound index files are numbered in the order in which they were created. You can also use SET ORDER to specify the controlling index file or tag. See SET ORDER in this chapter for a further discussion of the numbering of index files and tags.

If <expN2> is 0, records in the table/.DBF are displayed and accessed in physical record order, and the indexes remain open. Including ORDER 0 enables open index files to be updated while presenting the file in record number order. Including ORDER without <expN> is identical to including ORDER 0.

ORDER [idx index file]**ORDER [[TAG <tag name>] [OF <cdx file>]]**

You can also designate an .IDX index file as the controlling index file by specifying its name with <idx index file>.

To designate a tag of a .CDX index file as the controlling tag, specify the tag name with <tag name>. The tag name can be from the structural compound index file or any open compound index file. If identical tag names exist in open compound index files, include OF <cdx file> and specify the name of the compound index file containing the desired tag.

ASCENDING | DESCENDING

Include the ASCENDING or DESCENDING key word after the ORDER clause to specify whether the table/.DBF records are accessed and displayed in ascending or descending order. Including one of these key words doesn't change the index file or tag; it alters only the order in which records are displayed and accessed.

ALIAS <alias>

Include ALIAS <alias> to create an alias for the table/.DBF. You can refer to a table/.DBF by its alias in commands and functions that require or support an alias.

When a table/.DBF is opened, it is automatically assigned an alias, which is the table/.DBF name if ALIAS isn't included. You can create a different alias for the table/.DBF by including ALIAS and a new alias. An alias can contain up to 10 letters, digits or underscores and must begin with a letter or an underscore.

A default alias is assigned automatically if you open a single table/.DBF simultaneously in multiple work areas with AGAIN and you don't specify an alias when you open the table/.DBF in each work area.

A default alias is also assigned if a conflict occurs. For example:

```
CLOSE DATABASES
ACTIVATE WINDOW View           && Open the View Window
USE customer ALIAS invoices IN 1 && Alias is INVOICES
USE invoices IN 3              && Conflict, alias is C
```

EXCLUSIVE

Including EXCLUSIVE opens a table/.DBF for exclusive use on a network. See SET EXCLUSIVE for more information on the exclusive use of tables/.DBFs.

SHARE

Including SHARE opens a table/.DBF for shared use on a network. SHARE allows you to open a table/.DBF for shared use even when EXCLUSIVE is set ON. See SET EXCLUSIVE for more information on exclusive and shared use of tables/.DBFs.

NOUPDATE

Including NOUPDATE prevents changes to the table/.DBF its structure.

Example

This example opens three tables in three different work areas. The View window is opened to show where the tables are open and their aliases.

```
CLOSE DATABASES
ACTIVATE WINDOW View
SELECT A
USE customer IN 0
USE invoices IN 0
USE salesman IN 0
```

USED()

Purpose	Returns true (.T.) if a table/.DBF is open in a specified work area
Syntax	USED([<expN> <expC>])
Parameters	<expN> The work area of a table/.DBF <expC> A table/.DBF alias
Returns	Logical
See Also	ALIAS(), SELECT()

Description USED() returns true (.T.) if a table/.DBF is open in a specified work area; it returns false (.F.) if a table/.DBF isn't open in the specified work area.

Parameters <expN> | <expC>

The current work area is checked for an open table/.DBF if you don't specify a work area or alias. To test for an open table/.DBF in another work area, specify the table's work area with <expN> or its alias with <expC>.

If you include a table alias, USED() returns true if a table/.DBF with the alias you specify is open in one of the work areas.

Example

```
CLOSE DATABASES
SELECT A
USE customer
SELECT B
USE invoices
SELECT C
USE salesman
? USED(1)
.T.

? USED('B')
.T.

? USED(4)
.F.
```

User-Defined Functions (UDFs)

Purpose	Creates new functions
See Also	FUNCTION, PARAMETERS, PARAMETERS(), PROCEDURE, RETURN, SET UDFPARMS

Description Although FoxPro includes over 200 built-in functions, there are times when you'll want to create specialized functions for your programs. These functions are called user-defined functions (UDFs). User-defined functions let you extend the FoxPro language.

UDFs are a powerful part of the FoxPro language. They can be included in many FoxPro commands, enhancing their versatility. For example, issuing BROWSE opens a Browse window in which you view and change information in a table/.DBF. You can include a UDF in BROWSE to restrict and validate data entered in the Browse window.

What Is a UDF?

A UDF is a FoxPro program that returns a value to the *calling program* (the program that executes the UDF). A value can be returned to the calling program with RETURN <expr>. A UDF can be a stand-alone program or a procedure or function in a program.

Constructing a UDF

If a UDF is a function or a procedure, the first line must be a FUNCTION or PROCEDURE command that assigns the function or procedure a name. Any number of statements can follow this first line. See FUNCTION and PROCEDURE in this manual for additional information on creating functions and procedures in FoxPro.

UDFs shouldn't be given the same name or abbreviation as a built-in FoxPro function because the built-in function takes precedence — the FoxPro function is executed, not the UDF. Also, UDFs shouldn't be given the same name as a memory variable or an array.

Returning a Value

A value can be returned to the calling program with RETURN <expr>.

Be sure to return the proper type data to the calling program. For example, if the calling program expects a numeric value to be returned by the UDF and the UDF returns a character value, an error is generated.

A true (.T.) value is automatically returned to the calling program if a UDF doesn't return a value (because RETURN wasn't included or a value wasn't included with RETURN).

Parameter Passing

Data can be passed in the form of *parameters* to a UDF. A PARAMETERS statement in the UDF identifies the data passed to the UDF and assigns local names to the data.

When parameters are passed to a UDF and the UDF is a stand-alone program, the first executable line in the UDF must be a PARAMETERS statement. If the UDF is a function or a procedure, a PARAMETERS statement must be the first executable line following the FUNCTION or PROCEDURE declaration. A maximum of 24 parameters can be passed to a UDF.

Variable Parameters

A variable number of parameters can be passed to a UDF. Use PARAMETERS() to determine the number of parameters passed to the UDF.

The number of parameters passed to a UDF can be less than the number of parameters listed in the PARAMETERS statement. In that case, the remaining parameters are initialized to false (.F.). If the number of parameters passed to the UDF is greater than the number of parameters listed in the PARAMETERS statement, the error message "Wrong number of parameters" is displayed.

The following UDF, called HOWMANY(), accepts up to 10 parameters. PARAMETERS() is used to return the number of parameters passed to the UDF:

```
FUNCTION howmany
PARAMETERS n1, n2, n3, n4, n5, n6, n7, n8, n9, n10  && Up to 10
RETURN PARAMETERS( )
```

Executing HOWMANY() with four parameters returns 4:

```
? HOWMANY('A', 'B', 'C', 'D')
```

Memory variables or array elements can be passed as parameters by reference or by value. By default, they are passed to a UDF by value. When memory variables and array elements are passed to a UDF by value, the values of the variables and array elements can be changed within the UDF, but aren't changed in the calling program.

When variables or array elements are passed by reference and the UDF changes the values of the passed parameters, the values of the variables or array elements in the calling program *are* changed.

Issue SET UDFPARMS TO REFERENCE before calling the UDF to pass a parameter by reference to a UDF.

See PARAMETERS and PARAMETERS() in this chapter for additional information on passing parameters to a UDF. See SET UDFPARMS for information on passing parameters by value or reference.

Calling a UDF

Like all FoxPro functions, UDFs are referenced by their name followed by a set of parentheses. The parentheses can contain parameters that are passed to the UDF. See the table at the end of this section for a list of commands that support UDFs.

UDFs in Logical Expressions

If you include one or more UDFs in a logical expression, the UDFs are evaluated from left to right.

When two operands (which can be UDFs) are connected with the AND relational operator, evaluation of the expression ends when an operand evaluates to false (.F.). For example, the two UDFs below return logical values and are connected with the AND operator:

```
logical1( ) AND logical2( )
```

If LOGICAL1() returns true, LOGICAL2() is evaluated. LOGICAL2() isn't evaluated if LOGICAL1() returns false.

If two operands are connected with the OR relational operator, evaluation of the expression ends when an operand evaluates to true (.T.).

UDFs in Commands

Many FoxPro commands support UDFs. UDFs let you execute routines that can validate data, display a message, enable or disable objects and so on. The following example demonstrates how a UDF can be used in a VALID clause in BROWSE to ensure that a proper tax rate is entered.

Example

A temporary table named TEMPDATA is created and BROWSE is issued to open a Browse window for the table. The VALID clause calls the RIGHTCOST() UDF, which tests whether a proper cost has been entered in the COST field. Costs can range from 5.00 to 20.00.

If the value you enter in the COST field doesn't fall within this range, a message is displayed with the proper tax rate range. The cursor can't move to another record until a proper tax rate is entered.


```

CLOSE DATABASES
CREATE TABLE tempdata (item c(10), cost n(5,2))
INSERT INTO tempdata (item,cost) VALUES ('widget',10.00)
INSERT INTO tempdata (item,cost) VALUES ('flimjan',18.00)
GO TOP
BROWSE FIELDS item, cost:F VALID rightcost( )
CLOSE DATABASES
ERASE tempdata.dbf
RETURN

```

FUNCTION rightcost && Called by BROWSE VALID

```

IF NOT BETWEEN(cost, 5.00, 20.00)    && Bad cost entered
    WAIT WINDOW 'Enter cost from 5.00 to 20.00' NOWAIT
    RETURN 0    && Stay on current record
ELSE    && Good cost entered
    RETURN .T.    && Okay to move to another field or record
ENDIF

```

Where UDFs Can Be Used

UDFs are supported only in certain clauses of some commands. The following table provides a complete list of commands and clauses that support UDFs.

Command/Function	Clauses
All commands supporting these clauses.	<row> <column> COLOR SCHEME FOR WHILE
= assignment operator	Example: <mvar> = UDF()
? and ?? commands	<expr1>, <expr2> ..., AT
??? command	<expC>
@ ... SAY	SAY <expr1>
@ ... GET	DEFAULT ERROR MESSAGE RANGE VALID WHEN
@ ... GET – check boxes, invisible buttons, push buttons, and radio buttons	DEFAULT MESSAGE VALID WHEN

Command/Function	Clauses
@ ... GET – lists and popups	DEFAULT MESSAGE RANGE VALID WHEN
@ ... GET – text editing regions	DEFAULT ERROR MESSAGE VALID WHEN
AVERAGE	<expr list>
BROWSE CHANGE EDIT	FIELDS <field list> – calculated fields :V – Verify option :E – Error option :P – Picture option :H – Heading option :W – When option ERROR FOR TIMEOUT TITLE VALID WHEN Format files in BROWSE, if specified
DEFINE BAR DEFINE PAD	PROMPT MARK MESSAGE SKIP FOR
DEFINE MENU	MARK MESSAGE
DEFINE POPUP	FOOTER MARK MESSAGE PROMPT FIELD TITLE
DEFINE WINDOW	FILL FOOTER TITLE

Command/Function	Clauses
DISPLAY FIELDS LIST FIELDS	Calculated fields
DO	WITH <parm list>
DO CASE	CASE <expL1>, <expr2> ...
DO WHILE	WHILE <expL>
EVALUATE()	Will evaluate a UDF
FOR	Initial <expN1> and final <expN2> counter values
IF	IF <expL>
INDEX	Index expression <expr>
KEYBOARD	Expression (<expC>) stuffed into the keyboard buffer
LABEL	Field expressions
PLAY MACRO	TIME <expN>
READ	ACTIVATE DEACTIVATE SHOW TIMEOUT VALID WHEN
REPLACE	WITH <expr1>, <expr2> ...
REPORT	Field expressions Report variables Grouping expressions
RETURN	RETURN <expr>
SET COLOR OF SCHEME	Second color scheme expression (<expN2>)
STORE	Stored expression (<expr>)
SUM	<expr list>
WAIT	Message expression (<expC>) TIMEOUT <expN>

VAL()

Purpose	Returns a numeric expression from a specified character expression composed of numbers
Syntax	VAL(<expC>)
Returns	Numeric
See Also	SET DECIMALS, STR()

Description VAL() returns a numeric expression that corresponds to the value of the numbers contained in a character expression. VAL() processes the numbers in the character expression from left to right until a non-numeric character is encountered (leading blanks are ignored). VAL() returns 0 if the first character of the character expression isn't a number.

Parameter <expC>
Specify a character expression composed of eight numbers with <expC>.

Examples

```
STORE '12' TO A
STORE '13' TO B
? VAL(A) + VAL(B)
25.00

STORE '1.25E3' TO C
? 2 * VAL(C)
2500.00
```

VARREAD()

Purpose	Returns the name of the memory variable, array element, or field used to create the current @ ... GET field or control
Syntax	VARREAD()
Returns	Character
See Also	@ ... GET, BROWSE, CHANGE, EDIT, SYS(18)

Description This function returns in upper-case the name of the memory variable, array element or field used to create the current @ ... GET field or control. Controls include check boxes; fields; invisible, push or radio buttons; lists; popups; spinners; and text editing regions.

If a Browse, Change or Edit window is active, the name of the current field is returned with the first letter of the field name capitalized.

VARREAD() is identical to SYS(18). Both can be used to pass the current field or control name to a procedure that in turn can provide context-sensitive help, as in the following example.

Example

```

SET TALK OFF
CLOSE DATABASES
ON KEY LABEL F1 DO Help WITH VARREAD( )
USE customer
SCATTER TO temp
DEFINE WINDOW input FROM 6,10 to 18,70 PANEL
ACTIVATE WINDOW input
@ 1,3 SAY 'Customer: ' GET company
@ 3,3 SAY 'Address: ' GET address
@ 5,3 SAY 'City: ' GET city
@ 7,3 SAY 'State: ' GET state
@ 7,18 SAY 'Zip: ' GET zip
@ 9,7 SAY 'Press <Escape> to cancel or <F1> for help'
READ
GATHER FROM temp
DEACTIVATE WINDOW input
RELEASE WINDOW input
ON KEY LABEL F1
RETURN

```

*** Help Procedure ***

```
PROCEDURE Help
PARAMETERS fieldname
DO CASE
    CASE fieldname = 'COMPANY'
        WAIT WINDOW 'Enter the company name' NOWAIT
    CASE fieldname = 'ADDRESS'
        WAIT WINDOW 'Enter the street address' NOWAIT
    CASE fieldname = 'CITY'
        WAIT WINDOW 'Enter the city name' NOWAIT
    CASE fieldname = 'STATE'
        WAIT WINDOW 'Enter a two character state abbreviation' NOWAIT
    CASE fieldname = 'ZIP'
        WAIT WINDOW 'Enter the five digit Zip code' NOWAIT
ENDCASE
RETURN
```

VERSION()

Purpose	Returns the number of the version of FoxPro you are using
Syntax	VERSION()
Returns	Character
See Also	GETENV(), OS()

Description VERSION() returns a character string containing the FoxPro version number.

Use VERSION() to conditionally execute version-specific portions of code.

Example ? VERSION()
 FoxPro 2.5

WAIT

Purpose Halts FoxPro operation until you press a key or click the mouse

Syntax

```

WAIT
  [<expC>]
  [TO <memvar>]
  [WINDOW [NOWAIT]]
  [CLEAR]
  [TIMEOUT <expN>]

```

See Also ACCEPT

Description WAIT halts operation and displays a message until a key is pressed or the mouse is clicked.

Clauses <expC>

Include <expC> to display your own message. If <expC> is omitted, the default system message “Press any key to continue...” is displayed.

TO <memvar>

Include TO <memvar> to save the key pressed to a memory variable or an array element. If the memory variable or an array element you specify with <memvar> doesn't exist, it is created. The null string is stored in <memvar> if you press Enter or a nonprintable key or key combination.

WINDOW

If you include WINDOW, the message is displayed in the system message window in the upper-right corner of the desktop in FoxPro for MS-DOS or the main FoxPro window in FoxPro for Windows. The window can be temporarily hidden by pressing the Ctrl or Shift key. In FoxPro for MS-DOS, you can move the window to a different location on the desktop by dragging it with the mouse or by pressing the Ctrl+F7 key combination.

NOWAIT

Include NOWAIT to create a message that behaves like a FoxPro system message, which is displayed in the upper-right corner of the desktop or main FoxPro window. An example of a FoxPro system message is the “Not found” message displayed after an unsuccessful search in a text editor or a Browse window.

As with a FoxPro system message, you can move the mouse or press a key to remove the message. A message created with NOWAIT doesn’t discard the keystroke you use to remove the message from the desktop or main FoxPro window. The keystroke is discarded if NOWAIT is omitted.

CLEAR

CLEAR provides a method to remove a system window or a WAIT window message from within a program. For example, talk from indexing, sorting and so on is directed to a system window if you issue SET TALK WINDOW. The window is removed when you press a key or move the mouse. Issue WAIT CLEAR to remove the window from within a program.

TIMEOUT <expN>

Include TIMEOUT to specify how long operation is halted. The numeric expression <expN> specifies how many seconds (fractional seconds are permitted) can elapse without input from the keyboard or the mouse before the WAIT is terminated. If TIMEOUT is included but isn’t the last clause in WAIT, a syntax error occurs.

Example

The following example uses WAIT WINDOW to display a message whenever the beginning or the end of CUSTOMER.DBF is encountered. Press **Forward** several times to advance through the table, then press **Backward** several times to take you to the beginning of the table. A message is displayed when you reach the beginning of the table.

```

SET TALK OFF
CLOSE DATABASES
CLEAR
USE customer
STORE .F. TO mexit
DEFINE WINDOW mskip FROM 9,30 TO 17,50 DOUBLE
DO WHILE NOT mexit
    ACTIVATE WINDOW mskip
    @ 1,5 PROMPT 'Forward '
    @ 3,5 PROMPT 'Backward '
    @ 5,5 PROMPT 'Quit    '
    MENU TO mchoice
    DEACTIVATE WINDOW mskip
    DO CASE
        CASE mchoice = 1
            SKIP
            IF EOF( )
                WAIT WINDOW 'End of file encountered'
                SKIP -1
            ENDIF
        CASE mchoice = 2
            SKIP - 1
            IF BOF( )
                WAIT 'Beginning of file encountered' WINDOW
                GO TOP
            ENDIF
        CASE MCHOICE = 3
            STORE .T. TO mexit
            LOOP
        ENDCASE
    @ 20,10 SAY 'Company: ' + company
ENDDO
RELEASE WINDOW mskip

```

WBORDER()

Purpose	Determines whether or not the specified window has a border
Syntax	WBORDER([<window name>])
Returns	Logical
See Also	ACTIVATE WINDOW, DEFINE WINDOW

Description This function returns true (.T.) if the specified window has a border, and it returns false (.F.) if the window doesn't have a border. By default, windows have borders. Include the NONE clause in DEFINE WINDOW to create a window without a border.

Parameter <window name>
<window name> is the name of the window whose border is in question.

WBORDER() returns a logical value for the current output window if a window name isn't included.

WCHILD()

Purpose	Returns either the number and names of child windows for the specified or current window
Syntax	WCHILD([<window name>] [<expN>])
Parameters	<window name> The parent window name <expN> 0 for the bottom child window A positive number for remaining child windows
Returns	Character or Numeric
See Also	ACTIVATE WINDOW, DEFINE WINDOW, WPARENT()

Description WCHILD() returns either the number of child windows in a parent window or the names of the child windows in the order in which they are stacked in the parent window.

You can create a window (the parent window) and place other windows (child windows) inside. Including the IN or IN WINDOW clause in DEFINE WINDOW creates a child window inside the parent window. A child window created and activated inside a parent window cannot be moved outside the parent window. If the parent window is moved, the child window moves with it.

The number of child windows in the current output window is returned if you issue WCHILD() without any arguments.

Parameters <window name>

To return the number or names of child windows for a window other than the current output window, specify the window's name with <window name>. The number of child windows in the specified window is returned if you include just a window name without the numeric expression <expN>. If you include both <window name> and <expN>, WCHILD() returns the names of the child windows in the specified window.

<expN>

WCHILD() returns the names of child windows in the current output window if you include <expN> but not <window name>.

The numeric expression <expN> can be 0 or any positive value. The name of the child window at the bottom of the stack of child windows in the current output window is returned if <expN> is 0.

If <expN> is a positive number, WCHILD() returns the name of the next child window in the window stack. The name of next child window in the stack is returned if you issue WCHILD() again with a positive number and so on. The null string is returned if WCHILD() is called more times than the number of child windows in the parent window. See ACTIVATE WINDOW for more information on window stacking.



If the desktop or main FoxPro window is active, all windows are children of the desktop or main FoxPro window. Issuing a series of WCHILD() functions with positive numbers returns the names of the active windows.

If you include both a window name and <expN>, separate the window name and <expN> with a comma.

WCOLS()

Purpose	Returns the number of columns within the specified window
Syntax	WCOLS([<window name>])
Returns	Numeric
See Also	SCOLS(), SROWS(), WLCOL(), WLROW(), WROWS()

Description WCOLS() returns the number of columns in a window.

If you don't specify a window, the number of columns in the active output window is returned. If no window is active, the number of columns on the desktop in FoxPro for MS-DOS or the main FoxPro window in FoxPro for Windows is returned.

Parameter <window name>

WCOLS() returns the number of columns in the window you specify with <window name>.

In FoxPro for Windows, the value returned by WCOLS() depends on the font specified for the window. Most fonts Microsoft Windows uses can be displayed in a wide variety of sizes, and some are proportionally spaced. A column corresponds to the average width of a letter in the current font. See the Font topic in the FoxPro online help for additional information about fonts.

You can specify the name of a system window (the Command window, the View window, a Browse window, and so on) in WCOLS() if the system window has been opened and is visible or is hidden. If you specify the name of a system window that is closed, the alert "WINDOW <window name> has not been defined" is displayed. The Debug window is an exception. Once the Debug window has been opened, its name can be included in WCOLS() if it is visible, hidden or closed.

You can also include the null string as the window name to return the number of columns on the desktop or main FoxPro window.

Note that the null string can be used to specify the desktop or main FoxPro window in functions that return window locations or sizes (WLCOL(), WLROW(), WCOLS(), WROWS(), and so on).

Example This example centers a short output message in the last row of a window of unknown size.

```
DO SendMesg WITH 'Message', WCOLS( ), WROWS( ) - 1
*** SendMesg ***
PROCEDURE SendMesg
PARAMETERS msg_string, w_cols, row
STORE (w_cols - LEN(msg_string))/2 TO col
@ row, col SAY msg_string
RETURN
```

WEXIST()

Purpose	Determines whether or not the definition of the specified window is in memory
Syntax	WEXIST(<window name>)
Returns	Logical
See Also	ACTIVATE WINDOW, DEFINE WINDOW, WONTOP(), WOUTPUT(), WVISIBLE()

Description WEXIST() returns true (.T.) if the window you specify exists, and it returns false (.F.) if the window doesn't exist. The specified window doesn't have to be active or visible for WEXIST() to return true, but it must be defined.

Parameter <window name>

Specify the name of the window with <window name>.

You can specify the name of a system window (the Command window, the View window, a Browse window, and so on) in WEXIST(). WEXIST() returns true (.T.) if the system window you specify is visible or hidden. WEXIST() returns false (.F.) if the specified system window is closed.

Two exceptions are the Command and Debug windows. Including the Command window's name in WEXIST() always returns a true value. If the Debug window has been opened, WEXIST() returns true, even if the Debug window is closed.

Examples

```
DEFINE WINDOW screen1 FROM 10,10 TO 20,69
DEFINE WINDOW screen2 FROM 1,0 TO 19,79
? WEXIST('screen1')
.T.
```

```
STORE 'screen2' TO win_name
? WEXIST(win_name)
.T.
```

```
? WEXIST('screen3')
.F.
```

```
RELEASE WINDOWS screen1, screen2
```


WFONT()

Purpose	Returns the name, size or style of the current font for a window in FoxPro for Windows
Syntax	WFONT(<expN> [, <window name>])
Parameters	<expN> Font attribute number <window name> Window name
Returns	Character and Numeric
Remarks	FoxPro for Windows only
See Also	FONTMETRIC(), GETFONT(), TXTWIDTH(), SYSMETRIC()

Description Use WFONT() to return the font name, font size or font style for the active output window or a specified window.

Parameters **<expN>**

<expN> specifies the font attribute you want to return. If <expN> is 1, WFONT() returns the name of the current font for the active or specified window. If <expN> is 2, WFONT() returns the font size. If <expN> is 3, WFONT() returns a code that identifies the font style.

The *font style code* is a character or set of characters that correspond to the current font style. For example, the current font style is bold italic if WFONT(3) returns BI.

Here are the codes for each font style:

Code	Font Style
B	Bold
I	Italic
N	Normal
O	Outline
Q	Opaque
S	Shadow
—	Strikeout
T	Transparent
U	Underline

<window name>

<window name> is the name of the window for which you want to determine the current font, font size or font style. Include the null string to return the current font, font size or font style for the main FoxPro window.

<window name> can be the name of a user-defined window created with DEFINE WINDOW or a text or memo editing window.

You can also include the name of a FoxPro system window (View, Trace, Debug, and so on). WFONT() can return font attributes only for a FoxPro system window that has been opened and is currently visible or hidden. If the system window you specify is closed, the alert “WINDOW <window name> has not been defined” is displayed.

WFONT() returns the current font, font size or font style for the active output window if a window name is omitted.

Example The following example creates a user-defined window called FONTCHAR. The window is activated and its font characteristics are displayed in the window. The font characteristics are then displayed for the main FoxPro window.

```

DEFINE WINDOW fontchar ;
    FROM 1,1 TO 3,35 ;
    FONT 'MS SANS SERIF',8 ;
    STYLE 'BI'    && Define window with font and style
ACTIVATE WINDOW fontchar

?WFONT(1), WFONT(2), WFONT(3)                && Fontchar window
ACTIVATE SCREEN
?
?
?
? 'Font characteristics for the window FONTCHAR'
?
?
?
? WFONT(1,''), WFONT(2,''), WFONT(3,'')      && Main FoxPro window
?
? 'Font characteristics for the main FoxPro window'
?
?
?
WAIT WINDOW
RELEASE WINDOW fontchar
CLEAR

```

WLAST()

Purpose	Returns the name of the window that was active prior to the current window or determines whether or not the named window was active prior to the current window
Syntax	WLAST([<window name>])
Returns	Character or Logical
See Also	ACTIVATE WINDOW, DEFINE WINDOW, WEXIST(), WONTOP(), WOUTPUT(), WVISIBLE()

Description Use WLAST() to determine if a specific window was active prior to the current window. You can also use WLAST() to determine which window was active prior to the current window.

WLAST() returns the null string if the window that was active prior to the current window is the Debug, Trace or Command window. Program execution isn't be affected by bringing these windows forward when you are debugging a program that uses WLAST().

Parameter <window name>

To determine if a specific window was active prior to the current window, specify the window's name with <window name>. WLAST() returns true (.T.) if the specified window was active prior to the current window. It returns false (.F.) if the specified window wasn't active prior to the current window or if the window you specify doesn't exist.

The name of the window that was active prior to the current window is returned if you omit <window name>.

WLCOL()

Purpose	Returns the column coordinate of the top-left corner of the current or specified window
Syntax	WLCOL ([<window name>])
Returns	Numeric
See Also	SET DISPLAY, WCOLS(), WLROW(), WROWS()

Description WLCOL() returns the column coordinate of the top-left border of a window. The column coordinate is relative to the desktop in FoxPro for MS-DOS or the main FoxPro window in FoxPro for Windows.

The value returned by WLCOL() depends on the current video display mode. In FoxPro for MS-DOS, the display mode can be changed with SET DISPLAY. In FoxPro for Windows, the display mode can be changed with Windows Setup.

Windows can be positioned off the desktop or main FoxPro window. If the specified window is to the left of the desktop or main FoxPro window, WCOL() returns a negative value. If the window's left border is to the right of the desktop or main FoxPro window, WCOL() returns a positive value greater than the width of the desktop or main FoxPro window.

Parameter <window name>

The column coordinate of the top-left of the active output window is returned if <window name> isn't included. If no window is active, the column coordinate of the top-left border of the desktop or the main FoxPro window is returned. The column coordinate for the desktop or the main FoxPro window is also returned if the null string is included as the window name.

If you specify a window name with <window name>, the column coordinate of the top-left of the specified window is returned.

If a FoxPro system window (the Command window, the View window, a Browse window, and so on) has been opened and is visible or hidden (its name appears on the **Window** menu), you can include its name in WLCOL(). If you specify the name of a FoxPro system window that is closed, the alert "WINDOW <window name> has not been defined" is displayed. The Debug window is an exception. Once the Debug window has been opened, its name can be included in WLCOL() if it is visible, hidden or closed.

WLROW()

Purpose	Returns the row coordinate of the top-left corner of the current or specified window
Syntax	WLROW([<window name>])
Returns	Numeric
See Also	SET DISPLAY, WCOLS(), WLCOL(), WROWS()

Description WLROW() returns the row coordinate of the top-left border of a window. The row coordinate is relative to the desktop in FoxPro for MS-DOS or the main FoxPro window in FoxPro for Windows.

The value returned by WLROW() depends on the current video display mode. In FoxPro for MS-DOS, the display mode can be changed with SET DISPLAY. In FoxPro for Windows, the display mode can be changed with the Windows Setup.

Windows can be positioned off the desktop or main FoxPro window. If the top of the window is above the desktop or main FoxPro window, WLROW() returns a negative value. If the top of the window is below the bottom of the desktop or main FoxPro window, WLROW() returns a positive value greater than the height of the desktop of main FoxPro window.

Parameter <window name>

The row coordinate of the top border of the active output window is returned if <window name> isn't included. If no window is active, the row coordinate of the top-left border of the desktop or the main FoxPro window is returned. The row coordinate for the desktop or the main FoxPro window is also returned if the null string is included as the window name.

If you specify a window name with <window name>, the row coordinate of the top-left border of the specified window is returned.

If a FoxPro system window (the Command window, the View window, a Browse window, and so on) has been opened and is visible or hidden (its name appears on the **Window** menu), you can include its name in WLROW(). If you specify the name of a FoxPro system window that is closed, the alert "WINDOW <window name> has not been defined" is displayed. The Debug window is an exception. Once the Debug window has been opened, its name can be included in WLROW() if it is visible, hidden or closed.

WMAXIMUM()

Purpose	Determines whether or not the current or specified window is maximized
Syntax	WMAXIMUM([<window name>])
Returns	Logical
See Also	DEFINE WINDOW, WMINIMUM(), ZOOM WINDOW

Description Windows can be enlarged to fill the entire desktop in FoxPro for MS-DOS or main FoxPro window in FoxPro for Windows.

A user-defined window created with DEFINE WINDOW can be maximized only if the ZOOM key word is included in its definition.

WMAXIMUM() returns true (.T.) if the current or specified window is maximized. It returns false (.F.) if the window isn't maximized.

In FoxPro for MS-DOS, you can maximize a window by choosing the **Zoom** ↑ option from the **Window** menu, by pressing Ctrl+F10, by clicking on the zoom control or by issuing ZOOM WINDOW.

In FoxPro for Windows, you can maximize a window by choosing the **Maximize** option from the window's **Control** menu, by pressing Ctrl+F10, by clicking on the Maximize button, by double-clicking on the top border of the window or by issuing ZOOM WINDOW.

Parameter **<window name>**

To determine whether a specific window is maximized, specify the window's name with <window name>. You can specify the name of a FoxPro system window (the Command window, the View window, a Browse window, and so on).

A logical value is returned for the current window if <window name> isn't included in WMAXIMUM().

WMINIMUM()

Purpose	Determines whether or not a current or specified window is minimized
Syntax	WMINIMUM([<window name>])
Returns	Logical
See Also	DEFINE WINDOW, WMAXIMUM(), ZOOM WINDOW

Description Windows can be reduced to a minimize size in FoxPro for MS-DOS or to an icon in FoxPro for Windows.

A user-defined window created with DEFINE WINDOW can be minimized only if the MINIMIZE key word is included in its definition.

WMINIMUM() returns true (.T.) if the current or specified window is minimized. It returns false (.F.) if the window isn't minimized.

In FoxPro for MS-DOS, you can minimize a window by choosing the **Zoom** ↓ option from the **Window** menu, by pressing Ctrl+F9, by double-clicking on the top border of the window or by issuing ZOOM WINDOW.

In FoxPro for Windows, you can minimize a window by choosing the **Minimize** option from the window's **Control** menu, by pressing Ctrl+F9, by clicking on the Minimize button or by issuing ZOOM WINDOW.

Parameter <window name>

To determine whether a window is minimized, specify the window's name with <window name>. You can specify the name of a FoxPro system window (the Command window, the View window, a Browse window, and so on).

A logical value is returned for the current window if <window name> isn't included in WMINIMUM().

WONTOP()

Purpose	Determines whether or not the current or specified window is in front of all other active windows
Syntax	WONTOP([<window name>])
Returns	Character or Logical
See Also	ACTIVATE WINDOW, DEFINE WINDOW, WEXIST(), WOUTPUT(), WVISIBLE()

Description WONTOP() returns true (.T.) if the specified window is frontmost. It returns false (.F.) if the window isn't frontmost or doesn't exist.

Parameter <window name>

If <window name> isn't included in WONTOP(), the name of the frontmost window is returned.

To determine if a window is frontmost, specify the window's name with <window name>. You can specify the name of a FoxPro system window (the Command window, the View window, a Browse window, and so on).

If the Debug, Trace or Command window is frontmost, WONTOP() (when used without a window name) returns the null string. When you are debugging a program that uses WONTOP(), program execution shouldn't be affected by bringing these windows forward.

Example This program example uses WONTOP() to display the name of the frontmost window. Before you run this program, open some windows (Browse windows, the View window, and so on) or desk accessories.

```
IF NOT EMPTY(WONTOP( ))
    WAIT WINDOW 'Foremost window: ' + WONTOP( )
ELSE
    WAIT WINDOW 'No windows are open'
ENDIF
```

WOUTPUT()

Purpose	Determines whether or not output is being directed to the current or specified window
Syntax	WOUTPUT([<window name>])
Returns	Character or Logical
See Also	ACTIVATE WINDOW, DEFINE WINDOW, WEXIST(), WONTOP(), WVISIBLE()

Description This function returns true (.T.) if the specified user-defined window is the current output window. The last user-defined window activated with ACTIVATE WINDOW is the current output window.

Parameter **<window name>**

WOUTPUT() returns the name of the window to which output is currently directed if <window name> is omitted. WOUTPUT() returns the null string if output is being directed to the desktop in FoxPro for MS-DOS or the main FoxPro window in FoxPro for Windows.

To determine whether output is being directed to a window, specify the window's name with <window name>. False (.F.) is returned if the window you specify doesn't exist or is a FoxPro system window. Output cannot be directed to a FoxPro system window.

Example In this example, a window is created and activated. WOUTPUT() is used to display the name of this active output window. The window is closed and removed from memory. If another window is active, its name is displayed. If another window isn't active, a message is displayed indicating that output is directed to the desktop or the main FoxPro window.

```
DEFINE WINDOW output FROM 2,2 TO 12,32 TITLE 'Output Window'
ACTIVATE WINDOW output
```

```
WAIT WINDOW 'Output window: ' + WOUTPUT( )
```

```
RELEASE WINDOW output
```

```
IF EMPTY(WOUTPUT( ))
    WAIT WINDOW 'Output being directed to the desktop'
ELSE
    WAIT WINDOW 'Output window: ' + WOUTPUT( )
ENDIF
```

WPARENT()

Purpose	Returns the name of the parent window of the current or specified window
Syntax	WPARENT([<window name>])
Returns	Character
See Also	ACTIVATE WINDOW, DEFINE WINDOW, WCHILD()

Description This function returns the name of the parent window of a child window.

You can use DEFINE WINDOW to create a window and place it in a parent window. The child window becomes integrated with its parent window. For example, a child window defined and activated inside a parent window can't be moved outside the parent window. If the parent window is moved, the child window moves with it.

Parameter <window name>

To return the parent name for a window, specify the window's name with <window name>. The null string is returned if the specified window doesn't have a parent window.

If <window name> is omitted, the name of the parent window of the current output window is returned. The null string is returned if the current output window doesn't have a parent window.

Example The following example defines a parent and child window. It then uses WPARENT() to identify which window is the parent.

```
CLEAR ALL
CLEAR
DEFINE WINDOW test1 ;
    FROM 1,1 TO 20,20 ;
    TITLE 'Test1'      && Parent window
ACTIVATE WINDOW test1
DEFINE WINDOW test2 ;
    FROM 1,1 TO 10,10 ;
    TITLE 'Test2' ;
    IN WINDOW test1    && Child window
ACTIVATE WINDOW test2

WAIT WINDOW 'The parent window is ' + WPARENT( )
RELEASE WINDOW test1, test2
```

WREAD()

Purpose	Determines whether or not the current or specified window is involved in the current READ
Syntax	WREAD([<window name>])
Returns	Logical
See Also	RDLEVEL(), READ, READKEY(), WONTOP()

Description WREAD() returns true (.T.) if the specified window is participating in the current READ. False (.F.) is returned if the specified window isn't involved in the current READ or doesn't exist.

You can have a window participate in a READ by:

- Including the window's title in the READ WITH clause. Windows that typically aren't involved in a READ can participate in this way. Browse windows, memo and text editing windows, FoxPro system windows and so on can participate in a READ.
- Creating a control in the window. A READ can span multiple windows if you create controls in multiple windows and then issue READ.

Parameter <window name>

To determine whether a window is participating in a READ, specify the name of the window with <window name>.

If a window name isn't included, WREAD() returns a logical value for the window that is currently frontmost (that is, the window whose name is returned by WONTOP()).

WROWS()

Purpose	Returns the number of rows within a specified window
Syntax	WROWS([<window name>])
Returns	Numeric
See Also	SCOLS(), SROWS(), WCOLS(), WLCOL(), WLROW()

Description WROWS() returns the number of rows in a window.

If you don't specify a window, the number of rows in the active output window is returned. If no window is active, the number of rows in the desktop in FoxPro for MS-DOS or the main FoxPro window in FoxPro for Windows is returned.

Parameter <window name>

WROWS() returns the number of rows in the window you specify with <window name>.

In FoxPro for Windows, the value returned by WROWS() depends on the font specified for the window. Most fonts Microsoft Windows uses can be displayed in a wide variety of sizes, and some are proportionally spaced. A row corresponds to the height of the current font. See the Font topic in the FoxPro online help for additional information about fonts.

You can specify the name of a system window (the Command window, the View window, a Browse window, and so on) in WROWS() if the system window has been opened and is visible or is hidden. If you specify the name of a system window that is closed, the alert "WINDOW <window name> has not been defined" is displayed. The Debug window is an exception. Once the Debug window has been opened, its name can be included in WROWS() if it is visible, hidden or closed.

You can also include the null string as the window name to return the number of rows on the desktop or main FoxPro window.

Note that the null string can be used to specify the desktop or main FoxPro window in functions that return window locations or sizes (WLCOL(), WLROW(), WCOLS(), WROWS(), and so on).

Example See the example in WCOLS() in this chapter.

WTITLE()

Purpose	Returns the title assigned to the current or specified window
Syntax	WTITLE([<window name>])
Returns	Character
See Also	ACTIVATE WINDOW, DEFINE WINDOW, WEXIST(), WLAST(), WONTOP(), WOUTPUT(), WREAD()

Description WTITLE() can be used to return a window’s title, which appears in the top border of the window. The title is returned for the frontmost window if <window name> isn’t included in WTITLE(). WTITLE() returns the null string if the Debug, Trace or Command window is frontmost or output is being directed to the desktop.

Parameter **<window name>**
To return the title for a window, specify the name of the window with <window name>. The title assigned to the window with the TITLE clause is returned if you include the name of a user-defined window.

When you issue BROWSE WINDOW to open a Browse window in a user-defined window, the title of the user-defined window is returned by WTITLE() if the Browse window doesn’t have a title. The title of the Browse window is returned if the Browse window does have a title.

Window Names and Titles

Names are assigned to user-defined windows, system windows and Browse windows in the following manner:

User-defined windows are assigned names when they are created with DEFINE WINDOW. There is a distinction between user-defined window names and titles. By default, user-defined windows don’t have titles. If the TITLE clause is included when the window is created, the specified title appears in the top border of the window but isn’t the name of the window.

By default, each system window, which are part of the FoxPro interface, derives its name from the title of the window. Examples of system windows include the Command window, the View window, the Trace window, the Filer, the Help window, and the Puzzle.

CREATE or MODIFY COMMAND, REPORT, LABEL and FILE windows derive their names from the name of the file being created or modified. The name of the window includes the file name extension.

A Browse window derives its name from the title of the window. The name and title of the Browse window is assigned in one of three ways:

1. The default title assignment of the table/.DBF alias;
2. The window title, if any; or
3. The Browse title, if any.

By default, the Browse window name is the table alias.

Some Additional Notes on Window Names

If you are unsure of the name assigned to a window, check the **Window** menu. All window names are listed at the bottom of the **Window** menu.

Two windows exist if you issue BROWSE WINDOW <window name>. The Browse window is a separate window and takes on the attributes of the specified user-defined window. If a window is active when you issue BROWSE and you don't include the WINDOW clause, the Browse window takes on the attributes of the active window. You can override this behavior by including NORMAL in BROWSE.

You can include window names that contain spaces in commands and functions that accept window names (MOVE WINDOW, DEACTIVATE WINDOW, WONTOP(), and so on) by specifying the part of the window name that begins with the first non-space character and continuing until the first non-space character is encountered.

For example, a Browse window with the name Invoice Entry can be moved with this command:

```
MOVE WINDOW invoice BY 1,1
```

The names of user-defined windows cannot contain spaces, but the names of Browse and System windows can.

WVISIBLE()

Purpose	Determines whether or not the specified window has been activated and isn't hidden
Syntax	WVISIBLE(<window name>)
Returns	Logical
See Also	ACTIVATE WINDOW, DEFINE WINDOW, WEXIST(), WONTOP(), WOUTPUT()
Description	This function returns true (.T.) if the specified window is shown or activated and isn't hidden. Windows are shown and activated with SHOW WINDOW and ACTIVATE WINDOW. WVISIBLE() returns false (.F.) if the window has not been activated, has been hidden with HIDE WINDOW, has been deactivated with DEACTIVATE WINDOW, or doesn't exist.
Parameter	<window name> <window name> is the name of the window whose visibility you want to determine.
Example	This example uses WVISIBLE() to display the names of all visible desk accessories (the Puzzle, the Calculator, and so on) in a window. Open some of the FoxPro desk accessories before you run this program. <pre> STORE '' TO mvisible STORE mvisible + IIF(WVISIBLE('ASCII'), ' ASCII Chart', '') ; TO mvisible STORE mvisible + IIF(WVISIBLE('CALCULATOR'), ' Calculator', '') ; TO mvisible STORE mvisible + IIF(WVISIBLE('CALENDAR'), ' Calendar', '') ; TO mvisible STORE mvisible + IIF(WVISIBLE('FILER'), ' Filer', '') ; TO mvisible STORE mvisible + IIF(WVISIBLE('PUZZLE'), ' Puzzle', '') ; TO mvisible STORE mvisible + IIF(WVISIBLE('SPECIAL'), ' Special Characters', '') ; TO mvisible </pre>


```
IF EMPTY(mvisible)
    WAIT WINDOW 'No Desk Accessories are visible'
ELSE
    WAIT WINDOW 'Visible:' + mvisible
ENDIF
```

YEAR()

Purpose	Returns the year from the specified date expression
Syntax	YEAR(<expD>)
Returns	Numeric
See Also	DATE(), DOW(), SET CENTURY
Description	YEAR() returns the numeric year from a date expression. YEAR() always returns the year with the century. The CENTURY setting (ON or OFF) doesn't affect the returned value.
Parameter	<expD> Specify a date expression with <expD>. The date expression may be a function that returns a date, or a date type memory variable, array element or field. It can also be a literal date string (for example, {06/06/92}).
Examples	? YEAR(DATE()) 1992

ZAP

Purpose	Removes all records from a table/.DBF
Syntax	ZAP
See Also	DELETE, PACK

Description ZAP removes all records from the current table/.DBF, leaving just the table/.DBF structure. Issuing ZAP is equivalent to issuing DELETE ALL followed by PACK, but ZAP is much faster.

If SET SAFETY is ON, FoxPro asks if you would like to remove the records from the current table/.DBF.



Records zapped from the current table/.DBF cannot be recalled.

ZOOM WINDOW

Purpose	Changes the size and position of a user-defined window or a FoxPro system window
Syntax	ZOOM WINDOW <window name> MIN [AUTO] MAX NORM [AT <row1, column1> FROM <row1, column1> [SIZE <row2, column2> TO <row2, column2>]]
See Also	ACTIVATE WINDOW

Description This command lets you change the size and position of system and user-defined windows. Windows can be reduced to a minimum size, enlarged to fill the entire desktop in FoxPro for MS-DOS, the main FoxPro window in FoxPro for Windows or can be sized anywhere in between. If you create a user-defined window with **DEFINE WINDOW** and the **IN DESKTOP** clause in FoxPro for Windows, the window you create can be enlarged to fill the entire Windows desktop.

Windows can be “zoomed” directly from minimum to maximum size and vice versa. You can also specify where to place the resized window on the desktop, main FoxPro window or user-defined window.

You can use **ZOOM WINDOW** to resize all system windows except the Filer, the Calculator, and the Puzzle.

Clauses <window name>

<window name> is the name of the window whose size you want to change.

MIN

Include **MIN** to reduce the window to minimum size. In FoxPro for MS-DOS, just the name of the window or the window’s title remains. In FoxPro for Windows, the window is reduced to an icon.

Reducing a window to a minimum size makes it easy to place many windows on the desktop, the main FoxPro window, the Windows desktop or in a user-defined window simultaneously and still have ample space for output. For example, you can minimize the FoxPro desk accessories (the Calculator, the Calendar/Diary, the Puzzle, and so on) and place them in the lower portion of the desktop or main FoxPro window for easy access and use.

All system windows can be reduced to a minimum size. System windows must be open on the desktop, the main FoxPro window or a user-defined window before they can be minimized.

A user-defined window can be minimized after it is defined. It doesn't have to be activated before you change its size.

AUTO

In FoxPro for MS-DOS, if you include AUTO and MIN, the window is minimized and docked in the lower-right corner of the desktop. If AUTO is included with MAX or NORM, it is ignored.

AUTO is ignored in FoxPro for Windows.

MAX

Include MAX to expand a window to fill the entire desktop, the main FoxPro window, the Windows desktop or a user-defined window. If a child window is placed in a parent window and the child is expanded to its maximum size, it expands to fill the parent window. If any of the additional ZOOM WINDOW clauses (AT, SIZE, TO or FROM) are included with MAX, it is ignored.

Only user-defined windows defined with ZOOM can be zoomed to maximum size.

NORM

Include NORM to return a window to its original size after it has been minimized or maximized. NORM can also be used to move or resize a window without changing its size. Use ZOOM WINDOW NORM without any additional clauses to return a minimized or maximized window to its original size and location.

AT <row1, column1> | FROM <row1, column1>

You can specify the placement of a window by including the optional AT or FROM clauses. In FoxPro for MS-DOS, if NORM or MIN are included, the left edge of the window is placed on the desktop at the location specified with <row1, column1>. If MAX is included, AT and FROM are ignored.

In FoxPro for Windows, if NORM is used, the left edge of the window is placed in the main FoxPro window at the location specified with <row1, column1>. If MIN is included, the window is displayed as an icon in the lower portion of the main FoxPro window. AT and FROM are ignored. If MAX is used, AT and FROM are ignored.

In FoxPro for Windows, if the window is created with the IN DESKTOP clause (for example, DEFINE WINDOW mywind FROM 10,10 TO 20,20 IN DESKTOP), the left edge of the window is placed in the Windows desktop at the location specified with <row1, column1>. If MIN is included, the window is displayed as an icon in the lower portion of the Windows desktop. AT and FROM are ignored. If MAX is included, AT and FROM are ignored.

ZOOM WINDOW <window name> NORM AT <row1, column1> restores a minimized or maximized window to its original size at a specific location. The <row1, column1> coordinates specify where the upper left corner of the window is placed. A window's location can also be changed with MOVE WINDOW. See MOVE WINDOW in this chapter for more information.

SIZE <row2, column2> | TO <row2, column2>

You can also specify a window size that differs from the original by including SIZE or TO. If SIZE is included, the window is <row2> rows high and <column2> columns wide. If the TO clause is included, the lower-right corner of the window is placed at the position specified with <row2, column2>.

Example

In this example, a Browse window is opened for CUSTOMER.DBF. The Browse window is minimized, returned to its default size, minimized at a specific location, enlarged to a specific size and then maximized.

```
CLOSE DATABASES
CLEAR ALL
CLEAR
USE customer
BROWSE NORMAL NOWAIT
```

```
ZOOM WINDOW customer MIN
WAIT WINDOW TIMEOUT 3 ;
'MIN clause - This window will timeout. Please wait.'
```

```
ZOOM WINDOW customer NORM
WAIT WINDOW TIMEOUT 3 ;
'NORM clause - This window will timeout. Please wait.'
```

```
ZOOM WINDOW customer MIN AT 10,10
WAIT WINDOW TIMEOUT 3 ;
'MIN AT 10,10 clause - This window will timeout. Please wait.'
```

ZOOM WINDOW customer NORM AT 1,1 SIZE 22,25

WAIT WINDOW TIMEOUT 3 ;

'NORM & SIZE clauses - This window will timeout. Please wait.'

ZOOM WINDOW customer NORM FROM 10,10 TO 22,70

WAIT WINDOW TIMEOUT 3 ;

'NORM & TO clauses - This window will timeout. Please wait.'

ZOOM WINDOW customer MAX

WAIT WINDOW TIMEOUT 3 ;

'MAX clause - This window will timeout. Please wait.'

CLEAR ALL

4 System Memory Variables Alphabetically

System memory variables are “built-in” memory variables that FoxPro creates and maintains automatically. System memory variables are by default PUBLIC; you can declare a system memory variable PRIVATE.

The following table lists each FoxPro system memory variable, its type and its default value.

Variable	Type	Default Value
_ALIGNMENT	C	LEFT
_BEAUTIFY	C	BEAUTIFY.APP
_BOX	L	.T.
_CALCMEM	N	0.0
_CALCVALUE	N	0.0
_CLIPTEXT	C	Null string
_CUROBJ	N	0
_DBLCLICK	N	0.5
_DIARYDATE	D	Current date
_DOS	L	.T. in FoxPro for MS-DOS
_FOXDOC	C	FOXDOC.EXE
_GENGRAPH	C	GENGRAPH.PRG
_GENMENU	C	GENMENU.PRG
_GENPD	C	GENPD.APP
_GENSCRN	C	GENSCRN.PRG
_GENXTAB	C	GENXTAB.PRG
_INDENT	N	0
_LMARGIN	N	0
_MAC	L	.T. in FoxPro for Macintosh
_MLINE	N	0
_PADVANCE	C	FORMFEED
_PAGENO	N	1
_PBPAGE	N	1
_PCOLNO	N	Current column
_PCOPIES	N	1
_PDRIVER	C	Null string
_PDSETUP	C	Null string
_PECODE	C	Null string
_PEJECT	C	BEFORE

Variable	Type	Default Value
_PEPAGE	N	32,767
_PLENGTH	N	66
_PLINENO	N	0
_PLOFFSET	N	0
_PPITCH	C	DEFAULT
_PQUALITY	L	.F.
_PRETEXT	C	Null string
_PSCODE	C	Null string
_PSPACING	N	1
_PWAIT	L	.F.
_RMARGIN	N	80
_STARTUP	C	FOXSTART.APP
_TABS	C	Null string
_TALLY	N	0
_TEXT	N	-1
_THROTTLE	N	0
_TRANSPORT	C	TRANSPRT.PRG
_UNIX	L	.T. in FoxPro for UNIX®
_WINDOWS	L	.T. in FoxPro for Windows
_WRAP	L	.F.

_ALIGNMENT

Purpose	Aligns text between margins
Syntax	<code>_ALIGNMENT = <expC></code>
Remarks	Included for backward compatibility — use the Report Writer instead
See Also	<code>_LMARGIN</code> , <code>_RMARGIN</code> , <code>_WRAP</code>

Description `_ALIGNMENT` contains a character value that determines how output is aligned between the current margins.

`<expC>` must evaluate to one of the following:

- `LEFT` — Produces left justified output (default).
- `CENTER` — Produces centered output.
- `RIGHT` — Produces right justified output.

`_ALIGNMENT` is active only when `_WRAP` is set to true (.T.).

_BEAUTIFY

Purpose	Specifies a beautification application for FoxPro programs
Syntax	_BEAUTIFY = <program name>
Remarks	FoxPro for Windows only
See Also	_FOXDOC, MODIFY COMMAND

Description In FoxPro for Windows, the **Program** menu has a **Beautify...** option that lets you make cosmetic changes to the appearance of a program. You can change the capitalization of key words and variables to upper-case or lower-case, automatically indent items by the number of spaces you specify, or use established tab settings for indents. Extra indents can be added automatically beneath procedures or DO CASE statements, and you can choose to leave existing FoxDoc headings unchanged or delete them automatically.

You can also choose to add action diagram symbols to your program to give you a quick visual check on the organization of your program. You can choose **Beautify...** as many times as you want, selecting different combinations of options each time to create the appearance you want for your program.

The Beautify dialog contains an **Action diagram symbols** check box. Action diagrams are ordinary program files that have been modified to connect structured programming commands such as IF ... ENDIF with vertical lines. Choosing this check box also causes the Beautify application to do additional checking to ensure the structured programming commands are balanced — for example, an ENDIF for every IF.

Action diagrams will not compile in FoxPro, so be sure to remove the action diagram symbols before saving your program. If don't remove the action diagram symbols, you can run Beautify again with the **Action diagram symbols** check box unchecked. This will remove the symbols.

The _BEAUTIFY system memory variable contains the name of the application that FoxPro for Windows uses when you choose the **Beautify...** option. By default, _BEAUTIFY contains BEAUTIFY.APP, installed in your FoxPro for Windows directory.

To specify a different beautification application, store the application's name in _BEAUTIFY. If your beautification application is in a directory other than the current default directory, include a path with the application name.

If you rename BEAUTIFY.APP or move it to another directory, store the new BEAUTIFY.APP file name and its path in _BEAUTIFY.

You can also specify a beautification application in your FoxPro for Windows CONFIG.FPW configuration file. Include the line:

```
_BEAUTIFY = <program name>
```

The name of your beautification application is specified with <program name> and can include a path to the beautification application.

Ordinarily you won't need to use _BEAUTIFY. If you write a different version of a beautification program, you can have FoxPro run your beautification application by storing its name in _BEAUTIFY.

Your beautification application must accept a single parameter, the name of the FoxPro program to beautify. Your beautification application can modify the FoxPro program, write the modifications to a temporary file and then return the name of the temporary file to FoxPro. FoxPro reads the temporary file back into the editing session under the original FoxPro program name and then deletes the temporary file.

_BOX

Purpose	Prints boxes
Syntax	_BOX = <expl>
Remarks	Included for backward compatibility — use the Report Writer instead
See Also	@ ... BOX, @ ... TO, DEFINE BOX

Description _BOX contains a logical value that determines whether or not boxes (as specified with the DEFINE BOX command) are printed around lines of text. If _BOX is set to .T., the default, then boxes are printed; otherwise, they are not.

This system memory variable affects only boxes that you create with DEFINE BOX.

You can draw boxes on the screen or in a window with @ ... BOX or @ ... TO.

_CALCMEM

Purpose	Returns or initializes the Calculator memory value
Syntax	<code>_CALCMEM = <expN></code>
See Also	ACTIVATE WINDOW, _CALCVALUE, STORE

Description _CALCMEM contains the numeric value that FoxPro stores in the Calculator's memory. You can save the result of a calculation in the Calculator's memory and return the result to a program, or you can place a specific value in the Calculator's memory before using the Calculator.

To initialize the Calculator's memory, store a numeric value to _CALCMEM with STORE or the = assignment operator. When you use the Calculator, its memory contains the value you specified.

Example The following program example stores the numeric constant 1234 to _CALCMEM. The program then displays the Calculator and stuffs the keyboard with the letter R. The keystroke R displays the value stored in the Calculator's memory.

```
STORE 1234 TO _CALCMEM
ACTIVATE WINDOW calculator
CLEAR TYPEAHEAD
KEYBOARD CHR(82)
```

_CALCVALUE

Purpose	Returns or initializes the Calculator display value
----------------	---

Syntax	<u>_CALCVALUE</u> = <expN>
---------------	---

See Also	ACTIVATE WINDOW, _CALCMEM, STORE
-----------------	----------------------------------

Description _CALCVALUE contains the numeric value that the Calculator displays. You can return the result of a calculation to a program or you can place a specific value in the Calculator's display before using the Calculator.

To initialize the Calculator's display, store a numeric value to _CALCVALUE with STORE or the = assignment operator. When you use the Calculator it displays the value you specified.

Example In the following example the numeric value 1234 is stored to _CALCVALUE. The Calculator then displays the value 1234.

```
STORE 1234 TO _CALCVALUE  
ACTIVATE WINDOW calculator
```


_CLIPTEXT

Purpose	Returns or initializes the clipboard contents
Syntax	_CLIPTEXT = <expC>
See Also	PRIVATE, PUBLIC, STORE

Description The clipboard is a temporary storage area for text. When you copy text from a text editing area, the text is placed on the clipboard. The contents of the clipboard can then be pasted into another text editing area. With _CLIPTEXT, you can return the contents of the clipboard or STORE a character expression on the clipboard.

Some examples of where you can cut, copy or paste text are the FoxPro text editor, memo editing windows, the Browse and Debug windows, dialogs, the Expression Builder and active GETs. With the Capture desk accessory in FoxPro for DOS, you can copy to the clipboard any text displayed on the desktop or in a user-defined window.

_CLIPTEXT contains the contents of the clipboard. You can place a character expression <expC> on the clipboard with STORE or the = assignment operator. _CLIPTEXT by default is a PUBLIC memory variable; it can be declared PRIVATE.

Example The following program example displays the clipboard in a window.

```
*** clipboard display example ***

*** Initialize variables & define windows ***

SET TALK OFF
STORE _CLIPTEXT TO ctext
STORE 1 TO windtop
*** Define an outline & scrolling window for the clipboard ***
DEFINE WINDOW outline FROM 1,0 to 4,79;
    TITLE " clipboard " CLOSE
DEFINE WINDOW clipwind FROM -1,-1 TO 30,78

ACTIVATE WINDOW outline
ACTIVATE WINDOW clipwind IN outline && Place in outline window
```

```
*** Show the clipboard in the clipboard window ***

IF EMPTY(ctext)                                && clipboard empty
  @ 1,1 SAY "clipboard is empty!"
ELSE
  IF AT(CHR(13),ctext) = 0                      && No carriage returns (CR)
    @ 1,1 SAY ctext                            && Show clipboard
  ELSE
    STORE AT(CHR(13), ctext, 1) to crpos1 && 1st CR
    @ 1,1 SAY SUBSTR(ctext, 1, crpos1 - 1) && Show 1st line
    FOR lineno = 1 to 19
      STORE AT(CHR(13), ctext, lineno + 1) to crpos2 && Next CR

*** The next line shows each additional clipboard line ***

      @ lineno,0 SAY SUBSTR(ctext, crpos1 + 1, (crpos2 - crpos1) - 1)
      STORE crpos2 to crpos1

    NEXT      && Loop for next clipboard line
  ENDIF      && Test for 0 or more carriage returns (CR)
ENDIF        && Test for empty clipboard

DO WHILE .T.  && Test for up or down arrows, mouse click, any key

  keyval = INKEY(5,'MH')      && Check for key press, mouse click
                                && or 5 seconds

  IF keyval = 24 AND BETWEEN(windtop,1,18) && Down arrow
    MOVE WINDOW clipwind BY -1,0 && Move clipboard window up
    windtop = windtop + 1
  ENDIF
  IF keyval = 5 AND BETWEEN(windtop,2,19) && Up arrow
    MOVE WINDOW clipwind BY 1,0 && Move clipboard window down
    windtop = windtop - 1
  ENDIF
  IF keyval != 5 AND keyval != 24 && quit out
    DEACTIVATE WINDOW outline,clipwind
    SET TALK ON
    EXIT
  ENDIF
ENDDO
```

_CUROBJ

Purpose	Returns or initializes the current GET object number
Syntax	_CUROBJ = <expN>
See Also	@ ... GET, OBJNUM(), READ, SHOW GET, SHOW OBJECT

Description _CUROBJ contains the number <expN> of the currently selected GET object. GET objects are created with @ ... GET and include fields, check boxes, push and invisible buttons, popups, lists and text editing regions. _CUROBJ is frequently queried or changed in a user-defined function called by a VALID or WHEN clause, or other READ and @ ... GET clauses.

You can return the value from _CUROBJ to determine which object is selected, or you can store a value to _CUROBJ to select a specific GET object.

The GET object number is determined by the order the GETs are issued. Each individual button in a set of push buttons, radio buttons or invisible buttons is considered a separate object.

You can change the value of _CUROBJ to position the cursor on another GET object. This is typically done in VALID and WHEN routines.

Example The following program example demonstrates how GET objects are numbered and how _CUROBJ can be used to return the object number of the current object. Press Tab or the arrow keys to select different objects. Pressing F2 displays the object number for the current object.

```
CLEAR
SET TALK OFF
STORE 1 to x, y, z

ON KEY LABEL F2 @ 12,2 SAY 'Object # ' + STR(_CUROBJ)    && Show #

@ 4,2 GET x  PICTURE '@*R \<Apples;\<Oranges;\<Limes'    && Radio
@ 2,2 GET y  && Get field
@ 8,2 GET z  PICTURE '@^ \<Pears;\<Bananas;\<Pineapples' && Popup

READ CYCLE

ON KEY LABEL F2  && Reset F2 key to default
SET TALK ON
```

_DBLCLICK

Purpose	Specifies the time interval between double and triple mouse clicks
Syntax	_DBLCLICK = <expN>
See Also	INKEY()

Description _DBLCLICK contains a numeric value that determines the time interval FoxPro uses to check for a double or triple mouse click. _DBLCLICK is the time interval *between* mouse clicks. For example, if _DBLCLICK is set to 0.5 seconds, you have 1/2 second to click twice for a double-click, and 1 second to click 3 times for a triple-click.

The larger the value of _DBLCLICK, the longer you can wait between the first and second click for FoxPro to interpret the two clicks as a double-click. If _DBLCLICK is set to a very small value, even quick double (or triple) clicks may be interpreted as two (or three) single clicks.

The numeric expression <expN> specifies seconds. Internal timing is measured in clock “ticks,” which are each about 1/18th of a second. If you store an integer or decimal value to _DBLCLICK, FoxPro may store it as a slightly different value due to rounding with “ticks.”

The default value for _DBLCLICK is 0.5 seconds. _DBLCLICK can range from 0.05 to 5.5 seconds (1 to 100 “ticks”), inclusive.

Example The following program example demonstrates how you can detect a single or double mouse click:

```
*** Double-click detection example ***

*** Establish the program environment ***

SET TALK OFF
SET ESCAPE OFF
CLEAR

*** Create and open a window ***

DEFINE WINDOW MOUSECHK FROM 4,4 TO 20,75 DOUBLE
ACTIVATE WINDOW MOUSECHK
@ 6,16 SAY 'Press a key or click the mouse to start'
@ 8,16 SAY '          or press the Escape key to quit'
```

```
inval1 = 0                                && Initialize inkey memory variable

*** Main program loop ***

DO WHILE .T.
    inval1 = INKEY(.06,'HM') && hide cursor, check mouse
    IF inval1 = 0                && No key or mouse press
        LOOP
    ENDIF
    IF inval1 = 27                && Escape key pressed, exit main loop
        EXIT
    ENDIF
    IF inval1 = 151                && Single mouse click
        row = MROW( )            && Get mouse pointer position
        col = MCOL( )
        TIMELIMIT = SECONDS( ) + _DBLCLICK && Time for double-click
        inval2 = 0
        DO WHILE SECONDS( ) < TIMELIMIT && Check for double mouse click
            inval2 = INKEY('HM')

*** Double mouse click and mouse pointer inside window ***

            IF inval2 = 151 AND col != -1 AND row != -1
                CLEAR
                @ 7,11 SAY 'Double-click at '
                ?? 'Row ' + ALLTRIM(STR(row)) + ', Column ' ;
                   + ALLTRIM(STR(col)) + ' of this window'
                EXIT
            ENDIF
        ENDDO

*** Single mouse click and mouse pointer inside window ***

        IF inval2 = 0 AND col != -1 AND row != -1
            CLEAR
            @ 7,11 SAY 'Single-click at '
            ?? 'Row ' + ALLTRIM(STR(row)) + ', Column ' ;
               + ALLTRIM(STR(col)) + " of this window"
        ENDIF
```

```
*** Mouse pointer outside of window ***

    IF col = -1 OR row = -1
        CLEAR
        @ 7,17 SAY 'Mouse clicked outside of this window'
        LOOP
    ENDIF
ENDIF

*** Keyboard character pressed ***

    IF inval1 # 151 AND BETWEEN(inval1, 0, 256) && Good CHR( ) value
        CLEAR
        @ 7,18 SAY 'Character typed:'
        ?? ' ' + CHR(inval1) + ', ASCII code ' + ALLTRIM(STR(inval1))
    ENDIF

ENDDO    && End of main loop

*** Restore original environment ***

SET TALK ON
SET ESCAPE ON
CLEAR WINDOWS

*** End of program ***
```

_DIARYDATE

Purpose	Returns or initializes the date of the Calendar/Diary
Syntax	_DIARYDATE = <expN>
See Also	STORE

Description _DIARYDATE contains the selected date in the Calendar/Diary. With this system memory variable, you can display the Calendar/Diary with a specific date selected, or you can return the selected Calendar/Diary date.

By default, the current date is stored in _DIARYDATE. When you open the Calendar/Diary, the current date is selected. You can store a different date to _DIARYDATE so that the Calendar/Diary opens with this new date selected.

Selecting a new date when the Calendar/Diary is open causes the new date to be stored to _DIARYDATE. When you close the Calendar/Diary, _DIARYDATE contains the last selected date.

Example In the following program example, the Calendar/Diary opens with October 31, 2001 selected and the value of _DIARYDATE appears below. The program then uses the DATE() function to select the current date, and displays _DIARYDATE. July 4, 1776 is then stored to _DIARYDATE and _DIARYDATE displays again.

```
CLEAR
SET TALK OFF
SET CENTURY ON
STORE {10/31/2001} TO _DIARYDATE
@ 24,0 SAY _DIARYDATE
ACTIVATE WINDOW calendar
WAIT WINDOW
STORE DATE( ) TO _DIARYDATE
@ 24,16 SAY _DIARYDATE
WAIT WINDOW
STORE {07/04/1776} TO _DIARYDATE
@ 24,32 SAY _DIARYDATE
```

_DOS

Purpose	Contains true (.T.) if you are using FoxPro for MS-DOS
Syntax	_DOS = <expL>
Remarks	New to FoxPro 2.5
See Also	_MAC, STORE, _UNIX, VERSION(), _WINDOWS

Description The _DOS system memory variable contains a logical true (.T.) if you are using FoxPro for MS-DOS. _DOS contains a logical false (.F.) if you are using a different FoxPro version (FoxPro for Macintosh, FoxPro for UNIX or FoxPro for Windows).

The value contained in _DOS cannot be changed with STORE or =.

_FOXDOC

Purpose	Specifies the name and location of FoxDoc, the automatic program documenter
Syntax	_FOXDOC = <program name>
See Also	_BEAUTIFY

Description With _FOXDOC you can specify the program that FoxPro uses when it documents FoxPro programs. By default, _FOXDOC uses FOXDOC.APP. You can include a path with the program name.

If you choose to install FoxDoc during supplemental program installation, FoxDoc and its associated files are installed in the FoxPro directory. If you rename FOXDOC.APP or move FoxDoc and its associated files to another directory, store the new FOXDOC.APP file name and the directory name in _FOXDOC.

For further information about FoxDoc, see the Documenting Applications with FoxDoc chapter in the FoxPro *Developer's Guide*.

_FOXGRAPH

Purpose	Specify the name and location of FoxGraph [®] , the optional graphing package
Syntax	_FOXGRAPH = <program name>
Remarks	FoxPro for MS-DOS only
See Also	_GENGRAPH, _GENXTAB

Description In FoxPro for MS-DOS, you can use _FOXGRAPH to specify the program (and its location) that FoxPro uses when it graphs the contents of FoxPro tables/DBFs. By default, _FOXGRAPH uses FOXGRAPH.EXE. You can include a fully qualified path with the program name.

By default, FoxGraph and its associated files are installed in the FoxPro directory. To run FoxGraph from the **Program** menu or the Command window, FoxGraph must be in the FoxPro directory or in a directory locatable along the DOS path. If you move FoxGraph and its associated files to another directory, store FOXGRAPH.EXE and its location in _FOXGRAPH.

_GENGRAPH

Purpose	Specifies the program used to output RQBE query results to Microsoft Graph (FoxPro for Windows) or FoxGraph (FoxPro for MS-DOS)
Syntax	_GENGRAPH = <program name>
See Also	CREATE QUERY

Description With _GENGRAPH you can specify the program that FoxPro uses to output RQBE query results to Microsoft Graph in FoxPro for Windows, or to FoxGraph in FoxPro for MS-DOS. By default, _GENGRAPH uses the program supplied with FoxPro, GENGGRAPH.PRG in FoxPro for MS-DOS or GENGGRAPH.APP in FoxPro for Windows. You can include a path with the program name.

Queries are created in the RQBE window. In the RQBE window, you can choose **Graph** from the **Output** popup to specify where the results of a query are directed. Query results can be directed to a graph created with Microsoft Graph, the graphing package for FoxPro for Windows, or with FoxGraph, the optional graphing package for FoxPro for MS-DOS.

For further information about creating a query in the RQBE window, see CREATE QUERY in this manual or the RQBE chapter in the *FoxPro User's Guide*.

_GENMENU

Purpose	Specifies a menu generation program
----------------	-------------------------------------

Syntax	_GENMENU = <program name>
---------------	--

See Also	CREATE MENU, _GENSCRN
-----------------	-----------------------

Description _GENMENU lets you specify the program that FoxPro uses to generate menu code from an .MNX table. By default, _GENMENU uses the program supplied with FoxPro, GENMENU.PRG.

You can include a path with the menu generation program name.

_GENPD

Purpose	Specifies a printer driver interface program for FoxPro
Syntax	_GENPD = <expC>
See Also	_PDRIVER, _PDSETUP, SET PDSETUP, SET PRINTER

Description The system memory variable _GENPD contains the name of the current printer driver interface program. By default, this program is the application supplied with FoxPro, GENPD.APP.

The program whose name is stored in _GENPD is executed when you do one of the following:

- Choose the **Printer Driver Setup** check box in the Printer Setup dialog.
- Choose the **Set Printer Driver** check box in the Report Writer's Page Layout or the Label Designer's Label Environment dialog.
- Issue SET PDSETUP.
- Store a printer driver setup name in _PDSETUP.
- Specify a default printer setup before starting FoxPro.

If you store the name of your printer driver interface program in _GENPD, your program is executed when one of the events listed above occurs.

If you do not include an extension with your printer driver setup program when you store its name in _GENPD, FoxPro looks for a version of the program in the following order and executes the program if found:

- An executable (.EXE) version of the program
- An application (.APP) version of the program
- A compiled (.FXP) version of the program
- The program (.PRG)

The printer driver interface program you specify in _GENPD can take virtually any form, but it must have a PARAMETERS statement as its first line. FoxPro passes two parameters to the program, a numeric value and name of the printer driver setup.

The first parameter is one of the following numeric values:

- FoxPro passes 0 to the program if you execute the program by choosing the **Printer Driver Setup** check box in the Printer Setup dialog, issue SET PDSETUP, or store a printer driver setup name in _PDSETUP.
- FoxPro passes 2 to the program if you choose the **Set Printer Driver** check box in the Report Writer's Layout dialog or in the Label Designer's Label Environment dialog.

To prevent recursion, it is a good idea to test the numeric parameter passed to the printer driver setup program. For example, you could store a printer driver program name in _PDRIVER that automatically executes the PDONLOAD procedure in the printer driver program. Your PDONLOAD procedure executes your printer setup interface application and passes it a value of 1.

The printer setup interface application tests the numeric value. If the value is 1, a printer driver program name *should not* be stored in _PDRIVER in the program. If, in this case, a printer driver program name *is* stored in _PDRIVER, PDONLOAD is automatically executed again and your printer driver setup program is executed again in an infinite loop.

The second parameter FoxPro passes to the printer driver setup program is the name of the printer driver setup specified in SET PDSETUP or stored to _PDSETUP. If you execute the printer driver setup program by choosing the **Printer Driver Setup** check box in the Printer Setup dialog, FoxPro passes a question mark (?) as the second parameter to the program specified in _GENPD.

_GENSCRN

Purpose	Specifies a screen generation program
----------------	---------------------------------------

Syntax	_GENSCRN = <program name>
---------------	--

See Also	CREATE SCREEN, _GENMENU
-----------------	-------------------------

Description _GENSCRN lets you specify the program that FoxPro uses to generate screen code from an .SCX file. By default, _GENSCRN uses GENSCRN.PRG.

You can include a path with the screen generation program name.

_GENXTAB

Purpose	Specifies the program used to output RQBE results in cross tabular format
Syntax	_GENXTAB = <program name>
See Also	CREATE QUERY, _GENGRAPH, SELECT – SQL

Description With _GENXTAB you can specify the program that FoxPro uses to output RQBE results to Microsoft Graph in FoxPro for Windows, FoxGraph in FoxPro for MS-DOS, a report form, or a table/.DBF in cross tabular format.

By default, _GENXTAB uses the program GENXTAB.PRG.

_INDENT

Purpose	Indents first line of each paragraph
Syntax	<code>_INDENT = <expN></code>
Remarks	Included for backward compatibility — use the Report Writer instead
See Also	<code>_LMARGIN</code> , <code>_PLOFFSET</code> , <code>_RMARGIN</code> , <code>SET MARGIN</code> , <code>_WRAP</code>

Description `_INDENT` contains a numeric value that instructs the ? command to indent (or outdent) the first line of each paragraph a specified number of characters from the left margin.

The value of `_INDENT` can range from the negative value of `_LMARGIN` to the length of the line minus one (`( _RMARGIN - _LMARGIN ) - 1`). Positive `_INDENT` values cause the output line to be indented. Negative values cause the line to be outdented. `_INDENT` is 0 by default.

`_INDENT` is active only when `_WRAP` is set to `.T`.

_LMARGIN

Purpose	Sets the left margin
Syntax	<code>_LMARGIN = <expN></code>
Remarks	Included for backward compatibility — use the Report Writer instead
See Also	?, _INDENT, _PLOFFSET, _RMARGIN, SET MARGIN, _WRAP

Description _LMARGIN contains a numeric value that determines the position of the left margin for output generated with the ? command. Valid _LMARGIN values range from 0 to 254.

When you send output to the printer, the _LMARGIN value represents the number of additional spaces between the current page offset and the text. The page offset is determined by _PLOFFSET or SET MARGIN.

_LMARGIN is active only when _WRAP is set to .T.

_MAC

Purpose	Contains true (.T.) if you are using FoxPro for Macintosh
Syntax	_MAC = <expL>
Remarks	New to FoxPro 2.5
See Also	_DOS, STORE, _UNIX, VERSION(), _WINDOWS

Description The _MAC system memory variable contains a logical true (.T.) if you are using FoxPro for Macintosh. _MAC contains a logical false (.F.) if you are using a different FoxPro version (FoxPro for MS-DOS, FoxPro for UNIX or FoxPro for Windows).

The value contained in _MAC cannot be changed with STORE or =.

_MLINE

Purpose	Sets the memo field offset for the MLINE() function
Syntax	_MLINE = <expN>
See Also	MEMLINES(), MLINE(), SET MEMOWIDTH

Description MLINE() returns one line of text from a memo field. MLINE() stores the location of its memo field offset to the _MLINE system memory variable. Use _MLINE as the second numeric argument in the MLINE() function to increase the performance of MLINE().

The startup default value for _MLINE is 0. Be sure to reset _MLINE to 0 before using _MLINE again with the MLINE() function.

For additional information about the use of _MLINE and an example of its use, see the MLINE() function in this manual.

PADVANCE

Purpose	Sets the page advance method
Syntax	_PADVANCE = <expC>
Remarks	Included for backward compatibility — use the Report Writer instead
See Also	EJECT, _PEJECT, _PLENGTH, _PLINENO, PROW()

Description _PADVANCE contains a character value that determines whether FoxPro advances to the top of a page by using a form feed character or by using line feeds.

<expC> must evaluate to one of the following:

- **FORMFEED** – Sends a form feed character when the page advances. FORMFEED is the default. This option uses the printer's internal top-of-form controls.
- **LINEFEEDS** – Sends a calculated number of line feeds when the page advances. This option positions the printer at the top of the next page by internally calculating how many line feeds are needed and then sending that number of continuous line feeds.

When calculating how many line feeds to send, FoxPro uses the formula (**_PLENGTH – _PLINENO**) in the following situations:

- When you issue **EJECT PAGE** without an **ON PAGE** handler
- When you issue **EJECT PAGE** with an **ON PAGE** handler and the line position is greater than the **ON PAGE** line value
- When **PRINTJOB** or **ENDPRINTJOB** cause **_PEJECT** to send a page eject to the printer

FoxPro uses the formula (**_PLENGTH – MOD(PROW(), _PLENGTH)**) when you issue:

- **EJECT**
- **SET DEVICE TO PRINTER** and force an eject with a subsequent **@ ... SAY**

Despite the setting of _PADVANCE, you can always send a form feed character to the printer by outputting a **CHR(12)**.

_PAGENO

Purpose	Returns or sets current page number
Syntax	_PAGENO = <expN>
See Also	ON PAGE, _PBPAGE, _PEPAGE, _PLENGTH, _PLINENO

Description _PAGENO contains a numeric value that determines the current page number. The startup default is 1. _PAGENO allows you to print page numbers in streaming output without defining, initializing and incrementing a memory variable for that purpose.

<expN> can be a value from 1 to 32,767, inclusive. _PAGENO, _PBPAGE and _PEPAGE work in unison. If _PAGENO is set (or incremented) such that it falls outside the range of _PBPAGE to _PEPAGE, FoxPro doesn't print any pages.

PBPAGE

Purpose	Returns or sets first page to print
Syntax	_PBPAGE = <expN>
Remarks	Included for backward compatibility — use the Report Writer instead
See Also	_PAGENO, _PEPAGE, PRINTJOB

Description _PBPAGE contains a numeric value that determines the first page to print. When _PBPAGE is greater than _PAGENO, no pages print. Instead, FoxPro produces pages internally and increments _PAGENO and other system memory variables. When _PAGENO reaches the value of _PBPAGE, printing begins.

The _PBPAGE value can range from 1 to 32,767, inclusive. The startup default is 1. The value of _PBPAGE must be less than or equal to _PEPAGE.

_PCOLNO

Purpose	Returns or sets current column number
Syntax	_PCOLNO = <expN>
Remarks	Included for backward compatibility — use the Report Writer instead
See Also	? ??, PCOL(), _PLINENO, SET PRINTER

Description _PCOLNO contains a numeric value that establishes the current column position. You can use _PCOLNO to determine the next column address for output. As FoxPro sends output to the print device, it automatically increments the _PCOLNO value to indicate the current column position. You can assign any integer value to _PCOLNO, from 0 to 255, in order to position output to a desired column. Assigning a value to _PCOLNO has the same effect as using the AT clause of the ? command.

Unlike the PCOL() function, which returns the current position of the printhead and does not change if SET PRINTER is OFF, _PCOLNO changes whenever output occurs. SET PRINTER has no effect on _PCOLNO.



When sending output to the printer with SET PRINTER ON, you can cause the printhead to backspace and overstrike previously printed output by manipulating the _PCOLNO value. To do so, _WRAP must be set to .F. and _PCOLNO must be assigned an appropriate value that is less than the current value.



If _WRAP is .T., assigning a _PCOLNO value that is less than the current value causes text in the internal buffer to be overwritten. In this case the text in the buffer is replaced and does not print.

_PCOPIES

Purpose	Returns or sets number of copies to print
Syntax	_PCOPIES = <expN>
Remarks	Included for backward compatibility — use the Report Writer instead
See Also	PRINTJOB ... ENDPRINTJOB

Description _PCOPIES contains a numeric value that determines how many copies of the output print. <expN> can be a value from 1 to 32,767, inclusive. The startup default is 1.

_PCOPIES has an effect only when you use it with PRINTJOB ... ENDPRINTJOB, and therefore has an effect only in FoxPro programs. Place the _PCOPIES assignment statement before the PRINTJOB to set the number of copies to print.

PDRIVER

Purpose	Specifies a printer driver program
Syntax	<code>_PDRIVER = <expC></code>
See Also	? I ??, SET PDSETUP

Description You can specify a printer driver program by storing the name of the program to _PDRIVER.

Storing the name of a printer driver program to _PDRIVER unloads the current printer driver program if one is loaded. When you store a printer driver program name in _PDRIVER, the PDONUNLOAD procedure in the currently loaded printer driver program executes (if it has a PDONUNLOAD procedure). The printer driver you store to _PDRIVER then loads, and its PDONLOAD procedure executes (if it has a PDONLOAD procedure).

When you store a printer driver program to _PDRIVER, the message “Printer driver installed” appears. To suppress this message, issue the following command before you store a printer driver program name to _PDRIVER:

```
SET NOTIFY OFF
```

_PDSETUP

Purpose	Loads or clears a printer driver setup
Syntax	_PDSETUP = <expC>
See Also	_GENPD, _PDRIVER, SET PDSETUP, SET PRINTER

Description _PDSETUP loads a printer driver setup or clears the current printer driver setup. A printer driver setup is a combination of settings that includes the printer driver program and printing information, such as page orientation, default font size and style, margins and so on.

Printer driver setups are stored in your FOXUSER.DBF resource file and you can interactively create them and assign them names in the Printer Setup Editing dialog.

You can also load or clear a printer driver setup with SET PDSETUP. To load a printer driver setup, store the name of the setup in _PDSETUP. You can clear the current printer driver setup by storing the null string in _PDSETUP. Storing the null string in _PDSETUP also clears the _PDPARMS array from memory.

If the printer driver setup name you store in _PDSETUP does not exist in your resource file, the current printer setup interface application executes. If you are using GENPD.APP, the default printer setup interface application included with FoxPro, the Printer Setup Editing dialog appears so you can create a setup with the name you specified.

When the printer driver setup name you include in <expC> does not exist in your resource file, you can prevent the printer driver setup application from executing by preceding the setup name with a dash (-).

To load a default printer driver setup when you start FoxPro, include the following line in your configuration file.

```
PDSETUP = '<setup name>' WITH <parm list>
```

Be sure to enclose the setup name in quotation marks.

_PECODE

Purpose	Sets ending print codes
Syntax	_PECODE = <expC>
Remarks	Included for backward compatibility — use the Report Writer instead
See Also	? ??, PRINTJOB ... ENDPRINTJOB, _PSCODE

Description _PECODE contains a character value that is output when ENDPRINTJOB is executed. <expC> can contain any collection of print codes up to a maximum of 255 characters. The default _PECODE is the null string. Valid codes range from 0 to 255.

Consult your printer manual for complete information on the effect of specific control code sequences.

PEJECT

Purpose	Sets occurrence of page eject
Syntax	_PEJECT = <expC>
Remarks	Included for backward compatibility — use the Report Writer instead
See Also	EJECT, EJECT PAGE, _PADVANCE, _PECODE, PRINTJOB ... ENDPRINTJOB, _PSCODE

Description _PEJECT contains a character value that determines when FoxPro sends a page eject during output.

<expC> must evaluate to one of the following:

- BEFORE – Sends a page eject before starting output.
- AFTER – Sends a page eject after output is complete.
- BOTH – Sends a page eject before and after print output.
- NONE – Sends no page eject before or after print output.

If <expC> is a literal character string, it must be enclosed in quotation marks.

You can set _PEJECT in the Command window or during program execution, but it only has an effect within the PRINTJOB ... ENDPRINTJOB command structure.

_PEPAGE

Purpose	Returns or sets ending page number
Syntax	_PEPAGE = <expN>
Remarks	Included for backward compatibility — use the Report Writer instead
See Also	_PAGENO, _PBPAGE, PRINTJOB ... ENDPRINTJOB

Description _PEPAGE contains a numeric value that determines the last page to print. When _PAGENO reaches a value that is greater than the value stored in _PEPAGE, FoxPro halts the output of data.

The _PEPAGE value can range from 1 to 32,767, inclusive. The value of _PEPAGE cannot be less than _PBPAGE. The startup default is 32,767. Use _PEPAGE in conjunction with _PBPAGE to print a range of pages. If, for example, you want to print only one page of a long report, you can store the page number in _PBPAGE and _PEPAGE and only the specified page prints.

_PLENGTH

Purpose	Returns or sets length of page
Syntax	_PLENGTH = <expN>
Remarks	Included for backward compatibility — use the Report Writer instead
See Also	EJECT, EJECT PAGE, ON PAGE, _PADVANCE, _PLINENO

Description _PLENGTH contains a numeric value that determines the number of output lines per page. The _PLENGTH value can range from 1 to 32,767, inclusive. The default is 66. Use _PLENGTH in conjunction with _PLINENO to determine when a page eject is needed.

_PLINENO

Purpose	Returns or sets current line number
Syntax	<code>_PLINENO = <expN></code>
Remarks	Included for backward compatibility — use the Report Writer instead
See Also	<code>_PCOLNO</code> , <code>PCOL()</code> , <code>_PLENGTH</code> , <code>PROW()</code>

Description `_PLINENO` contains a numeric value that establishes the current line position. Use `_PLINENO` to determine the next line for output. As FoxPro sends output to the current print device, it increments `_PLINENO` to indicate the current position. `<expN>` can be a value ranging from 0 to `(_PLENGTH - 1)`.

Unlike the `PROW( )` function, which returns the current position of the printhead and is not incremented when `SET PRINTER` is `OFF`, FoxPro increments `_PLINENO` whenever output goes to a printer, the screen or a file. `SET PRINTER` has no effect on `_PLINENO`.

_PLOFFSET

Purpose	Sets page offset
Syntax	_PLOFFSET = <expN>
Remarks	Included for backward compatibility — use the Report Writer instead
See Also	_LMARGIN, SET MARGIN

Description _PLOFFSET contains a numeric value that determines the number of printer columns (the offset) from the left edge of the paper to the left edge of the text in all printed output. Output sent to the screen is not affected by _PLOFFSET. Additional _LMARGIN space is calculated from the _PLOFFSET.

_PLOFFSET can contain a value from 0 to 254. _PLOFFSET is equivalent to SET MARGIN. Adjusting either one automatically changes the other.

The setting of _PLOFFSET does not affect reports created with the Report Writer and run with REPORT FORM. Although _PLOFFSET is adjusted as a report created with the Report Writer runs, it returns to its original value after the report is run. The Printer Indent setting in the Page Layout dialog of the Report Writer determines the offset from the left edge of the paper.

_PPITCH

Purpose	Sets printer pitch
Syntax	_PPITCH = <expC>
Remarks	Included for backward compatibility — use the Report Writer instead
See Also	_PDRIVER

Description The GENPD.APP printer driver application provided with FoxPro for MS-DOS does not use _PPITCH. Any value stored to the _PPITCH system memory value has no effect upon printed output.

_PPITCH is available for your use in a printer driver application that you create.

PQUALITY

Purpose	Sets print quality
Syntax	_PQUALITY = <expL>
Remarks	Included for backward compatibility — use the Report Writer instead
See Also	_PDRIVER

Description The GENPD.APP printer driver application provided with FoxPro for MS-DOS does not use _PQUALITY. Any value stored to the _PQUALITY system memory value has no effect upon printed output.

_PQUALITY is available for your use in a printer driver application that you create.

_PRETEXT

Purpose	Specifies a character expression to preface text merge lines
----------------	--

Syntax	_PRETEXT = <expC>
---------------	--------------------------------

See Also	\ \, SET TEXTMERGE, _TEXT
-----------------	-----------------------------

Description With SET TEXTMERGE you can merge text within a program. You can output lines of text, the contents of memory variables and the results of expressions and functions to the screen, a window, a printer, a text file, or a low-level file.

If you store a character expression to _PRETEXT, the character expression is added to the beginning of the output text line. To facilitate program indentation, you can store a tab or tabs to _PRETEXT.

_PSCODE

Purpose	Sets starting print codes
Syntax	_PSCODE = <expC>
Remarks	Included for backward compatibility — use the Report Writer instead
See Also	? ??, _PECODE, PRINTJOB ... ENDPRINTJOB

Description _PSCODE contains a character value that is output when PRINTJOB executes. <expC> can contain any collection of print codes up to 255 characters; the default is the null string. Valid codes range from 0 to 255.

Consult your printer manual for complete information on the effect of specific control code sequences.

_PSPACING

Purpose	Sets printer line spacing
Syntax	_PSPACING = <expN>
Remarks	Included for backward compatibility — use the Report Writer instead
See Also	? ??, _BOX, DEFINE BOX, _PCODE

Description _PSPACING contains a numeric value that determines if output is single-, double-, or triple-spaced. The default is single-spacing (1). <expN> must be a value from 1 to 3.

You can set _PSPACING from the Command window to alter output from commands such as DISPLAY and LIST. _PSPACING also controls the height of boxes defined with DEFINE BOX. A box defined with a height of 5 will have a height of 15 if _PSPACING is set to 3.

_PWAIT

Purpose	Sets pause between pages of output
Syntax	_PWAIT = <expL>
Remarks	Included for backward compatibility — use the Report Writer instead
See Also	EJECT, EJECT PAGE, _PLENGTH, _PLINENO

Description _PWAIT contains a logical value that determines whether or not the printer pauses between pages of output. _PWAIT allows printing on single sheets of paper, as opposed to continuous-feed (fan-fold) forms. When a page is ejected from the printer with EJECT or when _PLINENO exceeds _PLENGTH, FoxPro suspends output to the printer so you can manually insert another sheet of paper.

_RMARGIN

Purpose	Sets right margin
Syntax	_RMARGIN = <expN>
Remarks	Included for backward compatibility — use the Report Writer instead
See Also	_INDENT, _LMARGIN, _PLOFFSET, SET MARGIN, _WRAP

Description _RMARGIN contains a numeric value that determines the position of the right margin for output generated with the ? command. The minimum _RMARGIN value is the greater of _LMARGIN + 1 or _LMARGIN + _INDENT + 1. The maximum _RMARGIN value is 255. The startup default is 80.

_RMARGIN is active only when _WRAP is set to .T.

STARTUP

Purpose	Specifies a start up application
----------------	----------------------------------

Syntax	_STARTUP = <program name>
---------------	--

Description The system memory variable _STARTUP contains the name of the application that executes when you start FoxPro. By default, this application is FOXSTART.APP, an application supplied with FoxPro that adds the **Run** menu to the main menu bar. If FOXSTART.APP is not in the directory where FoxPro executable files reside, _STARTUP contains the null string.

To execute your own application when FoxPro starts, save a copy of the original FOXSTART.APP under a different name or in a different directory. Then, change the name of your application to FOXSTART.APP.

You can also specify a command or program to execute when FoxPro starts by including one of the following in your configuration file:

```
COMMAND = <command>
```

or

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

A startup application always executes before a command or program specified in your configuration file. For additional information on configuring FoxPro, see the Customizing FoxPro chapter in the *FoxPro Installation and Configuration* manual.

_TABS

Purpose	Specifies tab settings
Syntax	_TABS = <expC>
Remarks	Included for backward compatibility — use the Report Writer instead
See Also	_INDENT, _WRAP

Description _TABS contains a character value that determines where tab stops appear in printed output. <expC> represents a tab list which is a series of comma-delimited numbers in ascending order. By default, _TABS contains the null string, which sets the _TABS spacing at eight character intervals (8,16,24,32,40, and so on).

When _WRAP is set to .T., FoxPro ignores any _TABS values greater than or equal to _RMARGIN. The TABS statement that you include in your configuration file is equivalent to _TABS, and specifying a TABS value automatically sets the value in _TABS.

TALLY

Purpose	Returns number of records processed by the most recently executed table/.DBF command
Syntax	<code>_TALLY = <expN></code>
See Also	SET ODOMETER, SET TALK

Description Certain table/.DBF processing commands return information about their status (“talk”) while they execute. When such a command finishes executing, it displays the number of records it processed (if you have SET TALK ON), and stores this number to the _TALLY system memory variable.

Commands That Return Status Information			
APPEND FROM	COUNT	PACK	SORT
AVERAGE	DELETE	REINDEX	SUM
CALCULATE	INDEX	REPLACE	TOTAL
COPY TO	JOIN	SELECT - SQL	UPDATE

When you start FoxPro, _TALLY is set to zero. Executing one of the commands above replaces the _TALLY value with the number of records the command processed.

Example This example uses SELECT to return the number of customers in California. The result is automatically stored to _TALLY. The _TALLY value is displayed.

```
SELECT * FROM customer ;
  WHERE state = 'CA' ;
  INTO CURSOR temp

? _TALLY
```

_TEXT

Purpose	Directs output from the \, \\ and TEXT ... ENDTEXT text merge commands to a low-level file
Syntax	_TEXT = <expN>
See Also	\\, \\, FOPEN(), FCLOSE(), FCREATE(), _PRETEXT, SET TEXTMERGE, SET TEXTMERGE DELIMITERS

Description \, \\ and TEXT ... ENDTEXT facilitate the merging of text with the contents of table/DBF files, memory variables and the results of functions and expressions. Output generated by these text merge commands can be directed to the screen or a window or to a low-level file. The _TEXT system memory variable lets you direct text merge output from these commands to a low-level file.

To send output from the \ and \\ and TEXT ... ENDTEXT text merge commands to a low-level file, include the TO <file> clause in SET TEXTMERGE. The file handle of <file> is stored to _TEXT. If the file handle of another low-level file was previously stored in _TEXT, that low-level file is closed.

FCREATE() creates and opens low-level files and FOPEN() opens existing files. These functions return a positive file handle if the file is successfully created or opened. Storing this file handle to _TEXT directs any subsequent output from the \, \\ and TEXT ... ENDTEXT text merge commands to the file. Use STORE or = to store a file handle to _TEXT. You must open a low-level file with write privileges in order for it to accept output from the text merge commands.

You can close low-level files with FCLOSE() or CLOSE ALL. You can also use SET TEXTMERGE TO without a file name to close the low-level file whose handle is stored in _TEXT.

The startup default value of _TEXT is -1. If you are directing text merge output to a file whose file handle is stored in _TEXT, you can turn off the output to that file without closing the file by storing -1 to _TEXT. By storing different file handles and -1 to _TEXT, you can direct text merge output to alternate files.

Example The following program demonstrates how to direct text merge output to alternate files:

```
SET TALK OFF
SET TEXTMERGE ON NOSHOW      && Enable text merge, no output to screen
SET TEXTMERGE DELIMITERS TO  && Default text merge delimiters <<, >>

SET TEXTMERGE TO date.txt    && Create and send output to date.txt
STORE _TEXT TO mdatehandl    && Save date.txt's file handle
STORE -1 TO _TEXT            && Output off to date.txt; keep it open

SET TEXTMERGE TO time.txt    && Create and send output to time.txt
STORE _TEXT TO mtimehandl    && Save time.txt's file handle

*** Send the following text to time.txt ***

\The time is:

STORE mdatehandl TO _TEXT    && Now direct output to date.txt

*** Send the following text to date.txt ***

\Today's date is:

STORE mtimehandl TO _TEXT    && Now direct output to time.txt

*** Output the time on the same line ***

\\ <<TIME( )>>
STORE mdatehandl TO _TEXT    && Now direct output to date.txt

*** Output the date on the same line ***

\\ <<DATE( )>>

CLOSE ALL                    && Close all files
TYPE date.txt                && See what's in this file..
WAIT WINDOW                  && Pause
TYPE time.txt                && ..and what's in this file
ERASE date.txt
ERASE time.txt
```

_THROTTLE

Purpose	Specifies the execution speed of programs when the Trace window is open
Syntax	_THROTTLE = <expN>
See Also	SET DEBUG, SET STEP

Description The Trace window, one of the debugging tools available in FoxPro, displays a program's source code as the program executes. The Trace window highlights each line in the program when the line executes. The numeric value of _THROTTLE determines the delay between execution of each program line when the Trace window is open. The **Trace Between Breaks** option on the Trace window's **Program** menu must be on for a delay to occur between program line execution. In FoxPro for MS-DOS, a diamond appears and in FoxPro for Windows, a check mark appears beside the **Trace Between Breaks** option when it is on.

The value of _THROTTLE is the delay in seconds between execution of each program line. The startup default for _THROTTLE is 0, with no pause between execution of program lines. _THROTTLE can range from 0 to 5.5 seconds. For example, if _THROTTLE is set to 0.5 seconds, a 1/2 second delay occurs between execution of each program line.

You can also adjust the program execution delay by selecting the **Throttle...** option from the Trace window's **Program** menu.

_TRANSPORT

Purpose	Specifies the program to transport FoxPro 2.0 screens, labels and reports to FoxPro 2.5 formats
Syntax	<code>_TRANSPORT = <program name></code>
Remarks	New to FoxPro 2.5
See Also	CREATE LABEL, CREATE REPORT, CREATE SCREEN

Description With `_TRANSPORT` you can specify the program FoxPro 2.5 uses when it transports FoxPro 2.0 screens, labels and reports to FoxPro 2.5 formats. By default, `_TRANSPORT` contains `TRANSPRT.PRG`, the screen, label and menu conversion program installed in your FoxPro 2.5 directory.

The program you specify with `_TRANSPORT` runs when you attempt to open a FoxPro 2.0 screen, label or report in FoxPro 2.5. To specify a program other than `TRANSPRT.PRG` to transport your screens, labels and reports, store the program's name in `_TRANSPORT`. If your conversion program is in a directory other than the current default directory, include the path with the program name.

If you rename `TRANSPRT.PRG` or move it to another directory, store the new file name and/or its directory in `_TRANSPORT`.

You can also specify a conversion program in your FoxPro 2.5 configuration file by including the following line:

```
_TRANSPORT = <file name>
```

`<file name>` specifies the name of your conversion program and can include a path.

If you write your own transporter, it must accept one parameter — the name of the report, label or screen form to transport. Your transporter must return one parameter. The parameter can be one of the following values:

Return Code	Meaning
1	Successfully transported.
2	Opened without transporting.
3	Transportation cancelled.

For more information about transporting FoxPro 2.0 screens, labels and reports, refer to the Using Files from Other Platforms chapter in the FoxPro for Windows *Developer's Guide* or the FoxPro for MS-DOS *Update* manual.

_UNIX

Purpose	Contains true (.T.) if you are using FoxPro for UNIX
Syntax	_UNIX = <expL>
Remarks	New to FoxPro 2.5
See Also	_DOS, _MAC, STORE, VERSION(), _WINDOWS

Description The _UNIX system memory variable contains a logical true (.T.) if you are using FoxPro for _UNIX. _UNIX contains a logical false (.F.) if you are using a different FoxPro version (FoxPro for MS-DOS, FoxPro for Macintosh or FoxPro for Windows).

The value contained in _UNIX cannot be changed with STORE or =.

_WINDOWS

Purpose	Contains true (.T.) if you are using FoxPro for Windows
Syntax	_WINDOWS = <expL>
Remarks	New to FoxPro 2.5
See Also	_DOS, _MAC, STORE, _UNIX, VERSION()

Description The _WINDOWS system memory variable contains a logical true (.T.) if you are using FoxPro for Windows. _WINDOWS contains a logical false (.F.) if you are using a different FoxPro version (FoxPro for MS-DOS, FoxPro for Macintosh or FoxPro for UNIX).

The value contained in _WINDOWS cannot be changed with STORE or =.

_WRAP

Purpose	Sets word wrap
Syntax	_WRAP = <expL>
Remarks	Included for backward compatibility — use the Report Writer instead
See Also	_ALIGNMENT, _INDENT, _LMARGIN, SET, _RMARGIN

Description _WRAP contains a logical value that determines whether or not output automatically word wraps. By default, _WRAP is set to .F.

When _WRAP is set to .T., output from the ? command that would extend beyond the right margin (_RMARGIN) automatically wraps to the left margin (_LMARGIN). _WRAP must be set to .T. before _ALIGNMENT, _INDENT, _LMARGIN and _RMARGIN have any effect on output.

Index

Because commands and functions are organized alphabetically in this manual, individual commands and functions are not listed as main entries in this index.

! (exclamation point), 1-13, 3-840
 != (not equal to sign), 1-14
 " " (double quotation marks), 1-10
 # (pound sign), 1-14
 \$ (dollar sign), 1-23, 3-11
 % modulus operator, 3-8
 & (ampersand), 1-16, 3-9
 && (double ampersand), 3-690
 '' (single quotation marks), 1-10
 () (parentheses)
 Functions and, 1-7 - 1-8
 Grouping logical expressions, 1-13
 Grouping numeric expressions, 1-11
 * (asterisk)
 Blinking/bright colors, 3-881
 Comments with, 3-690
 Multiplication operator, 1-11
 + (plus sign), 1-11, 3-881
 , (comma), 1-15
 - (minus sign), 1-11
 ... (ellipsis), 1-9
 / (slash), 1-11
 12/24-hour time format, 3-939
 = (equal sign), 1-14, 3-13
 ? command, 3-16
 ?? command, 3-16
 ??? command, 3-20
 @ ... GET, 3-46, 3-915
 @ ... GET - Check Boxes, 3-48
 @ ... GET - Popups, 3-83
 @ ... GET - Push Buttons, 3-98
 @ ... GET - Radio Buttons, 3-113
 @ ... GET - Spinners, 3-127
 @ ... EDIT, 3-46, 3-135
 @ ... BOX, 3-144
 @ ... CLEAR, 3-145
 @ ... FILL, 3-146
 @ ... TO, 3-152
 \ | \ \ command, 3-14
 ^ caret, 1-11
 { } (braces), 1-12
 | (vertical bar), 1-9

A

Abbreviations

Commands and functions, 1-16
 Key words, 1-8

Aborting execution, 3-262

Absolute values, 3-157

Accessing

Debug/Trace windows, 3-910
 Filtered records, 3-927
 System menu bar, 3-1001

ACOPY(), 3-159

ACOS(), 3-161

ACTIVATE MENU, 3-162

ACTIVATE POPUP, 3-163

ACTIVATE SCREEN, 3-169

ACTIVATE WINDOW, 3-170, 3-427

Activating

Menu bar, 3-162
 Objects with READ, 3-781
 Popups, 3-163, 3-795
 Print job settings, 3-749
 Screens, 3-169
 Windows, 3-170
 Work areas, 3-860

Active window, 3-1136

Adding

Numeric field values, 3-1060
 Records to table from another file, 3-193
 Records to tables from array, 3-198

Addition operator, 1-11

ADEL(), 3-174

ADIR, 3-176

Advancing pages, 3-478, 4-29

AFIELDS(), 3-182

AFONT(), 3-183

AINS(), 3-185

ALEN(), 3-187

Aliases

Default, 1-21
 Defined, 1-16
 Described, 1-21
 FoxPro assigned, 1-22
 Memory variables, 1-23
 Names, 1-14
 Table alias, 1-21
 Table names, 3-356
 Target of relation, 3-1087
 User-assigned, 1-21
 Using, 1-22
 Work areas, 3-189

- Aligning text, 4-3
- `_ALIGNMENT`, 4-3, 4-60
- ALL scope clause, 1-18
- Allocating space for memo field, 3-882
- `ALLTRIM`(), 3-190
- Alphabetic characters
 - `ISALPHA`(), 3-590
 - `ISLOWER`(), 3-593
 - `ISUPPER`(), 3-594
- Amounts, payment, 3-739
- Ampersand (&)
 - Comments with, 3-690
 - Macro substitutions, 1-16, 3-9
- AND logical operator, 1-13, 1-19
- Angle brackets, 1-8
- Angle size, 3-223
- `ANSITOOEM`(), 3-191
- API external routines, 3-945
- `APPEND`, 3-192
- `APPEND FROM`, 3-193, 3-1003, 4-51
- `APPEND FROM ARRAY`, 3-198, 3-318
- `APPEND GENERAL`, 3-202
- `APPEND MEMO`, 3-204
- Appending
 - Records to table, 3-192
 - Records with `INSERT - SQL`, 3-586
- Application startup, 4-49
- Applications, creating, 3-252
- Arc cosine, 3-161
- Arc sine, 3-208
- Arc tangent, 3-217
- Array elements
 - Character expressions and, 1-10
 - Copying, 3-159
 - Date expressions and, 1-12
 - Deleting, 3-174
 - Expressions and, 1-10
 - Logical expressions and, 1-13
 - Name expressions and, 1-16
 - Numeric expressions and, 1-11
- Array manipulation
 - `ACOPY`(), 3-159
 - `ADEL`(), 3-174
 - `ADIR`(), 3-176
 - `AELEMENT`(), 3-180
 - `AFIELDS`(), 3-182
 - `AINS`(), 3-185
 - `ALEN`(), 3-187
 - `APPEND FROM ARRAY`, 3-198
 - `ASCAN`(), 3-206
 - `ASORT`(), 3-210

- `ASUBSCRIPT`(), 3-213
- `COPY TO ARRAY`, 3-317
- `DECLARE`, 3-390
- `DIMENSION`, 3-445
- `SCATTER`, 3-851
- Arrays
 - Adding records to tables, 3-198
 - Clearing, 3-289
 - Copying from records to, 3-317, 3-851
 - Defined, 1-16
 - Defining, 3-751, 3-764
 - Displaying contents, 3-457
 - Inserting columns, 3-185
 - Inserting elements, 3-185
 - Inserting rows, 3-185
 - Memory variable, 3-390, 3-445
 - Memory variable arrays, 3-206
 - Moving fields with `GATHER`, 3-542
 - Names, 1-14
 - Placing file information into, 3-176
 - Placing structure information into, 3-182
 - Regional, 3-805
 - Restoring, 3-824
 - Returning element numbers, 3-180
 - Sorting, 3-210
 - Storing data to, 3-1052
- `ASC`(), 3-205
- `ASCAN`(), 3-206
- ASCII
 - Characters, 3-285
 - Files, 3-1099
 - Values, 3-285
- ASCII code, 3-205
- `ASIN`(), 3-208
- `ASORT`(), 3-210
- Assigning
 - Routines to menu bars, 3-718
 - Routines to menu pads, 3-720
 - Routines to popup options, 3-717
 - Routines to popups, 3-721
- Asterisk (*)
 - Blinking/bright colors, 3-881
 - Comments with, 3-690
 - Multiplication operator, 1-11
- `ASUBSCRIPT`, 3-213
- `AT`, 3-215
- `AT`(), 1-23
- `ATAN`(), 3-217
- `ATC`(), 3-218
- `ATCLINE`, 1-23, 3-219
- `ATLINE`(), 1-23, 3-221

- ATN2(), 3-223
- Attributes
 - BELL, setting, 3-880
 - Screen, setting, 3-881
- AUTOSAVE, 3-879
- AVERAGE, 1-18, 3-224, 3-1003, 4-51
- Averaging numeric expressions, 3-224

B

- Background color, 3-890
- Backspacing printer, 4-32
- Bar pads, 3-403, 3-409
- BAR(), 3-226
- Bars
 - Popup, 3-297, 3-684
 - Status, 3-997
- Bars, menu
 - Activating, 3-699
 - Assigning routines to, 3-718
 - Creating, 3-393, 3-403
 - Described, 3-403
 - Hiding, 3-561
 - Number of, 3-544
 - Placing on stack, 3-767
 - Pulling off stack, 3-744
 - Returning active, 3-647
 - Returning last chosen pad, 3-731
- _BEAUTIFY, 4-4
- Beginning of file, 3-230
- Beginning subroutines, 3-538
- Bell, setting, 3-880
- BETWEEN(), 3-229
- Binary files, loading, 3-612
- Binary routines, 3-814
- Blanks, trailing, 3-839, 3-1096
- Block size, 3-882
- BOF(), 3-230
- Bold type, 1-24
- Border strings, 1-16
- Borders
 - Defining box borders, 3-883
 - Windows with, 3-1127
- _BOX, 4-6
- Boxes
 - Borders for, 3-883
 - Drawing around text, 3-401
 - Drawing with @ ... BOX, 3-144
 - Drawing with @ ... TO, 3-152
- Braces ({}), 1-12
- Brackets, 1-8 - 1-9

- Breakpoints, 3-1010
- BROWSE, 1-18, 3-233, 3-247
- Browse window, 3-885
- Buffers
 - Clearing, 3-292
 - Flushing to disk, 3-879
 - Keyboard, 3-286, 3-599, 3-1011
 - Saving screens/windows, 3-845
- BUILD APP, 3-252
- BUILD EXE, 3-253
- BUILD PROJECT, 3-254
- Buttons
 - Invisible, 3-44, 3-62
 - Mouse, 3-630
 - Push, 3-44
 - Radio, 3-45

C

- _CALCMEM, 4-7
- CALCULATE, 1-18, 3-256, 3-1003
- Calculating field totals, 3-1092
- Calculator desk accessory, 4-8
- _CALCVALUE, 4-8
- Calendar desk accessory, 4-15
- CALL, 3-260
- Calling UDFs, 3-1116
- CANCEL, 3-262
- CapsLock key, status, 3-988
- CapsLock keyboard mode, 3-263
- CAPSLOCK(), 3-263
- Caret (^), 1-11
- Carrying data to new records, 3-886
- CDOW(), 3-264
- CDX(), 3-265
- CEILING(), 3-267
- Center padding, 3-733
- Century, 3-887
- CHANGE, 1-18, 3-269
- Change window, 3-269
- Changes from version 2.0, 3-3
- Changing
 - Decimal point, 3-972
 - Directories, 3-912
 - Drives, 3-912
 - File size, 3-495
 - Popup size, 3-1038
 - Table records, 3-476

Character expressions

- \$ function, 3-11
- Alphabetic, 3-590
- Character type constants, 1-10
- Character type database fields, 1-10
- CHRTRAN(), 3-288
- Comparing, 3-608
- Converting, 3-352, 3-473
- Described, 1-9 - 1-10
- Formatting, 3-1095
- ISLOWER(), 3-593
- ISUPPER(), 3-594
- Leftmost characters, 3-606
- Length, 3-607, 3-1097
- Logical expressions and, 1-13
- Lower-case characters, 3-625
- Numbers, 3-592
- Occurrences, 3-697
- Phonetic differences, 3-444
- Phonetic representations, 3-1048
- Removing leading blanks, 3-626
- Removing trailing blanks, 3-1096
- Repeating, 3-819
- Returning ASCII code, 3-205
- Returning character strings from, 3-1057
- Returning characters, 3-1059
- Returning line numbers, 3-219, 3-221
- Returning position, 3-218
- Returning position with AT(), 3-215
- Searching with RAT(), 3-776
- Searching with RATLINE(), 3-777

Character functions

- \$, 3-11
- CHR(), 3-285
- CHRTRAN(), 3-288
- CTOD(), 3-352
- DIFFERENCE(), 3-444
- DTOC(), 3-473
- EMPTY(), 3-479
- EVALUATE(), 3-485
- INLIST(), 3-582
- INT(), 3-589
- ISALPHA(), 3-590
- ISLOWER(), 3-593
- ISUPPER(), 3-594
- LEFT(), 3-606
- LEN(), 3-607
- LIKE(), 3-608
- LOWER(), 3-625
- LTRIM(), 3-626
- MAX(), 3-628

- MIN(), 3-650
- OCCURS(), 3-697
- Overview, 2-7 - 2-9
- PADC() function, 3-733
- PADL(), 3-733
- PADR(), 3-733
- RAT(), 3-776
- RATLINE(), 3-777
- REPLICATE(), 3-819
- RIGHT(), 3-832
- RTRIM(), 3-839
- SOUNDEX(), 3-1048
- SPACE(), 3-1049
- STR(), 3-1054
- STUFF(), 3-1057
- SUBSTR(), 3-1059
- SYS functions, 3-1065
- TRANSFORM(), 3-1095
- TRIM(), 3-1096
- TYPE(), 3-1099, 3-1101
- UPPER(), 3-1107

Character strings

- Accepting from screen, 3-158
- Comparison, 1-14
- Composed of spaces, 3-1049
- Converting, 3-475
- Described, 1-10
- Padding, 3-733
- Removing trailing blanks, 3-1096
- Replacing with STUFF(), 3-1057
- Returning from numeric expression, 3-1054
- Returning with all blanks, 3-190
- Trimming, 3-839
- Writing to files/ports, 3-528, 3-541

Characters

- Keyboard buffer, 3-1011
- Returning from expressions, 3-1059
- Returning rightmost character, 3-832
- Check boxes, 3-43, 3-48
- Child READs, 3-790
- Child windows, 3-1128
- CHR(), 3-285
- CHRS AW(), 3-286
- CHRTRAN(), 3-288
- Clauses, defined, 1-8
- CLEAR, 3-289
- CLEAR ALL, 3-289

- Clearing
 - Keyboard typeahead buffer, 3-292
 - Printer driver setup, 3-970, 4-35
 - Screens, 3-145, 3-289, 3-888
 - Specified items, 3-289
 - Windows, 3-145, 3-289, 3-292, 3-389
- Clicking, time interval, 4-12
- Clipboard contents, 4-9
- _CLIPTEXT, 4-9
- Clock, 3-889
- CLOSE, 3-293
- CLOSE MEMO, 3-295
- Closing
 - File types, 3-293
 - Format files, 3-294
 - FoxPro session, 3-772
 - Index files, 3-294
 - Memo window, 3-295
 - View window, 3-1015
- CMONTH(), 3-296
- CNTBAR(), 3-297
- CNTPAD(), 3-300
- COL(), 3-301
- Color pairs
 - Color scheme, 3-854
 - Defined, 3-890
 - Lists, 3-891
- Color scheme, 3-892
- Color sets, 3-324
- Colors
 - See also FoxPro User's Guide*
 - Changing with @ ... FILL, 3-146
 - Codes for, 3-890
 - Color set, 3-892
 - Display capability, 3-591
 - Loading color set, 3-896
 - Setting, 3-881
 - Specifying, 3-894
 - User-defined menus/windows, 3-893, 3-896
- Columns
 - Defined, 1-16
 - Deleting from array, 3-174
 - Inserting into arrays, 3-185
 - Numbers, 3-855, 3-1130, 4-32
 - Performing I/O at, 3-21
 - Position of mouse pointer, 3-629
 - Returning cursor column position, 3-301
 - Returning number, 3-187
 - Returning printer position, 3-740
 - Subscript, returning, 3-213
 - Window column location, 3-1137
- Combining optimizable expressions, 1-19
- Comma (,), 1-15
- Command window
 - FoxPro commands, 1-2
 - Placing commands in, 3-920
 - Program file vs., 1-3
- Commands
 - Abbreviations with, 1-16
 - Benefiting from Rushmore, 3-964
 - Defined, 1-7
 - Executing, 3-849
 - FoxPro language, 1-7
 - Help for, 1-5
 - Listed, 3-1
 - New and enhanced, 3-3
 - Placing in Command window, 3-920
 - Table, 2-14
 - Typographic conventions for, 1-7
 - UDFs in, 3-1116
- Commands and functions by category
 - Array manipulation, 2-27
 - Data conversion, 2-13
 - Data types, character functions, 2-7 - 2-9
 - Data types, numeric functions, 2-10
 - Database record manipulation, 2-17
 - Date/time functions, 2-12
 - Debugging and error handling, 2-21
 - Described, 2-1
 - Event handlers, 2-28
 - Field manipulation, 2-14
 - FoxPro environment, 2-42
 - FoxPro file management, 2-47
 - Input/Output, 2-30, 2-34
 - Memory variable manipulation, 2-25
 - Menu bars and popups, 2-35
 - Network, 2-49
 - Program execution, 2-22
 - Programming low-level file functions, 2-24
 - Reports and labels, 2-41
 - Structured programming commands, 2-24
 - Table indexes, 2-19
 - Table manipulation, 2-15
 - Table relations, 2-20
 - Typing/Keyboard/Mouse, 2-39
 - Windows, 2-37
- Comments, 3-690
- Common logarithm, 3-622

- Communication ports
 - Closing, 3-496
 - Opening, 3-521
 - Writing character strings to, 3-528, 3-541
- Communications ports
 - Returning number of bytes, 3-531
 - Returning series of bytes, 3-504
- Comparing character expressions, 3-608
- Comparisons
 - Exact matches, 3-923
 - Specifying SQL string comparisons, 3-877
- Compatibility with FoxBASE+, 3-898
- COMPILE, 3-302
- Compiling
 - Program files, 3-302
 - Saving errors to file, 3-949
 - Time constants, 3-391
- Compound index files
 - COPY INDEXES and, 3-306
 - Copying tags, 3-311
 - Deleting tags, 3-441
 - Returning names of, 3-265, 3-631
- Computing
 - Field totals, 3-1092
- COMSPEC, 3-547
- Conditional statements
 - IF ... ENDIF and, 3-565
 - IIF() and, 3-568
- Configuration file, 4-49
- Constants
 - Date, 1-12
 - Numeric, 1-11
 - Optimizable expressions, 1-19
- Context-sensitive help, 1-5
- CONTINUE, 3-304, 3-526
- Control statements (EXIT), 3-486
- Controls
 - Described, 3-43
 - Disabled, 3-46
 - Setting keystrokes for, 3-943
 - Syntax summary, 3-46
 - Typographic conventions, 1-24
- CONVERT.PRG, 4-55
- Converting
 - Character expressions, 3-352
 - Date expressions, 3-463, 3-473, 3-475, 3-633
 - Degrees to radians, 3-474
 - Radians to degrees, 3-838
- Copies, number to print, 4-33
- COPY commands, 3-305
- COPY FILE, 3-305
- COPY INDEXES, 3-306
- COPY MEMO, 3-307
- COPY STRUCTURE, 3-308
- COPY STRUCTURE EXTENDED, 3-309, 3-325
- COPY TAG, 3-311
- COPY TO, 1-18, 3-312, 3-314, 3-1003
- COPY TO ARRAY, 1-18, 3-317
- Copying
 - Array elements, 3-159
 - Commands in Command window, 1-2 - 1-3
 - Compound index file tags, 3-311
 - Data to different formats, 3-488
 - Files with COPY FILE, 3-305
 - From records to array, 3-317
 - From table to file, 3-312
 - Index files, 3-306
 - Memo field contents, 3-307
 - Table structure to records, 3-309
 - Table structures, 3-308
- COS(), 3-320
- Cosine, 3-161, 3-223, 3-320
- COUNT, 1-18, 3-321, 3-1003
- Counting
 - Fields, 3-497
 - Records, 3-321, 3-801
- Courier font, 1-24
- CREATE, 3-323
- CREATE COLOR SET, 3-324
- CREATE FROM, 3-325
- CREATE LABEL, 3-329
- CREATE MENU, 3-332
- CREATE PROJECT, 3-334
- CREATE QUERY, 3-337
- CREATE REPORT, 3-339
- CREATE REPORT - Quick Report, 3-342
- CREATE SCREEN, 3-344
- CREATE SCREEN - Quick Screen, 3-347
- CREATE TABLE, 3-349
- CREATE TABLE (SQL), 3-349
- CREATE VIEW, 3-351
- Creating
 - Applications, 3-252
 - Check boxes, 3-48
 - Color sets, 3-324
 - Compile time constants, 3-391
 - Expressions, 3-548
 - Files, 3-498
 - Invisible buttons, 3-62
 - Labels, 3-329, 3-601

Creating (continued)

- Lists, 3-71
 - Memory variable arrays, 3-390, 3-445
 - Menu bar pads, 3-409
 - Menu bars, 3-403
 - Menus, 3-332, 3-643
 - One-to-many relationships, 3-991
 - Popup menu options, 3-393
 - Popups, 3-83, 3-416
 - Programs, 1-3 - 1-4
 - Projects, 3-254, 3-334
 - Push buttons, 3-98
 - Queries, 3-337
 - Radio buttons, 3-113
 - Reports, 3-339, 3-342
 - Screens, 3-344, 3-347
 - Spinners, 3-127
 - Stand-alone executables, 3-253
 - Tables, 3-323, 3-325, 3-349, 3-569
 - Text editing regions, 3-135
 - UDFs, 3-1114
 - View files, 3-351
 - Windows, 3-427
- Creation dates/times, 3-916
- Criteria, matching, 3-304
- CTOD(), 3-352
- CURDIR(), 3-353
- _CUROBJ, 4-11
- Currency, 3-906
- Current
- Column numbers, 4-32
 - Cursor row position, 3-837
 - Fields, 3-1121
 - Line number, 4-40
 - Page number, 4-30
 - Record number, 3-802
 - Time, 3-1091
 - Work areas, 3-861
- Cursor
- Column position, 3-301
 - Row position, 3-837
 - Turning on/off, 3-908

D

Data conversion

- CHR(), 3-285
- CHRTRAN(), 3-288
- CTOD(), 3-352
- Data type categories, 2-13
- LOWER(), 3-625
- PROPER(), 3-762
- STRTRAN(), 3-1055
- SYS functions, 3-1065
- TRANSFORM(), 3-1095
- UPPER(), 3-1107

Data type, 2-13

Data type categories

- Character functions, 2-7 - 2-9
- Data conversion functions, 2-13
- Date/time functions, 2-12
- Numeric functions, 2-10

Data types

- Expressions and, 1-10
- Returning with TYPE(), 3-1101

Data Types - Character Functions

- ALLTRIM(), 3-190
- ASC(), 3-205
- AT(), 3-215
- ATC(), 3-218
- ATCLINE(), 3-219
- ATLINE(), 3-221

Data Types - Numeric Functions

- ABS(), 3-157
- ACOS(), 3-161
- ASIN(), 3-208
- ATAN(), 3-217
- ATN2(), 3-223

Data, carrying to new records, 3-886

Databases

See Tables

Date and time functions

- BETWEEN(), 3-229
- CDOW(), 3-264
- CMONTH(), 3-296
- CTOD(), 3-352
- _DIARYDATE, 4-15
- DATE(), 3-354
- DAY(), 3-355
- DMY(), 3-463
- DOW(), 3-472
- DTOC(), 3-473
- DTOS(), 3-475
- EMPTY(), 3-479

Date and time functions (continued)

GOMONTH(), 3-557

INLIST(), 3-582

MAX(), 3-628

MDY(), 3-633

MIN(), 3-650

MONTH(), 3-679

SECONDS(), 3-857

SET HOURS, 3-939

SET MARK TO, 3-955

TIME(), 3-1091

YEAR(), 3-1150

Date constants, 1-12

Date expressions

CDOW (), 3-264

Converting, 3-352, 3-463, 3-473, 3-475

Day of month, 3-355

Described, 1-9, 1-12

DOW() and, 3-472

Logical expressions, 1-13

Month, 3-296

MONTH(), 3-679

Year, 3-1150

Date formats, 3-633

DATE(), 3-354

Dates

Current system, 3-354

Delimiters in, 3-955

Format, 3-463, 3-909

GOMONTH, 3-557

Last updated, 3-627

Setting year format, 3-887

Day of month, 3-355

Day of week, 3-264, 3-472

DAY(), 3-355

DBF(), 3-356

_DBLCLICK, 4-12

DDEAbortTrans(), 3-357

DDEAdvise(), 3-358

DDEEnabled(), 3-361

DDEExecute(), 3-363

DDEInitiate(), 3-366

DDELastError(), 3-368

DDEPoke(), 3-369

DDERequest(), 3-371

DDESetOption(), 3-374

DDESetService(), 3-375

DDESetTopic(), 3-379

DDETerminate(), 3-382

DEACTIVATE MENU, 3-383

DEACTIVATE POPUP, 3-386

DEACTIVATE WINDOW, 3-389

Deactivating

Menus, 3-383

Popups, 3-386

Windows, 3-389

Debug window, 3-910

Debugging and error handling

ERROR(), 3-484

EXTERNAL, 3-491

LINENO(), 3-610

NOTE | * | &&, 3-690

ON ERROR, 3-702

ON READERROR, 3-715

ON(), 3-724

Opening Trace window, 3-999

PROGRAM, 3-758

RETRY, 3-830

SET DEBUG, 3-910

SET DEVELOPMENT, 3-916

SET DOHISTORY, 3-920

SET ECHO, 3-921

SET TALK, 3-1003

SET TRBETWEEN, 3-1010

SYS functions, 3-1065

Trapping for errors, 3-484, 3-702

VARREAD(), 3-1121

WAIT, 3-1124

Decimals

Changing point character, 3-972

Fixing number of decimal places, 3-928

Setting number of places, 3-911

DECLARE, 3-390, 3-445

#DEFINE ... #UNDEF, 3-391

DEFINE BAR, 3-393

DEFINE BAR/PAD commands, 3-393, 3-409

DEFINE BOX, 3-401, 4-6

DEFINE MENU, 3-403

DEFINE PAD, 3-409

DEFINE POPUP, 3-416

DEFINE WINDOW, 3-427

Defining

Boxes around text, 3-401

Function keys, 3-931

Menu bar pads, 3-409

Menu bars, 3-403

Popup menu options, 3-393

Popups, 3-416

Subroutine start, 3-538

Windows, 3-427

Definitions, window, 3-847

Degrees, converting, 3-474, 3-838

- DELETE, 1-18, 3-438, 3-1003, 4-51
- DELETE FILE, 3-440
- DELETE TAG, 3-441
- DELETED(), 3-442
- Deleting
 - Columns from arrays, 3-174
 - Commands in Command window, 1-2
 - Compound index file tags, 3-441
 - Elements from arrays, 3-174
 - Files, 3-440
 - Marked records, 3-729, 3-799
 - Records, 3-438, 3-442, 3-914
 - Rows from arrays, 3-174
- Delimiters
 - Dates, 3-955
 - Defined, 1-16
 - Setting, 3-915, 3-1008
- Designing menus, 3-332, 3-665
- Desk accessories, 3-508
- Dialogs
 - Displaying, 3-769
 - Typographic conventions for, 1-24
- Diary desk accessory, 4-15
- _DIARYDATE, 4-15
- DIFFERENCE(), 3-444
- DIMENSION, 3-445
- DIR, 3-449
- Directories
 - Changing, 3-912
 - Displaying, 3-449
 - DOS, 3-353
 - Searching, 3-507
 - Setting, 3-969
- DIRECTORY, 3-449
- Disabled controls, 3-46
- Disabling/enabling
 - Automatic record/file locking, 3-947
 - Browse window status bar, 3-885
 - Debug/Trace windows, 3-910
 - Escape key, 3-922
 - Field evaluation, 3-1005
 - Field heading display, 3-932
 - File protection facility, 3-987
 - Help facility, 3-933
 - Menus, 3-992
 - Multiple record locking, 3-960
 - Pathname display, 3-930
 - Popup bars, 3-1042
 - Printer output, 3-973
 - Rushmore, 3-964
 - Status bar display, 3-996
 - System message display, 3-962
 - Tracing between breakpoints, 3-1010
- Disk space available, 3-451, 3-882
- DISKSPACE(), 3-451
- DISPLAY, 1-18, 3-452, 3-611, 4-46
- Display elements, 3-1063
- DISPLAY FILES, 3-455
- DISPLAY MEMORY, 3-457
- Display modes, 3-918
- DISPLAY STATUS, 3-459
- DISPLAY STRUCTURE, 3-461
- Displaying
 - ASCII file contents, 3-1099
 - Boxes, 4-6
 - Browse window status bar, 3-885
 - Clock, 3-889
 - Disk directory listing, 3-449
 - Expression results, 3-16
 - Expressions, 3-452, 3-611
 - Field headings, 3-932
 - File information, 3-455
 - Font dialog, 3-553
 - FoxPro environment status, 3-459
 - GET objects, 3-1023, 3-1028
 - Help topics, 3-935
 - Keyboard Macros dialog, 3-950
 - Memory variable/array contents, 3-457
 - Menu bar, 3-162, 3-1027
 - Messages under status bar, 3-957
 - Open dialog, 3-550
 - Pathnames, 3-930
 - Popups, 3-163, 3-1031
 - Record changes, 3-977
 - Records, 3-452, 3-611
 - Save As dialog, 3-769
 - Single GET object, 3-1020
 - States of keys, 3-988
 - Status bar, 3-996
 - System menus, 3-1000
 - Table structure, 3-461
 - Windows, 3-170, 3-1033
 - Windows-style status bar, 3-997
- Division, 1-11, 3-653
- DMY, 3-463
- DO WHILE loops, 3-486
- DO, 1-4, 3-464
- Documenting FoxPro programs, 4-17
- Dollar sign (\$), 3-11
- DOS
 - Directory, 3-353
 - Environment variables, 3-547

- _DOS, 4-16
- Double quotation marks (" "), 1-10
- Double-clicking, 4-12
- Double-spacing, 4-46
- DOW(), 3-472
- Drawing
 - Boxes, 3-401
 - Boxes with @ ... BOX, 3-144
 - Boxes with @ ... TO, 3-152
- Drive
 - Defined, 1-16
- Driver, printer
 - _GENPD, 4-21
 - Loading/clearing setup, 3-970
 - _PPITCH and, 4-42
 - _PQUALITY, 4-43
 - Setup, 4-35
 - Specifying, 4-34
- Drives, changing, 3-912
- DTOC(), 3-473
- DTOR(), 3-474
- DTOS(), 3-475
- Duplicate index keys, 3-1014

E

- ECHO, 3-921
- EDIT, 1-18, 3-269, 3-476
- Editing
 - Commands, 1-2
 - Creating text edit regions, 3-135
 - Highlighting input fields, 3-942
 - Memo fields, 3-662, 3-1016
 - Opening text editing window, 3-654
 - Program files, 1-4
 - Records, 3-476
 - Tables, 3-269
 - Text editing regions, 3-45
 - Window, opening, 3-657
- EJECT, 3-477
- EJECT PAGE, 3-478
- Ejecting pages, 3-477 - 3-478, 4-37
- Elements, array
 - Deleting, 3-174
 - Inserting into arrays, 3-185
 - Returning element numbers, 3-180
 - Returning number, 3-187
 - Returning subscript, 3-213
- Ellipsis (...), 1-9
- Embedded objects, editing, 3-657
- EMPTY(), 3-479

- Enabling/disabling
 - Automatic record/file locking, 3-947
 - Browse window status bar, 3-885
 - Debug/Trace windows, 3-910
 - Escape key, 3-922
 - Field evaluation, 3-1005
 - Field heading display, 3-932
 - File protection facility, 3-987
 - Help facility, 3-933
 - Menu pads, 3-1043
 - Menus, 3-992
 - Multiple record locking, 3-960
 - Pathname display, 3-930
 - Printer output, 3-973
 - Rushmore, 3-964
 - Status bar display, 3-996
 - System message display, 3-962
 - Tracing between breakpoints, 3-1010
- End of file, 3-480, 3-500
- #ENDIF, 3-565
- Ending
 - Page number, 4-38
 - Print codes, 4-36
- ENDPRINTJOB, 3-749
- ENDSCAN, 3-849
- ENDTEXT, 3-1089
- Environment, DOS, 3-547
- EOF(), 3-480
- Equal sign (=), 1-14, 3-13
- ERASE, 3-482
- Error handling, 3-484, 3-702, 3-910
- ERROR(), 3-484
- Errors
 - Function execution, 3-501
 - Input, 3-715
 - PROGRAM(), 3-758
 - Saving to file, 3-949
 - Trapping, 3-702
- Escape key
 - Enabling/disabling, 3-922
 - Trapping for, 3-703
- EVALUATE(), 3-485
- Evaluating
 - Expressions with = command, 3-13
 - Expressions with ? | ?? command, 3-16
 - Expressions with EVALUATE, 3-485
 - Fields with SET TEXTMERGE, 3-1005

- Event handlers
 - ON ERROR, 3-702
 - ON ESCAPE, 3-703
 - ON KEY, 3-704
 - ON KEY =, 3-705
 - ON KEY LABEL, 3-708
 - ON PAD, 3-711
 - ON SELECTION PAD, 3-720
 - ON SELECTION POPUP, 3-721
 - ON(), 3-724
 - POP KEY, 3-743
 - PUSH KEY, 3-766
- Exact matches, 3-923
- Exclamation point (!)
 - As RUN command, 3-840
 - Logical negative operator, 1-13
- Exclusive use, opening tables, 3-925
- Executables, creating stand-alone, 3-253
- Executing commands
 - in Command window, 1-2
 - Logical condition, 3-468
 - Moving through tables, 3-849
 - Program file, 1-3, 3-840
 - RETRY, 3-830
- Executing looped statements, 3-524
- Executing low-level functions, 3-501
- Executing program files
 - DO command, 3-464
 - SET ECHO, 3-921
- Executing programs, aborting, 3-262
- Execution
 - Current program, 3-758
 - Resuming, 3-829
 - Speed, 4-54
 - Suspending, 3-1062
- Existing windows, 3-1132
- Exit, 3-486
 - Change window, 3-269
 - Fields, 3-904
 - FoxPro session, 3-772
 - Loops, 3-486
- EXP(), 3-487
- Exponentiation, 1-11
- Exponents, 3-487
- EXPORT, 1-18, 3-488
- Exporting data, 3-488
- Expressions
 - Angle brackets, 1-8
 - Between other expressions, 3-229
 - Blank, 3-479
 - Calculating in, 3-256

- Character, 1-10, 1-13, 3-11, 3-1096
- Converting character to date, 3-352
- Data type, 1-10, 3-1101
- Date, 1-12 - 1-13, 3-264
- Described, 1-9
- Displaying, 3-452, 3-611
- Evaluating, 3-13, 3-16, 3-485
- Examples, 1-7
- Filter, 3-510
- Formatting, 3-1095
- Grouping, 1-11
- Logical, 1-13, 1-20
- Name, 1-16
- Null string, 1-10
- Numbers, 1-10
- Numeric, 1-11, 1-13, 3-157
- Relational operators, 1-13
- Searching arrays, 3-206
- Searching with INLIST(), 3-582
- Space between, 3-995
- Types, 1-9
- EXTERNAL, 3-491

F

- False, logical expression value, 1-13, 1-20
- FCHSIZE(), 3-495
- FCLOSE(), 3-496
- Fcode, 1-16
- FCOUNT(), 3-497
- FCREATE(), 3-498, 4-52
- FEOF(), 3-500
- FERROR(), 3-501
- FFLUSH(), 3-503
- FGETS(), 3-504
- FIELD(), 3-506
- Fields
 - Accessing, 3-926
 - Counting, 3-497
 - Current, 3-1121
 - Delimiting, 3-915
 - Displaying headings, 3-932
 - Evaluating, 3-1005
 - Exiting, 3-904
 - Highlighting input, 3-942
 - Inserting/removing space between, 3-995
 - Manipulation commands, 2-14
 - Names, 1-14
 - Names, returning with FIELD(), 3-506
 - Sizes, 3-534

- Fields, table
 - Averaging, 3-224
- File extensions
 - .FLL, 3-945
 - .PLB, 3-945
 - .VUE, 3-351
 - .WIN, 3-847
- File formats
 - Copying data to different, 3-488
 - Importing, 3-569
- File maintenance desk accessory, 3-508
- File management
 - CLOSE ALL, 3-293
 - CLOSE ALTERNATE, 3-293
 - CLOSE DATABASES, 3-293
 - CLOSE FORMAT, 3-294
 - CLOSE INDEX, 3-294
 - CLOSE PROCEDURE, 3-294
 - COPY FILE, 3-305
 - CREATE VIEW, 3-351
 - CURDIR(), 3-353
 - DELETE FILE, 3-440
 - DIR, 3-449
 - DIRECTORY, 3-449
 - DISKSPACE(), 3-451
 - DISPLAY FILES, 3-455
 - ERASE, 3-482
 - FILE(), 3-507
 - FILER, 3-508
 - FLUSH, 3-517
 - FoxPro environment, 2-47
 - FULLPATH(), 3-535
 - GETFILE(), 3-550
 - LOCFILE(), 3-616
 - MODIFY COMMAND, 3-654
 - MODIFY FILE, 3-654
 - PUTFILE(), 3-769
 - RENAME, 3-815
 - SET ALTERNATE, 3-876
 - SET DEFAULT, 3-912
 - SET FULLPATH, 3-930
 - SET PATH, 3-969
 - SET SAFETY, 3-987
 - SET VIEW, 3-1015
 - SYS functions, 3-1065
 - TYPE, 3-1099
- File menu, 1-3 - 1-4
- File names, 1-8, 1-14
- File pointers, 3-532
- File types, closing, 3-293
- FILE(), 3-507
- FILER, 3-508
- Files
 - Automatic locking, 3-947
 - Beginning of, 3-230
 - Changing size, 3-495
 - Closing, 3-496
 - Copying, 3-305, 3-312
 - Creating, 3-498
 - Deleting, 3-440
 - Directing output to, 3-876
 - Displaying information, 3-455
 - End, 3-500
 - Erasing, 3-482
 - Exclusive use, 3-925
 - Finding on disk, 3-507
 - Flushing to disk, 3-503
 - Format, 3-929
 - Index, 3-940, 3-965
 - Locating on disk, 3-616
 - Locking, 3-514, 3-983
 - Number of bytes, 3-531
 - Opening, 3-521, 3-1108
 - Output to, 3-917
 - Program creation date/time, 3-916
 - Protecting, 3-987
 - Renaming, 3-815
 - Resource, 3-986
 - Saving compilation errors, 3-949
 - Saving keyboard macros, 3-844
 - Saving memory variables, 3-846
 - Saving window definitions, 3-847
 - Series of bytes, 3-504
 - Sorting, 3-1044
 - Unlocking with UNLOCK, 3-1102
 - Writing character strings, 3-528, 3-541
- Filling memo fields from files, 3-204
- FILTER(), 3-510
- Filtering help topics, 3-935
- Filters, 3-510, 3-927
- Financial functions
 - FV(), 3-540
 - PAYMENT(), 3-739
 - PV(), 3-771
- Financial operations, 3-256
- FIND, 3-511, 3-526
- Finding
 - See also* Searching
 - Files on disk, 3-507, 3-616
 - Next record, 3-304
 - Records, 3-511, 3-526, 3-614
- Fixed decimal places, 3-928

- FKLABEL(), 3-512
- FKMAX(), 3-513
- FLOCK(), 3-514
- FLOOR(), 3-516
- FLUSH, 3-517
- Flushing
 - Buffers to disk, 3-879
 - Files/ports, 3-496
 - Open file to disk, 3-503
 - Table/index changes, 3-517
- FONTMETRIC(), 3-518
- Fonts, 3-518, 3-553, 3-1133
- FOPEN(), 3-521, 4-52
- FOR ... ENDFOR, 3-524
- FOR clause, 1-18
- FOR loops, 3-486
- Foreground color, 3-890
- Form feed, 4-29
- Format files, closing, 3-294
- Formats
 - Copying data to different, 3-488
 - Date, 3-633, 3-909
 - File, importing, 3-569
 - Format files, 3-929
 - Time, 3-939
 - Year, 3-887
- Formatting expressions, 3-1095
- FOUND(), 3-526
- Foundation READs, 3-788
- FoxBASE+ compatibility, 3-898
- _FOXDOC, 4-17
- FoxGraph, 4-18 - 4-19, 4-24
- FOXHELP.DBF, 3-935
- FoxPro
 - Converting formats, 4-55
 - User interface, 1-2
 - Version number, 3-1123
- FoxPro-assigned aliases, 1-22
- FoxPro Distribution Kit, 3-253
- FoxPro environment
 - ADIR(), 3-176
 - CAPSLOCK(), 3-263
 - COL(), 3-301
 - Commands and functions, 2-42
 - CREATE COLOR SET, 3-324
 - CREATE VIEW, 3-351
 - DIR, 3-449
 - DIRECTORY, 3-449
 - DISKSPACE(), 3-451
 - DISPLAY, 3-452
 - Displaying status, 3-459
 - Environment commands and functions, 2-42
 - File management commands and functions, 2-47
 - FKLABEL(), 3-512
 - FKMAX(), 3-513
 - FONTMETRIC(), 3-518
 - GETENV(), 3-547
 - GETFILE(), 3-550
 - GETFONT(), 3-553
 - HELP, 3-559
 - INKEY(), 3-577
 - INSMODE(), 3-588
 - ISCOLOR(), 3-591
 - LOCFILE(), 3-616
 - MEMORY(), 3-636
 - Network commands and functions, 2-49
 - NUMLOCK(), 3-691
 - OS(), 3-728
 - PCOL(), 3-740
 - PROW(), 3-763
 - PUTFILE(), 3-769
 - QUIT, 3-772
 - ROW(), 3-837
 - SCHEME(), 3-854
 - SET, 3-875
 - SET ANSI, 3-877
 - SET BELL, 3-880
 - SET BLOCKSIZE, 3-882
 - SET BRSTATUS, 3-885
 - SET CENTURY, 3-887
 - SET CLEAR, 3-888
 - SET CLOCK, 3-889
 - SET COLOR, 3-890 - 3-892
 - SET COLOR OF, 3-893
 - SET COLOR OF SCHEME, 3-894
 - SET COLOR SET, 3-896
 - SET COLOR TO, 3-896
 - SET COMPATIBLE, 3-898
 - SET CURRENCY, 3-906
 - SET DATE, 3-909
 - SET DEBUG, 3-910
 - SET DEFAULT, 3-912
 - SET DELIMITERS, 3-915
 - SET DISPLAY, 3-918
 - SET FIXED, 3-928
 - SET FUNCTION, 3-931
 - SET HELP, 3-933
 - SET HELPFILTER, 3-935
 - SET INTENSITY, 3-942
 - SET KEYCOMP, 3-943

FoxPro environment (continued)

- SET LIBRARY, 3-945
- SET MARK OF, 3-952
- SET MARK TO, 3-955
- SET MOUSE, 3-959
- SET NOTIFY, 3-962
- SET PATH, 3-969
- SET POINT, 3-972
- SET RESOURCE, 3-986
- SET SCOREBOARD, 3-988
- SET SEPARATOR, 3-989
- SET STATUS, 3-996
- SET STATUS BAR, 3-997
- SET STICKY, 3-1000
- SET TALK, 3-1003
- SET TEXTMERGE, 3-1005
- SET TEXTMERGE
 - DELIMITERS, 3-1008
- SET TOPIC, 3-1009
- SET TYPEAHEAD, 3-1011
- SET VIEW, 3-1015
- SET WINDOW OF MEMO, 3-1016
- SET(), 3-1017
- SYS functions, 3-1065
- SYSMETRIC(), 3-1063
- TXTMETRIC(), 3-1097
- VERSION(), 3-1123
- WFONT(), 3-1133

FoxPro language

- Commands and functions, 1-7
- Described, 1-1
- Understanding, 1-7
- Using, 1-1

FoxPro system information, 3-1065

- FOXSWAP, 3-840
- FOXUSER.DBF, 4-35
- FPUTS(), 3-528
- FREAD(), 3-531
- FSEEK(), 3-532
- FSIZE(), 3-534
- FULLPATH(), 3-535
- FUNCTION, 3-538

Function keys

- Defining, 3-931
- Number of, 3-513
- Returning, 3-512

Functions

- Abbreviations with, 1-16
- Array manipulation, 2-27
- Character expressions and, 1-10
- Date expressions and, 1-12

- Defined, 1-7

- Event handlers, 2-28

- Expressions and, 1-10

- FoxPro environment, 2-42

- FoxPro file management, 2-47

- FoxPro language, 1-7

- Help for, 1-5

- Input/Output, 2-30

- Listed, 3-1

- Logical expressions and, 1-13

- Memo fields and, 1-23

- Memory variable manipulation, 2-25

- Menu bars and popups, 2-35

- Network, 2-49

- New and enhanced, 3-3

- Numeric expressions and, 1-11

- Parentheses, 1-7 - 1-8

- Printing, 2-34

- Reports and labels, 2-41

- Structured programming, 2-24

- Table, 2-14

- Typing/keyboard/mouse, 2-39

- Typographic conventions, 1-7

- Windows, 2-37

- Future values of investments, 3-540

- FV(), 3-540

- FWRITE(), 3-541

G

- GATHER, 3-542

Generating

- Menus, 4-20

- Screens, 4-23

- _GENGRAPH, 4-19

- _GENMENU, 4-20

- GENMENU.PRG, 4-20

- _GENPD, 4-21

- GENPD.APP, 4-21, 4-42

- _GENSCRN, 4-23

- genscrn.prg, 4-23

- _GENXTAB, 4-24

- GENXTAB.PRG, 4-24

- GET fields, 3-692

- GET object number, 4-11

- GET object, redisplaying, 3-1020

- GET-less READs, 3-788

- GETBAR(), 3-544

- GETDIR(), 3-546

- GETENV(), 3-547

- GETEXPR, 3-548

- GETFILE(), 3-550
- GETFONT(), 3-553
- GETPAD(), 3-554
- Global memory variables/arrays, 3-764
- GO/GOTO, 3-555
- GOMONTH(), 3-557
- Greater than, 1-14, 3-267
- Grouping
 - Logical expressions, 1-13
 - Numeric expressions, 1-11

H

- HAVING clause (SELECT-SQL), 3-872
- HEADER(), 3-558
- Headers, table, 3-558
- Headings, fields, 3-932
- HELP, 3-559
- Help facility
 - Displaying topics, 3-935
 - Enabling/disabling, 3-933
 - File structure, 3-935
 - Specifying topic, 3-1009
- Help Window, 3-559
- Help, context-sensitive, 1-5
- Help, online, 1-5, 3-1
- Help... option (System menu), 1-5
- HIDE MENU, 3-561
- HIDE POPUP, 3-562
- HIDE WINDOW, 3-563
- Hiding
 - Menu bars, 3-561
 - Popups, 3-562
 - Windows, 3-563
- Highlighting input fields, 3-942
- History, displaying, 3-920
- Hot keys, defining, 3-46
- Hours, setting format, 3-939
- Hyphen (-), 1-11

I

- #IF ... #ENDIF, 3-565
- IIF(), 3-568
- Images, screen/window, 3-845
- Immediate If, IIF(), 3-568
- IMPORT, 3-569
- Importable data file types, 3-569
- Importing data from other formats, 3-569
- _INDENT, 4-25, 4-60

- Indenting first line, 4-25

INDEX

- Directing status to screen/window, 3-1003
- FOR clause with, 1-18
- Returning status, 4-51
- Index files
 - Closing, 3-294
 - Compound, 3-311, 3-441, 3-631
 - Copying, 3-306
 - Open, 3-689
 - Rebuilding, 3-807
 - Reindexing, 3-807
 - Returning key expressions, 3-597
 - Returning master name, 3-726
 - Returning names, 3-265
- Index keys, duplicate, 3-1014
- Indexes
 - Commands and functions for, 2-19
 - Creating with INDEX, 3-571
 - Index file names, 1-14
 - Master index file, 3-965
 - Opening index files, 3-940
 - Optimizable expressions and, 1-19
 - Returning tag names, 3-1084
 - Saving changes to disk, 3-517
 - Searching, 3-858
 - Seeking indexed tables, 3-858
- Initial capital letters, 1-24
- Initializing
 - Calculator value, 4-8
 - Calendar/Diary date, 4-15
 - Clipboard contents, 4-9
- INKEY(), 3-577
- INLIST(), 3-582
- INPUT, 3-584
 - Errors, 3-715
 - Keyboard, waiting for, 3-1124
 - To memory variables, 3-584
- Input fields, highlighting, 3-942
- INSERT, 3-585
- INSERT (SQL), 3-586
- Insert key, 3-988
- Insert mode, 3-588
- Inserting
 - Columns into arrays, 3-185
 - Commands in Command window, 1-2
 - Elements into arrays, 3-185
 - New records, 3-585
 - Rows into arrays, 3-185
 - Space, 3-995
- INSMODE(), 3-588

- INT(), 3-589
- Integer, 3-589
- Intensity, 3-942
- Interest loans, 3-739
- Intervals, reporting, 3-963
- Investments, 3-771
- Invisible buttons, 3-44, 3-62
- I/O - commands and functions
 - Described, 2-30
 - Menu bars and popups, 2-35
 - Printing, 2-34
 - Reports and labels, 2-41
 - Typing/keyboard/mouse, 2-39
 - Windows, 2-37
- I/O - Data Formatting
 - ? | ??, 3-16
 - @ ... GET - Check Boxes, 3-48
 - @ ... GET - Invisible Buttons, 3-62
 - @ ... GET - Lists, 3-71
 - @ ... GET - Popups, 3-83
 - @ ... GET - Push Buttons, 3-98
 - @ ... GET - Radio Buttons, 3-113
 - @ ... GET - Spinners, 3-127
 - @ ... EDIT, 3-135
 - @ ... BOX, 3-144
 - @ ... CLEAR, 3-145, 3-289
 - @ ... FILL, 3-146
 - ACTIVATE SCREEN, 3-169
 - APPEND, 3-192
 - BROWSE, 3-233
 - CHANGE, 3-269
 - CLEAR READ, 3-291
 - CLOSE FORMAT, 3-294
 - COL(), 3-301
 - CREATE SCREEN, 3-344, 3-347
 - DISPLAY, 3-452
 - INKEY(), 3-577
 - INPUT, 3-584
 - LIST, 3-611
 - MODIFY SCREEN, 3-673
 - OBJNUM(), 3-692
 - RDLEVEL(), 3-779
 - READ, 3-781
 - RESTORE, 3-827
 - SAVE SCREEN, 3-845
 - SCOLS(), 3-855
 - SCROLL, 3-856
 - SET BLINK, 3-881
 - SET CARRY, 3-886
 - SET CLEAR, 3-888
 - SET COLOR commands, 3-890 - 3-892

- SET COLOR OF, 3-893
- SET COLOR SET, 3-896
- SET CONFIRM, 3-904
- SET CONSOLE, 3-905
- SET CURSOR, 3-908
- SET DECIMALS, 3-911
- SET DELIMITERS, 3-915
- SET DEVICE, 3-917
- SET FIXED, 3-928
- SET FORMAT, 3-929
- SET HEADING, 3-932
- SET INTENSITY, 3-942
- SET MARK TO, 3-955
- SET MEMOWIDTH, 3-956
- SET MESSAGE, 3-957
- SET ODOMETER, 3-963
- SET SPACE, 3-995
- SET TEXTMERGE, 3-1005
- SET TEXTMERGE DELIMITERS, 3-1008
- SHOW GET, 3-1020
- SHOW GETS, 3-1023
- SHOW OBJECT, 3-1028
- SROWS(), 3-1051
- SYS functions, 3-1065
- TEXT, 3-1089
- TRANSFORM, 3-1095
- UPDATED(), 3-1106
- I/O - Keyboard
 - ON KEY, 3-704
 - SAVE MACROS, 3-844
 - SET FUNCTION, 3-931
 - SET MACKEY, 3-950
 - SET SCOREBOARD, 3-988
 - SET TYPEAHEAD, 3-1011
- I/O - Menu bars and popups
 - ACTIVATE MENU, 3-162
 - ACTIVATE POPUP, 3-163
 - BAR(), 3-226
 - CLEAR MENUS, 3-290
 - CLEAR POPUPS, 3-290
 - CNTBAR(), 3-297
 - CNTPAD(), 3-300
 - CREATE MENU, 3-332
 - DEACTIVATE MENU, 3-383
 - DEFINE BAR, 3-393
 - DEFINE MENU, 3-403
 - DEFINE PAD, 3-409
 - DEFINE POPUP, 3-416
 - GETBAR(), 3-544
 - GETPAD(), 3-554

I/O - Menu bars and popups (continued)

HIDE MENU, 3-561
HIDE POPUP, 3-562
MENU, 3-643
MENU - System Menu Names, 3-637
MENU TO, 3-643
MENU(), 3-647
MOVE POPUP, 3-680
MRKBAR(), 3-684
MRKPAD(), 3-685
ON BAR, 3-699
ON PAD, 3-711
ON SELECTION BAR, 3-717
ON SELECTION MENU, 3-718
ON SELECTION PAD, 3-720
ON SELECTION POPUP, 3-721
PAD(), 3-731
POP MENU, 3-744
POP POPUP, 3-745
POPUP(), 3-746
PRMBAR(), 3-753
PRMPAD(), 3-754
PROMPT(), 3-759
PUSH MENU, 3-767
PUSH POPUP, 3-768
READ MENU, 3-795
SET BORDER, 3-883
SET MARK OF, 3-952
SET SKIP OF, 3-992
SHOW MENU, 3-1027
SHOW POPUP, 3-1031
SIZE POPUP, 3-1038
SKPBAR(), 3-1042
SKPPAD(), 3-1043

I/O - Mouse

MDOWN(), 3-630

I/O - printing

?, 3-16
??, 3-16
???, 3-20
\\ | \\, 3-14
CLOSE ALTERNATE, 3-293
DEFINE BOX, 3-401
DISPLAY, 3-452
EJECT, 3-477
EJECT PAGE, 3-478
LIST, 3-611
ON PAGE, 3-714
PCOL(), 3-740
PRINTJOB, 3-749

PRINTSTATUS(), 3-750

PROW(), 3-763

SET MARGIN, 3-951

SET PDSETUP, 3-970

SET PRINTER, 3-973

SYS functions, 3-1065

TYPE, 3-1099

I/O - reports and labels

\\ | \\, 3-14

CREATE LABEL, 3-329

CREATE REPORT, 3-339

CREATE REPORT -

Quick Report, 3-339, 3-342

LABEL, 3-601

MODIFY LABEL, 3-659

MODIFY REPORT, 3-670

REPORT, 3-820

I/O - typing/keyboard/mouse

CAPSLock(), 3-263

CHRSaw(), 3-286

CLEAR MACROS, 3-290

CLEAR TYPEAHEAD, 3-292

INSMODE(), 3-588

KEYBOARD, 3-599

LASTKEY(), 3-604

MCOL(), 3-629

MROW(), 3-687

NUMLOCK(), 3-691

ON KEY =, 3-705

ON KEY LABEL, 3-708

PLAY MACRO, 3-742

POP KEY, 3-743

PUSH KEY, 3-766

RESTORE MACROS, 3-826

I/O - typographic conventions, 1-24

I/O - Windows

ACTIVATE WINDOW, 3-170

CLEAR WINDOWS, 3-292

CREATE SCREEN, 3-344

CREATE SCREEN - Quick Screen, 3-347

DEACTIVATE WINDOW, 3-389

DEFINE WINDOW, 3-427

HIDE WINDOW, 3-563

MODIFY SCREEN, 3-673

MOVE WINDOW, 3-682

MWINDOW(), 3-688

RESTORE WINDOW, 3-828

SAVE WINDOW, 3-847

SET BORDER, 3-883

SET SHADOWS, 3-990

SET WINDOW OF MEMO, 3-1016

I/O - Windows (continued)
 SHOW WINDOW, 3-1033
 SYS functions, 3-1065
 WBORDER(), 3-1127
 WCHILD(), 3-1128
 WCOLS(), 3-1130
 WEXIST(), 3-1132
 WLAST(), 3-1136
 WLCOL(), 3-1137
 WLROW(), 3-1138
 WMAXIMUM(), 3-1139
 WMINIMUM(), 3-1140
 WONTOP(), 3-1141
 WOUTPUT(), 3-1142
 WPARENT(), 3-1143
 WREAD(), 3-1144
 WROWS(), 3-1145
 WTITLE(), 3-1146
 WVISIBLE(), 3-1148
 ZOOM WINDOW, 3-1152
 ISALPHA(), 3-590
 ISCOLOR(), 3-591
 ISDIGIT(), 3-592
 ISLOWER(), 3-593
 ISUPPER(), 3-594

J

JOIN, 1-18, 3-595, 4-51
 Joining tables, 3-595

K

Key expressions for index files, 3-597
 Key words, 1-7 - 1-8
 KEY(), 3-597
 Keyboard
 Buffer, 3-286, 3-599
 Characters in buffer, 3-1011
 Defining Function keys, 3-931
 Input/Output, 2-39
 Macros, 3-742, 3-844
 Macros dialog, 3-950
 Modes, 3-263
 Waiting for input, 3-1124
 Keys
 Defined, 1-16
 Display Keyboard Macros dialog, 3-950
 Function, 3-512 - 3-513
 Key presses, 3-796

Last key pressed, 3-604
 Setting keystrokes for controls, 3-943
 Status, 3-988
 Trapping key presses, 3-704 - 3-705,
 3-708
 Values, 3-577

L

LABEL, 1-18
 Label Layout window, 3-329, 3-659
 Labels, 2-41, 3-601
 Language, FoxPro, 1-1, 1-7
 LASTKEY, 3-604
 Leading blanks, 3-190, 3-626
 Left margin, setting, 3-951, 4-26
 Left padding, 3-733
 LEFT(), 3-606
 Leftmost characters, 3-606
 LEN(), 1-23, 3-607
 Length
 Character expressions, 3-607
 Page, 4-39
 Less than, 1-14
 Less than or equal to, 1-14, 3-516
 Libraries, external, 3-945
 LIBRARY, 3-945
 LIKE(), 3-608
 Line feed, 4-29
 Line number, current, 4-40
 Line numbers
 Executing routines at, 3-714
 Returning, 3-219, 3-221, 3-610
 Searching character expressions, 3-777
 Line spacing, 4-46
 LINENO(), 3-610
 Lines
 In memo field, 3-634
 Indenting, 4-25
 Returning from memo field, 3-651
 Linked objects, 3-657
 LIST, 1-18, 3-611, 4-46
 LIST STATUS, 3-611
 Listing
 Disk directories, 3-449
 Records/expressions, 3-611

Lists

- Color pairs, 3-854, 3-891
- Creating, 3-71
- Defined, 1-16
- Described, 3-44
- Names, 1-15

_LMARGIN, 4-26, 4-41, 4-48, 4-60

LOAD, 3-612

Loading

- Binary files, 3-612
- Color set, 3-896
- Printer driver setup, 4-35
- Printer driver setups, 3-970

Local memory variables, 3-751

LOCATE

- CONTINUE and, 3-304
- FOR clause with, 1-18
- FOUND() and, 3-526
- LOOKUP() vs., 3-623

Locating

- Files on disk, 3-616
- Next record, 3-304
- Records with LOCATE, 3-614

LOCFILE(), 3-616

LOCK(), 3-618

Locking

- Multiple record locking, 3-960
- Records, 3-833
- Records with LOCK(), 3-618
- Records/files automatically, 3-947
- SET REPROCESS, 3-983
- Table files, 3-514

LOG(), 3-621

LOG10(), 3-622

Logarithm, 3-621 - 3-622

LOGERRORS, 3-949

Logging compilation errors, 3-949

Logical AND, 1-13

Logical conditions, 3-468

Logical expressions

- Described, 1-9, 1-13
- Operators, 1-13 - 1-14
- Relational operators, 1-14
- UDFs in, 3-1116
- WHILE clauses and, 1-20

Logical inclusive OR, 1-13

Logical negative, 1-13

Logical operators, 1-19

Logical type fields, 1-13

LOOKUP(), 3-623

Loops and looping, 2-24, 3-486, 3-524

Low-level file functions

Described, 2-24

FCHSIZE(), 3-495

FCLOSE(), 3-496

FCLOSE(), 3-496

FCREATE(), 3-498

FEOF(), 3-500

FERROR(), 3-501

FFLUSH(), 3-503

FGETS(), 3-504

FOPEN(), 3-521

FOPEN(), 3-521

FPUTS(), 3-528

FREAD(), 3-531

FSEEK(), 3-532

FWRITE(), 3-541

HEADER(), 3-558

TEXT, 3-1089

LOWER(), 3-625

Lower-case characters, 3-593, 3-625

LTRIM(), 3-626

LUPDATE(), 3-627

M

_MAC, 4-27

MACKEY, 3-950

Macro substitution, 1-16, 3-9

Macros

Assigning to function keys, 3-931

Executing, 3-742

Keyboard, displaying, 3-950

Names, 1-16

Restoring, 3-826

Saving, 3-844

MARGIN, 3-951

Margins

Left, 3-951

Right, 4-48

Setting, 4-26

MARK, 3-952, 3-955

Mark character, 3-952

Marked records, 3-799

Marking

Menu pads, 3-685

Records for deletion, 3-442, 3-729, 3-914

Master index files, 3-726

Master tags, 3-726

Matching character expressions, 3-608

Matching criteria, 3-304

MAX(), 3-628

Maximizing windows, 3-1139

MCOL(), 3-629

MDOWN(), 3-630

MDX(), 3-265, 3-631

MDY(), 3-633

MEMLINE(), 1-23

MEMLINES(), 3-634

Memo fields

Allocating disk space for, 3-882

Copying, 3-307

Defined, 1-16

Filling from files, 3-204

Functions and, 1-23

Modifying, 3-662

Reducing size, 3-729

Restoring macros from, 3-826

Restoring memory variables, 3-824

Returning lines from, 4-28

Returning number of lines, 3-634

Returning specific line, 3-651

Saving keyboard macros, 3-844

Saving memory variables to, 3-846

Saving window definitions to, 3-847

Setting window for editing, 3-1016

Width, 3-956

Memory

Amount remaining, 3-636

Releasing, 3-814

Memory resident strings, 1-23

Memory variable arrays

Creating with DECLARE, 3-390

Creating with DIMENSION, 3-445

Searching for expressions, 3-206

Sorting, 3-210

Memory variable manipulation

&, 3-9

=, 3-13

ACOPY(), 3-159

ADEL(), 3-174

ADIR(), 3-176

AELEMENT(), 3-180

AFIELDS(), 3-182

AINS(), 3-185

ALEN(), 3-187

ASCAN(), 3-206

ASORT(), 3-210

ASUBSCRIPT(), 3-213

AVERAGE, 3-224

CALCULATE, 3-256

CLEAR ALL, 3-289

CLEAR MEMORY, 3-290

COUNT, 3-321

DISPLAY MEMORY, 3-457

GETEXPR, 3-548

INPUT, 3-584

PARAMETERS, 3-735

PRIVATE, 3-751

PUBLIC, 3-764

REGIONAL, 3-805

RELEASE, 3-810

RESTORE FROM, 3-824

SAVE TO, 3-846

STORE, 3-1052

SUM, 3-1060

WAIT, 3-1124

Memory variables

Alias designation, 1-23

Character expressions and, 1-10

Clearing, 3-289

Copying from records to, 3-851

Date expressions and, 1-12

Defining as parameters, 3-735

Defining PRIVATE, 3-751

Defining PUBLIC, 3-764

Displaying contents, 3-457

Expressions and, 1-10

Input to, 3-584

Logical expressions and, 1-13

Macro substitution and, 1-16

Moving fields with GATHER, 3-542

Name expressions and, 1-16

Names, 1-8, 1-14

Numeric expressions and, 1-11

Regional, 3-805

Releasing, 3-810

Restoring, 3-824

Saving, 3-846

Saving expressions to, 3-548

Storing data to, 3-1052

System, 4-1

MEMORY(), 3-636

MEMOWIDTH, 3-956

memvar, 1-15

MENU, 3-637, 3-643

Menu bar

Displaying, 3-1027

Message location, 3-957

Positioning pads, 3-554

System, 3-1001

Menu bar pads, 3-403, 3-409

Menu bars

- Activating with ON BAR, 3-699
- Activating with ON PAD, 3-711
- Active, 3-647
- Assigning routines to, 3-718
- Creating, 3-403
- Described, 3-403
- Displaying, 3-162
- Hiding, 3-561
- Input/Output, 2-35
- Last chosen pad, 3-731
- Number of, 3-544
- Placing on stack, 3-767
- Pulling off stack, 3-744

Menu Builder

- CREATE MENU, 3-332
- MODIFY MENU, 3-665

Menu pads

- Activating popups/menu bars, 3-711
- Assigning routines to, 3-720
- Enabled/disabled, 3-1043
- Last chosen, 3-731
- Marked, 3-685
- Names, 1-14, 3-554
- Number, 3-300
- Options, 3-754
- Prompts, 3-759
- Releasing, 3-810
- Specifying mark characters, 3-952

Menu, displaying, 3-1000

Menu system, 3-403, 3-643

MENU TO, 3-643

MENU(), 3-647

Menus

- Creating, 3-332, 3-643
- Deactivating, 3-383
- Enabling/disabling, 3-992
- Names, 1-14
- Popup, 3-393, 3-643, 3-795
- Releasing, 3-810
- User-defined, 3-893, 3-896

Merging

- Setting delimiters, 3-1008
- Tables, 3-595
- Text, 3-14, 3-1005

MESSAGE, 3-957

Messages, 3-957, 3-962

Microsoft Windows

- Display element, 3-1063
- Status bar, 3-997

MIN(), 3-650

Minimizing windows, 3-434, 3-1140

Minimum expression, 3-650

Minus sign (-), 1-11

_MLINE, 4-28

MLINE(), 1-23, 3-651

MOD(), 3-653

MODAL READ, 3-783

MODIFY COMMAND, 1-3 - 1-4, 3-654

MODIFY FILE, 3-654

MODIFY GENERAL, 3-657

MODIFY LABEL, 3-659

MODIFY MEMO, 3-295, 3-662

MODIFY MENU, 3-665

MODIFY PROJECT, 3-667

MODIFY REPORT, 3-670

MODIFY SCREEN, 3-673

MODIFY STRUCTURE, 3-676

MODIFY WINDOW, 3-677

Modifying

Labels, 3-659

Memo fields, 3-662

Programs, 1-4

Projects, 3-667

Queries, 3-669

Reports, 3-670

Screens, 3-673

Table structure, 3-676

Modulus, 1-11, 3-8, 3-653

Monochrome monitors, 3-891

MONTH(), 3-679

Months, 3-296, 3-557, 3-679

Mouse, 3-959

Button state, 3-630

Input/Output, 2-39

Interval between clicks, 4-12

Pointer column position, 3-629

Returning values from, 3-577

Sensitivity, 3-959

Trapping clicks, 3-708

Mouse pointer

On window, 3-688

Row position, 3-687

MOVE POPUP, 3-680

MOVE WINDOW, 3-682

Moving

File pointer, 3-532

Popups, 3-680

Table record pointers, 3-1040

Through tables, 3-849

Windows, 3-682

MRKBAR(), 3-684
MRKPAD(), 3-685
MROW(), 3-687
MULTILOCKS, 3-960
Multiplication operator, 1-11
MWINDOW(), 3-688

N

Name expressions, 1-16

Names

- Described, 1-14
- Examples, 1-7
- Field, 3-506
- Paths, 3-535
- Proper names, 3-762
- Separating with commas, 1-15
- Table names, 3-356
- Tag, 3-1084

NDX(), 3-689

NEAR, 3-961

Negative, logical, 1-13

Nested READs, 3-782

Network

- Displaying changes by other users, 3-977
- Exclusive files, 3-925
- FLOCK(), 3-514
- FoxPro environment, 2-49
- LOCK(), 3-618
- SET EXCLUSIVE, 3-925
- SET LOCK, 3-947
- SET MULTILOCKS, 3-960
- SET REFRESH, 3-977
- SET REPROCESS, 3-983
- UNLOCK, 3-1102

NEXT scope clause, 1-18

NOT, 1-13, 1-19

Not equal to, 1-14

NOTE, 3-690

NOTIFY, 3-962

Null strings, 1-10

Number sign (#), 1-14

Numbers

- Character expressions, 3-592
- Expression numbers, 1-10

Numeric constants

- Numeric expressions and, 1-11
- Pi, 3-741

Numeric expressions

- ASCII character, 3-285
- Averaging, 3-224

Character strings, 3-1054

Cosines, 3-320

Described, 1-9, 1-11

Formatting, 3-1095

Integer, 3-589

Logical expressions and, 1-13

Operators, 1-11

Place separator, 3-989

Returning absolute values of, 3-157

Returning arc cosine, 3-161

Returning arc sine, 3-208

Returning arc tangents, 3-217

Rounding, 3-836

Square roots, 3-1050

Tangent, 3-1086

Numeric fields

- Adding values, 3-1060

- Totaling, 3-1092

Numeric functions, 2-10

%, 3-8

BETWEEN(), 3-229

CALCULATE, 3-256

CEILING(), 3-267

COS(), 3-320

DTOR(), 3-474

EXP(), 3-487

FLOOR(), 3-516

FV(), 3-540

INT(), 3-589

ISDIGIT(), 3-592

LOG(), 3-621

LOG10(), 3-622

MAX(), 3-628

MIN(), 3-650

MOD(), 3-653

PAYMENT(), 3-739

PI(), 3-741

PV(), 3-771

RAND(), 3-773

ROUND(), 3-836

RTOD(), 3-838

SET DECIMALS, 3-911

SIGN(), 3-1036

SIN(), 3-1037

SQRT(), 3-1050

SYS, 3-1065

TAN(), 3-1086

VAL(), 3-1120

Numeric keypad mode, 3-691

Numeric type fields, 1-11

- NumLock key
 - Current state, 3-691
 - Displaying status, 3-988
- NUMLOCK(), 3-691

O

- Objects
 - Activating with READ, 3-781
 - Current, 3-1121
 - GET object number, 3-692
 - GET objects, redisplaying, 3-1028
- OBJNUM(), 3-692
- OCCURS(), 3-697
- ODOMETER, 3-963
- Offset, page, 4-41
- ON BAR, 3-699
- ON ERROR, 3-702, 3-724
- ON ESCAPE, 3-703, 3-724
- ON KEY, 3-704, 3-724
- ON KEY =, 3-705
- ON KEY LABEL, 3-708, 3-724, 3-743, 3-766
- ON PAD, 3-711
- ON PAGE, 3-714, 3-724
- ON READERROR, 3-715, 3-724
- ON SELECTION BAR, 3-717
- ON SELECTION MENU, 3-718
- ON SELECTION PAD, 3-720
- ON SELECTION POPUP, 3-721
- ON(), 3-724
- One-to-many relationships, 3-991
- Online help facility, 1-5, 3-1, 3-933
- Open dialog, 3-550
- Open... option (File menu), 1-4
- Opening
 - Editing window, 3-657, 3-662
 - File maintenance desk accessory, 3-508
 - Files, 3-498
 - Format files, 3-929
 - Help Window, 3-559
 - Index files, 3-940
 - Label Layout window, 3-329, 3-659
 - Menu Design window, 3-332, 3-665
 - Procedure files, 3-976
 - Project, 3-334
 - Project windows, 3-667
 - Report Layout window, 3-670
 - Reports, 3-339
 - RQBE window, 3-337, 3-669
 - Screen Design window, 3-344, 3-673
 - Table files, 3-1108

- Table files on network, 3-925
- Text editing window, 3-654
- Trace window, 3-921, 3-999
- View window, 3-875, 3-1015
- Operating system, 3-728
- Operator/operand type mismatch, 1-10
- Operators
 - Logical expressions, 1-13 - 1-14, 1-19
 - Modulus, 3-8
 - Numeric expressions, 1-11
 - Relational, 1-13
- OPTIMIZE, 3-964
- Options
 - Specifying mark characters for, 3-952
 - Square brackets and, 1-9
 - Vertical bars and, 1-9
- Options, menu, 3-544, 3-637
- Options, menu pad, 3-754
- Options, popup
 - Activating, 3-699
 - Assigning routines to, 3-717
 - Described, 3-403
 - Number, 3-226, 3-297
 - Prompts, 3-753
 - PROMPTS(), 3-759
 - Releasing, 3-810
- Options, popup menu, 3-393
- OR logical operator, 1-13, 1-19
- ORDER, 3-965
- Order in which new READs called, 3-782
- Order of precedence, 1-11, 1-13
- ORDER(), 3-726
- Ordering tables, 3-571
- OS(), 3-728
- Output
 - Directing to file, 3-876
 - Lines of text, 3-14, 3-1089
 - Number of copies to print, 4-33
 - Page offset, 4-41
 - Printer and port, 3-973
 - Printer pitch, 4-42
 - Query results to FoxGraph, 4-19
 - RQBE results to FoxGraph, 4-24
 - Setting left margin, 3-951
 - Text merge commands, 4-52
 - To printer with ??? command, 3-20
 - To screen, 3-169
 - To screens/windows, 3-905, 3-917
 - Typographic conventions for, 1-24
 - Window, 3-1142
- Overstriking printer, 4-32

P

PACK, 3-729, 3-1003, 4-51

Pad name, 1-15 - 1-16

Pad options, 3-754

PAD(), 3-731

PADC(), 3-733

PADL(), 3-733

PADR(), 3-733

Pads, menu

- Activating bars, 3-711

- Assigning routines to, 3-720

- Last chosen, 3-731

- Marked, 3-685

- Names, 3-554

- Number, 3-300

- Prompts, 3-759

- Releasing, 3-810

- System menu names, 3-637

Pads, menu bar

- Creating, 3-409

- Described, 3-403

_PADVANCE, 3-478, 4-29

Page

- Advance method, 4-29

- Eject setting, 4-37

- Ending, 4-38

- Length, 4-39

- Offset, 4-41

Page number, current, 4-30

_PAGENO, 4-30, 4-38

Pages

- Advancing, 3-477 - 3-478

- Pausing between, 4-47

- Printing, 4-31

PARAMETERS, 3-735

PARAMETERS(), 3-737

Parameters, passing to UDFs, 3-1115

Parent windows, 3-1143

Parentheses ()

- Functions and, 1-7 - 1-8

- Grouping logical expressions, 1-13

- Grouping numeric expressions, 1-11

Parm list, 1-16

Passing parameters to UDFs, 3-1115

Pasting commands, 1-2 - 1-3

Path, 1-16, 3-535, 3-969

PATH, 3-547, 3-969

Path name display, 3-930

Pausing between pages, 4-47

Payment amounts, 3-739

PAYMENT(), 3-739

_PBPAGE, 4-30 - 4-31, 4-38

_PCODE, 4-36

PCOL(), 3-477, 3-740

_PCOLNO, 4-32

_PCOPIES, 4-33

_PDPARMS, 4-35

_PDONLOAD, 4-34

_PDRIVER, 4-34

PDSETUP, 3-970, 4-35

_PECODE, 4-36

_PEJECT, 4-37

_PEPAGE, 4-30 - 4-31, 4-38

Percent sign (%), 1-11, 3-8

Phonetic differences, 3-444

Phonetic representations, 3-1048

PI(), 3-741

PICTURE codes, 3-34

Pitch, printer, 4-42

PLAY MACRO, 3-742

_PLENGTH, 4-29, 4-39, 4-47

_PLINENO, 4-39 - 4-40, 4-47

_PLINENO, 4-29

_PLOFFSET, 4-41

Plus sign (+)

- Addition operator, 1-11

- High intensity colors, 3-881

POINT, 3-972

Pointer, location, 3-961

Pointer, mouse, 3-629, 3-687 - 3-688

Pointer, record, 3-555

Pointers, file, 3-532

Pointers, record, 3-1040

POP KEY, 3-743

POP MENU, 3-744

POP POPUP, 3-745

Popup bars, 3-297, 3-684

Popup commands, 3-957

Popup menus, 3-148, 3-393, 3-643, 3-637

Popup options

- Assigning routines to, 3-717

- Number, 3-226

- Prompts, 3-753, 3-759

- Releasing, 3-810

- Specifying mark characters, 3-952

POPUP(), 3-746

Popups

- Activating, 3-699, 3-711, 3-795
- Active, 3-746
- Assigning routines to, 3-721
- Changing size, 3-1038
- Creating, 3-83, 3-416
- Deactivating, 3-386
- Described, 3-44, 3-403
- Displaying, 3-163, 3-1031
- Hiding, 3-562
- Input/Output, 2-35
- Moving, 3-680
- Names, 1-14
- Number of bars, 3-544
- Placing on stack, 3-768
- Popup name lists, 1-15
- Pulling off stack, 3-745
- Releasing, 3-810

Ports

- Closing, 3-496
- Defined, 1-16
- Number of bytes, 3-531
- Opening, 3-521
- Series of bytes, 3-504
- Specifying, 3-973
- Writing character strings to, 3-528, 3-541

Position, cursor column, 3-301

Positioning record pointer, 3-555

Pound sign (#), 1-14

_PPITCH, 4-42

_PQUALITY, 4-43

Precedence, 1-11

Preface text, 4-44

Present value of investment, 3-771

_PRETEXT, 4-44

Print codes, 4-45

Print quality, 4-43

Printer, 3-973

- Advancing to next page, 3-477
- Column position, 3-740
- Current row position, 3-763
- Driver application, 4-42 - 4-43
- Enabling/disabling, 3-973
- Line spacing, 4-46
- Pitch, 4-42
- Status, 3-750

Printer driver

- Installed message, 4-34
- Loading/clearing setup, 3-970
- Setup, 4-35

Specifying interface program, 4-21

Specifying program, 4-34

Printing

??? command and, 3-20

Activating job settings, 3-749

Backspacing, 4-32

Enabling output, 3-973

Ending page number, 4-38

First page, 4-31

Input/Output, 2-34

Number of copies, 4-33

Output to, 3-917

Overstriking, 4-32

Page offset, 4-41

Print codes, 4-36

Printer driver program, 4-34

Printer driver setup, 4-35

Setting left margin, 3-951

PRINTJOB ... ENDPRINTJOB, 3-749

PRINTSTATUS(), 3-750

PRIVATE, 3-751

Private system memory variables, 4-1

PRMBAR(), 3-753

PRMPAD(), 3-754

PROCEDURE, 3-756, 3-976

Procedure files, 3-976

Procedure name, 1-16

Program execution

BUILD APP, 3-252

BUILD EXE, 3-253

BUILD PROJECT, 3-254

CALL, 3-260

CANCEL, 3-262

CLEAR, 3-290

CLOSE PROCEDURE, 3-294

Commands and functions, 2-22

COMPILE, 3-302

CREATE PROJECT, 3-334

#DEFINE ... #UNDEF, 3-391

DO, 3-464

FUNCTION, 3-538

#IF ... #ENDIF, 3-565

LOAD, 3-612

MODIFY PROJECT, 3-667

PARAMETERS, 3-735

PARAMETERS(), 3-737

PROCEDURE, 3-756

RELEASE MODULE, 3-814

RESUME, 3-829

RETURN, 3-831

RUN, 3-840

Program execution (continued)

- SET ESCAPE, 3-922
- SET LIBRARY, 3-945
- SET LOGERRORS, 3-949
- SET PROCEDURE, 3-976
- SET SYSMENU, 3-1001
- Speed, 4-54
- SUSPEND, 3-1062
- SYS functions, 3-1065
- WAIT, 3-1124

Program files

- COMPILE, 3-302
- Creation date/time, 3-916
- Executing with DO, 3-464
- See also* FoxPro User's Guide

PROGRAM(), 3-758

Programming commands and functions

- Array manipulation, 2-27
- Event handlers, 2-28
- Low-level file functions, 2-24
- Memory variable manipulation, 2-25
- Program execution, 2-22

Programming with FoxPro language, 1-1

Programs

- Command window vs., 1-3
- Creating, 1-3 - 1-4
- Executing, 1-3 - 1-4, 3-840, 3-921
- Modifying, 1-4
- Saving, 1-3
- Suspending execution, 3-1062

Project files, 3-252

Project Manager, alerting, 3-491

Project window, 3-334, 3-667

Projects

- Creating, 3-254, 3-334
- Modifying, 3-667

PROMPT(), 3-759

Prompts

- PRMBAR(), 3-753
- PROMPT(), 3-759

Proper names, 3-762

PROPER(), 3-762

Protecting files, 3-987

PROW(), 3-477, 3-763, 4-40

_PSCODE, 4-45

_PSPACING, 4-46 PUBLIC, 3-764

Public system memory variables, 4-1

Push buttons, 3-44, 3-98

PUSH KEY, 3-766

PUSH MENU, 3-744, 3-767

PUSH POPUP, 3-768

PUTFILE(), 3-769

PV(), 3-771

_PWAIT, 4-47

Q

Quality, print, 4-43

Queries

- Creating, 3-337

- Modifying, 3-669

- via SELECT-SQL, 3-862

Query results

- _GENGRAPH and, 4-19

- Output to FoxGraph, 4-19

Question mark (?)

- In ? ! ?? command, 3-16

- In ??? command, 3-20

Quick Screen, 3-347

QUIT, 3-772

Quitting FoxPro, 3-772

R

Radians

- Converting from degrees, 3-474

- Converting to degrees, 3-838

Radio buttons, 3-45, 3-113

RAND(), 3-773

Random numbers, 3-773

Ranges, record, 1-18

RAT(), 3-776

RATLINE(), 3-777

RDLEVEL(), 3-779

READ

- Foundation READs, 3-788

- GET-less READs, 3-788

- Level, 3-779

- MODAL, 3-783

- VALID, 3-784

- Windows, 3-1144

- WITH, 3-784

READ MENU, 3-795

READKEY(), 3-796

Rebuilding open index files, 3-807

RECALL, 1-18, 3-799

RECCOUNT(), 3-801

RECNO(), 3-304, 3-802

RECORD, 1-18

Record pointer, 3-555

Record ranges, 1-18

Records

- Adding to table from another file, 3-193
- Adding to tables from array, 3-198
- Appending to table, 3-192
- Appending with INSERT -SQL, 3-586
- Automatic locking, 3-947
- Carrying data to new records, 3-886
- Changing with EDIT, 3-476
- Copying from, 3-851
- Copying table structure to, 3-309
- Copying to array, 3-317
- Counting, 3-321
- Current number, 3-802
- Deleted, 3-442
- Deleting, 3-438
- Displaying, 3-452
- Displaying changes, 3-977
- Displaying with LIST, 3-611
- Filtered, 3-927
- Inserting, 3-585
- Locating, 3-614
- Locking, 3-833, 3-983
- Locking multiple, 3-960
- Locking with LOCK(), 3-618
- Manipulation commands, 2-17
- Marked for deletion, 3-914
- Moving pointers, 3-1040
- Pointer location, 3-961
- Ranges, 1-18
- Recalling marked records, 3-799
- Removing all, ZAP for, 3-1151
- Removing with PACK, 3-729
- Returning number of, 3-801
- Returning size, 3-804
- Searching, 3-511
- SET NEAR, 3-961
- Tallying, 4-51
- Unlocking, 3-1102
- Updating, 3-816

RECSIZE(), 3-804

References, undefined, 3-491

Referencing work areas, 1-23

REFRESH, 3-977

REGIONAL, 3-805

REINDEX, 3-807, 3-1003, 4-51

RELATION(), 3-808, 3-979, 3-982

Relational expressions, 3-808

Relational operators, 1-13 - 1-14, 1-19

Relationships

- One-to-many, 3-991
- Tables, 3-979, 3-982

RELEASE, 3-810

RELEASE MODULE, 3-814

Releasing

- Memory, 3-814
- Record/file locks, 3-1102
- Table elements, 3-810

Remainder, 1-11, 3-8, 3-653

Removing

- All records, 3-1151
- Leading blanks, 3-626
- Marked records, 3-729
- Menus, 3-383
- Popups, 3-386
- Relationships, 3-982
- Space with SET SPACE, 3-995
- Trailing blanks, 3-1096
- Windows, 3-389

RENAME, 3-815

Repeating character expressions, 3-819

REPLACE

- Directing status to screen/window, 3-1003
- Field data, 3-816
- FOR clause with, 1-18
- Memo fields and, 1-23
- Returning status, 4-51

REPLICATE(), 3-819

REPORT, 1-18, 3-820

Report Layout window, 3-339, 3-670

Report Writer, 3-339

Reporting intervals, 3-963

Reports

- Creating, 3-339, 3-342
- Executing routines in, 3-714
- Input/Output, 2-41
- Modifying, 3-670
- Producing, 3-820

REPROCESS, 3-983

Reserved words

- See Key words

RESOURCE, 3-986

Resource files, updating, 3-986

REST scope clause, 1-18

RESTORE FORM, 3-824

RESTORE MACROS, 3-826

RESTORE SCREEN, 3-827

RESTORE WINDOW, 3-828

RESUME, 3-829

Resuming program execution, 3-829

Retrieving

- Color set, 3-896

- Data with SELECT (SQL), 3-862

RETRY, 3-830

RETURN, 3-831

Returning

- Arc cosine, 3-161

- Arc sine, 3-208

- Calculator value, 4-8

- Calendar/Diary date, 4-15

- Character strings, 3-190

- Clipboard contents, 4-9

- Control to calling procedure, 3-831

- Element numbers, 3-180

- Lines from memo fields, 4-28

- Number of columns in array, 3-187

- Number of elements in array, 3-187

- Number of rows in array, 3-187

Right margin, 4-48

RIGHT(), 3-832

Rightmost characters, 3-832

RLOCK(), 3-833

_RMARGIN, 4-48, 4-50, 4-60

ROUND(), 3-836

Rounding numeric expressions, 3-836

Routines

- Assigning to menu bars, 3-718

- Assigning to menu pads, 3-720

- Assigning to popup options, 3-717

- Assigning to popups, 3-721

- Running, 3-260

ROW(), 3-837

Rows

- Current cursor position, 3-837

- Current printer position, 3-763

- Defined, 1-16

- Deleting from array, 3-174

- Inserting into arrays, 3-185

- Number in window, 3-1145

- Performing I/O at, 3-21

- Returning location, 3-1138

- Returning number, 3-187

- Returning number on screen, 3-1051

- Subscript, returning, 3-213

RQBE results, 4-24

RQBE window, 3-337, 3-669

RTOD(), 3-838

RTRIM(), 3-839

RUN, 3-840

Running custom binary routines, 3-260

Rushmore technology, 1-18, 3-964

S

SAFETY, 3-987

Save As dialog, 3-769

SAVE MACROS, 3-844

SAVE option, 3-247

SAVE SCREEN, 3-845

SAVE TO, 3-846

SAVE WINDOW, 3-847

Saving

- Buffers to disk, 3-879

- Changes to data, 3-269

- Expressions, 3-548

- Indexes to disk, 3-517

- Keyboard macros, 3-844

- Memory variables, 3-846

- ON KEY LABEL commands, 3-766

- Program files, 1-4

- Programs, 1-3

- Screen/window images, 3-845

- Tables to disk, 3-517

- Window definitions, 3-847

SAY clauses, 3-22

SCAN, 1-18

SCAN ... ENDSCAN, 3-849

SCAN loops, 3-486

SCATTER, 3-851 - 3-852

SCHEME(), 3-854

SCOLS(), 3-855

Scope, 1-16

Scope clauses, 1-18

SCOREBOARD, 3-988

Screen Builder, 3-344, 3-673

Screen generation program, 4-23

Screen layout window, 3-344, 3-673

Screens

- Activating, 3-169

- Attributes/colors, 3-881

- Changing colors within, @ ... FILL, 3-146

- Clearing with @ ... CLEAR, 3-145

- Columns available, 3-855

- Creating, 3-344

- Creating with Quick Screen, 3-347

- Directing output to file, 3-876

- Directing status information to, 3-1003

- Modifying, 3-673

- Mouse pointer column position, 3-629

- Mouse pointer row position, 3-687

- Number of rows, 3-1051

- Output to, 3-905, 3-917

- Removing menus from, 3-383

Screens (continued)

- Restoring from memory, 3-827
- Saving images, 3-845
- Scrolling, 3-856
- Specifying when to clear, 3-888

SCROLL, 3-856

Search commands

- FIND, 3-511
- LOOKUP(), 3-623
- SEEK, 3-858
- SELECT-SQL, 3-862

Searching

- Character expressions, 3-776 - 3-777, 3-1055
- CONTINUE, 3-304
- Expressions, 3-582
- Files on disk, 3-616
- Indexed tables, 3-858
- LOCATE, 3-614
- Memory variable arrays
 - for expressions, 3-206
- Records, 3-526, 3-961
- Setting directory path for, 3-969
- Tables, 3-233, 3-623

Seconds since midnight, 3-857

SECONDS(), 3-857

SEEK, 3-526, 3-858

SEEK(), 3-858

Seeking indexed tables, 3-858

SELECT, 1-22, 3-860

SELECT - SQL, 3-1003

SELECT - SQL

- DISTINCT clause, 3-863
- FROM clause, 3-865
- HAVING clause, 3-872
- ORDER BY clause, 3-873
- WHERE clause, 3-868

SELECT(), 3-861

Sensitivity, mouse, 3-959

SEPARATOR, 3-989

SET, 3-875, 3-1017

SET ALTERNATE, 3-876

SET ANSI, 3-877

SET AUTOSAVE, 3-879

SET BELL, 3-880

SET BLINK, 3-881

SET BLOCKSIZE, 3-882

SET BORDER, 3-883

SET BRSTATUS, 3-885

SET CARRY, 3-886

SET CENTURY, 3-887

SET CLEAR, 3-888

SET CLOCK, 3-889

SET COLOR commands, 3-890 - 3-892

SET COLOR OF, 3-893

SET COLOR OF SCHEME, 3-894

SET COLOR SET, 3-896

SET COLOR TO, 3-896

SET commands, return values, 3-1018

SET COMPATIBLE, 3-898

SET CONFIRM, 3-904

SET CURRENCY, 3-906

SET CURSOR, 3-908

SET DATE, 3-909

SET DEBUG, 3-910

SET DECIMALS, 3-911

SET DEFAULT, 3-912

SET DELETED, 3-914

SET DELIMITERS, 3-915

SET DEVELOPMENT, 3-916

SET DEVICE, 3-917

SET DISPLAY, 3-918

SET DOHISTORY, 3-920

SET ECHO, 3-921

SET ESCAPE, 3-922

SET EXACT, 3-923

SET EXCLUSIVE, 3-925

SET FIELDS, 3-926

SET FILTER, 3-510, 3-927

SET FIXED, 3-928

SET FORMAT, 3-929

SET FULLPATH, 3-930

SET FUNCTION, 3-931

SET HEADING, 3-932

SET HELP, 3-933

SET HELPFILTER, 3-935

SET HOURS, 3-939

SET INDEX, 3-940

SET INTENSITY, 3-942

SET KEYCOMP, 3-943

SET LIBRARY, 3-945

SET LOCK, 3-947

SET LOGERRORS, 3-949

SET MACKEY, 3-950

SET MARGIN, 3-951, 4-41

SET MARK OF, 3-952

SET MARK TO, 3-955

SET MEMOWIDTH, 3-956

SET MESSAGE, 3-957

SET MOUSE, 3-959

SET MULTILOCKS, 3-960

SET NEAR, 3-961

- SET NOTIFY, 3-962
- SET ODOMETER, 3-963
- SET OPTIMIZE, 3-964
- SET PALETTE, 3-968
- SET PATH, 3-969
- SET PDSETUP, 3-970
- SET POINT, 3-972
- SET PRINTER, 3-973, 4-40
- SET PROCEDURE, 3-976
- SET REFRESH, 3-977
- SET RELATION, 3-979
- SET RELATION OFF, 3-982
- SET REPROCESS, 3-983
- SET RESOURCE, 3-986
- SET SAFETY, 3-987
- SET SCOREBOARD, 3-988
- SET SEPARATOR, 3-989
- SET SHADOWS, 3-990
- SET SKIP, 3-991
- SET SKIP OF, 3-992
- SET SPACE, 3-995
- SET STATUS, 3-996
- SET STATUS BAR, 3-997
- SET STEP, 3-999
- SET STICKY, 3-1000
- SET SYSMENU, 3-1001
- SET TALK, 3-1003
- SET TEXTMERGE, 3-1005, 4-44, 4-52
- SET TEXTMERGE DELIMITERS, 3-1008
- SET TEXTMERGE TO, 4-52
- SET TOPIC (Help), 3-1009
- SET TRBETWEEN, 3-1010
- SET TYPEAHEAD, 3-1011
- SET UNIQUE, 3-1014
- SET VIEW, 3-1015
- SET WINDOW OF MENU, 3-1016
- SET(), 3-1017
- Setting relations among tables, 3-979, 3-991
- Setup printer driver, 3-970
- SHADOWS, 3-990
- Sharing
 - Table files on network, 3-925
- SHOW GET, 3-1020, 3-1023
- SHOW GETS, 3-1023
- SHOW MENU, 3-1027
- SHOW OBJECT, 3-1028
- SHOW POPUP, 3-1031
- SHOW WINDOW, 3-427, 3-1033
- SIGN(), 3-1036
- Signs, numeric value, 3-1036
- SIN(), 3-223, 3-1037

- Single quotation marks (' '), 1-10
- Single-spacing, 4-46
- Size
 - Fields, 3-534
 - File, 3-495
 - Memo fields, 3-729
 - Records, 3-804
 - Table headers, 3-558
 - Windows, 3-1152
 - Windows display element, 3-1063
- SIZE POPUP, 3-1038
- Skel, 1-16
- Skeleton, file, 3-176
- SKIP, 3-991 - 3-992, 3-1040
- SKPBAR(), 3-1042
- SKPPAD(), 3-1043
- Slash (/), 1-11
- SORT, 1-18, 3-1003, 3-1044, 4-51
- Sorting
 - Memory variable arrays, 3-210
 - Table files, 3-1044
- Sound, bell, 3-880
- SOUNDEX(), 3-1048
- Sounds, phonetic, 3-1048
- Source code, 3-565
- SPACE, 3-995
- SPACE(), 3-1049
- Space, disk, 3-451
- Space, inserting, 3-995
- Spaces, character strings, 3-1049
- Spacing, printer line, 4-46
- Speed of program execution, 4-54
- Spinners
 - Creating, 3-127
 - Defined, 3-45
- Spreadsheet file types, 3-569
- SQL commands, 3-586
- SQL SELECT, 3-862
- SQL string comparisons, 3-877
- SQRT(), 3-1050
- Square brackets, 1-9
- Square roots, 3-1050
- SROWS(), 3-1051
- Starting
 - Print codes, 4-45
 - Subroutines, 3-538, 3-756
- _STARTUP, 4-49
- STARTUP.APP, 4-49
- Statement, defined, 1-7
- Statements, defined, 1-16
- Statistical operations, 3-256

Status

- Directing to screen/window, 3-1003
- FoxPro environment, 3-459
- Keys, 3-988
- Printer, 3-750
- SET, 3-1017
- Status bar, 3-885, 3-957, 3-996 - 3-997
- STATUS BAR, 3-997
- STEP, 3-999
- STICKY, 3-1000
- STORE, 3-1052, 4-52
- Storing
 - Buffers to disk, 3-879
 - Compilation errors to file, 3-949
 - Data, 3-1052
 - Memory variables, 3-846
- STR(), 3-1054
- String literals, 1-10
- String order, 3-878
- Strings
 - See also* String literals, Character strings, Null strings
 - Accepting character string data, 3-158
 - Border, 3-883
 - Matching, 3-923
 - Memory resident, 1-23
 - Returning character strings with all blanks, 3-190
 - Specifying SQL string comparisons, 3-877
- STRTRAN(), 3-1055
- Structured programming
 - Commands and functions, 2-24
 - DO CASE, 3-468
 - EXIT, 3-486
 - FOR ... ENDFOR, 3-524
 - #IF ... #ENDIF, 3-565
 - IIF(), 3-568
 - SCAN, 3-849
- Structures, table, 3-308, 3-461
- STUFF(), 3-1057
- Subroutines, defining start, 3-538, 3-756
- Subscripts, 3-213
- Substituting macros, 3-9
- SUBSTR(), 1-23, 3-1059
- Subtraction operator, 1-11
- SUM, 1-18, 3-1003, 3-1060, 4-51
- SUSPEND, 3-1062
- Suspending, 3-1062
- Switching
 - Display modes, 3-918

Symbol

- Currency, 3-906
- Syntax
 - Summary, 3-46
 - Typographic conventions for, 1-7
- SYS() functions, 3-1065
- SYSMENU, 3-1001
- SYSMETRIC(), 3-1063
- System information, 3-1065
- System memory variables
 - _ALIGNMENT, 4-3
 - _BOX, 4-6
 - _CALCMEM, 4-7
 - _CALCVALUE, 4-8
 - _CLIPTEXT, 4-9
 - _CUROBJ, 4-11
 - _DBLCLICK, 4-12
 - _DIARYDATE, 4-15
 - _FOXDOC, 4-17
 - _GENGRAPH, 4-19
 - _GENMENU, 4-20
 - _GENPD, 4-21
 - _GENSCRN, 4-23
 - _GENXTAB, 4-24
 - _INDENT, 4-25
 - Listed, 4-1
 - _LMARGIN, 4-26
 - _MAC, 4-27
 - _MLINE, 4-28
 - New and enhanced, 3-3
 - _PADVANCE, 4-29
 - _PAGENO, 4-30
 - _PBPAGE, 4-31
 - _PCODE, 4-36
 - _PCOLNO, 4-32
 - _PCOPIES, 4-33
 - _PDRIVER, 4-34
 - _PDSETUP, 4-35
 - _PEJECT, 4-37
 - _PEPAGE, 4-38
 - _PLENGTH, 4-39
 - _PLINENO, 4-40
 - _PLOFFSET, 4-41
 - _PPITCH, 4-42
 - _PQUALITY, 4-43
 - _PRETEXT, 4-44
 - _PSCODE, 4-45
 - _PSPACING, 4-46
 - _PWAIT, 4-47
 - _RMARGIN, 4-48
 - _STARTUP, 4-49

System memory variables (continued)

- _TABS, 4-50
- _TALLY, 4-51
- _TEXT, 4-52
- _THROTTLE, 4-54
- _TRANSPORT, 4-55
- _UNIX, 4-57
- _WINDOWS, 4-58
- _WRAP, 4-59

System menu bar, 3-1001

System Menu Names, 3-637

System menu, 1-5, 3-1000

System messages, 3-962

T

.T. true logical value, 1-13

Table fields, 1-10

Table operations

- Closing, 3-285, 3-293
- Copy memo, 3-307
- Copying, 3-305
- Creating, 3-323
- Deleting all records, 3-1151

Table structure, 3-676

Tables

- Aliases, 1-14, 1-21
- Appending records, 3-192
- Changing records, 3-476
- Copying from, 3-851
- Copying to file, 3-312
- Counting records, 3-321
- Creating, 3-323, 3-349
- Editing with CHANGE, 3-269
- Exclusive use, 3-925
- Field names, 1-14
- Fields, 3-926
- File names, 1-14
- Index commands and functions, 2-19
- Joining, 3-595
- Manipulation commands, 2-15
- Moving record pointers, 3-1040
- Moving through, 3-849
- One-to-many relationships, 3-991
- Open, 3-1113
- Opening files, 3-1108
- Placing structure information
 - into arrays, 3-182
- Record manipulation commands, 2-17
- Relationships, 2-20, 3-979
- Retrieving data from, 3-862

Saving changes to disk with FLUSH, 3-517

Searching, 3-623, 3-858

Seeking indexed tables, 3-858

Sorting, 3-1044

Structure, 3-308 - 3-309, 3-461

Updating, 3-1104

Work areas, 1-21

Tables - field manipulation

- AFIELDS(), 3-182
- ALIAS(), 3-189
- DBF(), 3-356
- DELETED(), 3-442
- EOF(), 3-480
- FCOUNT(), 3-497
- FIELD(), 3-506
- FOUND(), 3-526
- FSIZE(), 3-534
- GATHER, 3-542
- LUPDATE(), 3-627
- NDX(), 3-689
- RECNO(), 3-802
- RECSIZE(), 3-804
- SCATTER, 3-851
- SELECT (SQL), 3-862
- SET BLOCKSIZE, 3-882
- SET EXACT, 3-923
- SET TEXTMERGE, 3-1005
- SET TEXTMERGE DELIMITERS, 3-1008
- SET WINDOW OF MEMO, 3-1016
- SYS functions, 3-1065

Tables - indexes

- CDX(), 3-265
- CLOSE INDEX, 3-294
- COPY INDEXES, 3-306
- COPY TAG, 3-311
- DELETE TAG, 3-441
- INDEX, 3-571
- KEY(), 3-597
- MDX(), 3-631
- NDX(), 3-689
- ORDER(), 3-726
- REINDEX, 3-807
- SET INDEX, 3-940
- SET ORDER, 3-965
- SET UNIQUE, 3-1014
- SYS functions, 3-1065
- TAG(), 3-1084

Tables - record manipulation

- CONTINUE, 3-304
- FILTER(), 3-510
- FIND, 3-511
- GO/GOTO, 3-555
- INSERT - SQL, 3-586
- LOCATE, 3-614
- LOOKUP(), 3-623
- SEEK, 3-858
- SEEK(), 3-858
- SET FILTER, 3-927
- SET NEAR, 3-961
- SET RELATION, 3-979
- SKIP, 3-1040

Tables - relations

- RELATION(), 3-808
- SELECT (SQL), 3-862
- SET RELATION, 3-979
- SET RELATION OFF, 3-982
- SET SKIP, 3-991
- TARGET(), 3-1087

Tables - table manipulation

- APPEND, 3-192
- APPEND FROM, 3-193
- APPEND MEMO, 3-204
- AVERAGE, 3-224
- BROWSE, 3-233
- CHANGE, 3-269
- CLOSE MEMO, 3-295
- COPY MEMO, 3-307
- COPY STRUCTURE, 3-308
- COPY STRUCTURE
 - EXTENDED, 3-309
- COPY TO, 3-312
- COUNT, 3-321
- CREATE, 3-323
- CREATE CURSOR, 3-325
- CREATE FROM, 3-327
- CREATE QUERY, 3-337
- CREATE TABLE, 3-349
- DELETE, 3-438
- DELETED(), 3-442
- DISPLAY STRUCTURE, 3-461
- EXPORT, 3-488
- FLUSH, 3-517
- IMPORT, 3-569
- INSERT, 3-585
- JOIN, 3-595
- MEMLINES(), 3-634
- MLINE(), 3-651
- MODIFY MEMO, 3-662

- MODIFY GENERAL, 3-657
- MODIFY STRUCTURE, 3-676

- PACK, 3-729
- RECALL, 3-799
- REPLACE, 3-816
- SELECT, 3-860
- SELECT (SQL), 3-862
- SET ANSI, 3-877
- SET AUTOSAVE, 3-879
- SET DELETED, 3-914
- SET FIELDS, 3-926
- SET OPTIMIZE, 3-964
- SORT, 3-1044
- SUM, 3-1060
- TOTAL, 3-1092
- UPDATE, 3-1104
- USE, 3-1108
- USED(), 3-1113
- ZAP, 3-1151

_TABS, 4-50

TAG(), 3-1084

Tags

- Copying, 3-311
- Deleting, 3-441
- Master, 3-726, 3-965
- Names, 3-1084

TALK, 3-1003

_TALLY, 4-51

Tallying records, 4-51

TAN(), 3-1086

Tangent, 3-217, 3-1086

TARGET(), 3-1087

Terminating FoxPro session, 3-772

Terminating key, 3-904

Text, 4-52

- Aligning within margins, 4-3

- Boxes, 3-401

- Merging, 3-1005

- Output, 3-14, 3-1089, 4-52

- Preface, 4-44

- Setting merge delimiters, 3-1008

- Width, 3-1097

TEXT - ENDTEXT, 3-1089

Text edit regions, 3-45, 3-135

Text editing window, 3-654

Text lines, 1-16

Text merge, 3-1089

TEXTMERGE, 3-1005, 3-1008

_THROTTLE, 4-54

- Time
 - Checking, 3-916
 - Data type categories, 2-12
 - Format, setting, 3-939
 - TIME(), 3-1091
- TIME(), 3-1091
- Titles, window, 3-1146
- Toggling, Insert mode, 3-588
- TOPIC (Help), 3-1009
- Topic, defined, 1-16
- Topics, help, 3-935, 3-1009
- TOTAL, 1-18, 3-1003, 3-1092, 4-51
- Totaling numeric fields, 3-1092
- Trace window
 - Enabling, 3-910
 - Opening during execution, 3-921
 - Opening with SET STEP, 3-999
 - Specifying program execution speed, 4-54
 - Tracing between breakpoints, 3-1010
- Trailing blanks, 3-190, 3-839, 3-1096
- TRANSFORM(), 3-1095
- Translating characters, 3-288
- _TRANSPORT, 4-55
- Transporting FoxPro formats, 4-55
- Trapping
 - Any key, 3-704 - 3-705
 - Errors, 3-484, 3-702
 - Escape, 3-703
- TRBETWEEN, 3-1010
- Trigonometric functions, 3-320
- TRIM(), 3-1096
- Trimming character strings, 3-839
- Triple-spacing, 4-46
- True, logical expression value, 1-13
- TXTMETRIC(), 3-1097
- TYPE, 3-1099
- TYPE(), 3-1101
- TYPEAHEAD, 3-1011
- Typing, 2-39
- Typographic conventions, 1-7, 1-24

U

- UDFs
 - User-defined functions, 3-1114
 - Warnings about, 3-864
 - Where allowed (Table), 3-1117
- #UNDEF, 3-391
- UNIQUE, 3-1014
- Unique index keys, 3-1014
- _UNIX, 4-57

- UNLOCK, 3-1102
- UPDATE, 3-1104, 4-51
- UPDATED(), 3-1106
- Updating
 - Data, 3-1106
 - Last updated date, 3-627
 - Resource files, 3-986
 - Table records, 3-816
 - Tables, 3-1104
- UPPER(), 3-1107
- Upper-case characters, 1-8, 3-594, 3-1107
- USE, 3-1108 - 3-1109
- USED(), 3-1113
- User-assigned aliases, 1-21
- User-defined functions (UDFs), 3-1114
- User interface, FoxPro, 1-2

V

- VAL(), 3-1120
- VALID, 3-57, 3-784
- Values
 - Investments, 3-540
 - Returning with UDFs, 3-1114
- VAL(), 3-1120
- var, 1-16
- Variable parameters in UDFs, 3-1115
- Variables
 - See Memory variables
- Variables, system memory, 4-1
- VERSION(), 3-1123
- Vertical bar (|), 1-9
- VIEW, 3-1015
- View files, creating, 3-351
- View window
 - Opening, 3-875
 - SET VIEW, 3-1015
- Visible windows, 3-1148

W

- WAIT, 3-1124
- Waiting for keyboard input, 3-1124
- WBORDER(), 3-1127
- WCHILD(), 3-1128
- WCOL(), 3-1137
- WCOLS(), 3-1130
- WEXIST(), 3-1132
- WFONT(), 3-1133
- WHEN clauses, 3-59

WHILE clause, 1-18, 1-20

Width

 Memo fields, 3-956

 Text strings, 3-1097

Window titles, how derived, 3-1146

Windows, 4-58

 Borders, 3-1127

 Browse, 3-885

 Child, 3-1128

 Clearing, 3-292

 Clearing with @ ... CLEAR, 3-145

 Column location, 3-1137

 Columns in, 3-1130

 Creating, 3-427

 Current output, 3-1142

 Current READ, 3-1144

 Deactivating, 3-389

 Debug, 3-910

 Definitions, saving, 3-847

 Directing output to file, 3-876

 Directing status information to, 3-1003

 Display element size, 3-1063

 Displaying, 3-170, 3-1033

 Existing, 3-1132

 Frontmost, 3-1141

 Hiding, 3-563

 Input/Output, 2-37

 Last active, 3-1136

 Maximized, 3-1139

 Memo field editing, 3-1016

 Minimizable, defining, 3-434

 Minimized, 3-1140

 Mouse pointer, 3-629, 3-687 - 3-688

 Moving, 3-682

 Names, 1-14 - 1-15

 Number of rows, 3-1145

 Output to, 3-905, 3-917

 Parent, 3-1143

 Releasing, 3-810

 Restoring, 3-827 - 3-828

 Saving images, 3-845

 Scrolling, 3-856

 Shadows behind, 3-990

 Titles, 3-1146

 Trace, 3-910, 3-921, 3-999, 3-1010

 Typographic conventions, 1-24

 User-defined, 3-893, 3-896

 View, 3-1015

 Visible, 3-1148

 Zooming, 3-1152

Windows-style status bar, 3-997

WLAST(), 3-1136

WLROW(), 3-1138

WMAXIMUM(), 3-1139

WMINIMUM(), 3-1140

Word wrap, 4-59

Work areas

 Activating with SELECT, 3-860

 Alias, 1-21

 Current, 3-861

 Defined, 1-16

 Referencing, 1-23

 Returning alias, 3-189

 Table, 1-21

 Table names, 3-356

WOUTPUT(), 3-1142

WPARENT(), 3-1143

_WRAP, 4-59

Wrapping words, 4-59

WREAD(), 3-1144

Writing Character strings, 3-528, 3-541

WROWS(), 3-1145

WTITLE(), 3-1146

WVISIBLE(), 3-1148

Y

Year format, 3-887

YEAR(), 3-1150

Z

ZAP, 3-1151

Zeros, leading, 3-33

ZOOM WINDOW, 3-1152