Content

CH - 1	Introduction
CH - 2	Generating and Operatomg Database
CH - 3	Managing Database
CH - 4	Multiple File Handling, Functions and Arrays
CH - 5	Getting started with Programming
CH - 6	Advanced Techniques in Programming

ATMIYA

01. Index

- 0101. Introduction to DBMS and RDBMS
- 0102. Importance of DB and DBMS
- 0103. Database Models
- 0104. Data and File Concept
- 0105. Difference DBMS vs RDBMS
- 0107. Special Feature of FoxPro
- 0109. Field types
- 0110. Operators

ATMIYA

02. Index

- 0201. Creating Tables
- 0202. Opening and Closing Table
- 0203. List
- 0204. Display
- 0205. Goto
- 0206. Skip
- 0207. Edit
- 0208. Append
- 0209. Browse
- 0210. Delete
- 0211. Recall
- 0212. Pack
- 0213. Zap
- 0214. Replace
- 0215. Sum
- 0216. Total
- 0217. Average
- 0218. Change

ATMIYA

03. Index

- 0301. Sorting
- 0302. Indexing
- 0303. Locate
- 0304. Find
- 0305 Seek

04. Index

- 0401. Introduction to Work Area.
- 0402. Select Statement
- 0403. Set Relation
- 0404. Memory Variables
- 0405. Copy To
- 0406. Append From
- 0407. Copy Structure
- 0408. Join
- 0409. Update
- 0410. Dos Commands
- 0411. Scatter and Gether
- 0412. Function Index

05. Index

- 0501. ? / ?? / ???
- 0502. Input
- 0503. Accept
- 0504. @ … Say
- 0505. @ … Get
- 0506. Read
- 0507. If … EndIf
- 0508. Do Case … EndCase
- 0509. Do While … EndDo
- 0510. For … EndFor
- 0511. Scan … EndScan
- 0512. User Define Function (UDF)
- 0513. Procedures
- 0514. Return
- 0515. Set Commands

06. Index

- 0604. @ ... Box
- 0605. @ ... Clear
- 0606. @ ... Get
- 0607. @ ... Get - Check Box
- 0608. @ ... Get - Radio Button
- 0609. @ ... Get Push Button
- 0610. @ ... Get List
- 0611. @ ... Get Popup
- 0612. @ ... Menu
- 0613. @ ... Prompt
- 0614. Define Pad
- 0615. Define Bar
- 0616. Define Window
- 0617 Activate Window
- 0618. Deactivate Window
- 0619. Hide Window
- 0620. Show Window
- 0621. Release Window

0501. ? / ?? / ???

Syntax

? | ?? <expr1> [PICTURE <expC1>] |

[FUNCTION <expC2>] [AT <expN1>]

[FONT <expC3> [,<expN2>]]

[STYLE <expC4> | <expr2>]]

? <expr1>

A single question mark sends a carriage return and line feed. The results are displayed on the next line. If the expressions are omitted, a blank line is printed.

?? <expr1>

Two question marks display the expression results on the current line at the current position.

? | ?? <expr1> [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. For a list of those codes, see @ ... SAY.

[FUNCTION <expC2>]

The FUNCTION character expression <expC2> offers another method of including function codes in ? and ?? output. 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.

[FONT <expC3> [,<expN2>]]

The character expression <expC3> is the name of the font, and the numeric expression <expN2> is the font size.

? DATE( ) FONT 'ROMAN',16

[STYLE <expC4> | <expr2>]]

Include the STYLE clause to specify a font style for ? | ?? output.

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'

???

??? 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.

ATMIYA

0502. Input

Syntax

INPUT [<expC>] TO <memvar>

Inputs data from the keyboard into a system 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 system memory variables.

<expC>

Include <expC> to display a prompting message. <expC> is the message you want to display.

<memvar>

<memvar> is the system memory variable or an array element into which you want to input data entered from the keyboard. If <memvar> is a system memory variable that doesn't exist, INPUT creates it.

The expression input from the keyboard determines the type of system memory variable or array element created. If you input a numeric value, a numeric system memory variable or array element is created; if you input a character value, a character system 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.

Example:

INPUT to mnum_exp

? mnum_exp

INPUT 'Enter company: ' TO mcompany

? mcompany

0503. Accept

Syntax

ACCEPT [<expC>] TO <memvar>

Accepts character string data from the screen. This command allows you to enter character data directly into a memory variable or array element without delimiting the characters with quotation marks.

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.

<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.

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

?
```

? mcustname

0504. @ ... Say

Syntax

@ <row, column> SAY <expr>

[FUNCTION <expC1>] [PICTURE <expC2>]

[SIZE <expN1>, <expN2>]

[FONT <expC3>[, <expN3]] [STYLE <expC4>]

[COLOR SCHEME <expN4> | COLOR <color pair list>]

Displays output at a specified row and column position. Use this command to display formatted output on the desktop in FoxPro. It can also be used to format output for a printer.

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.

@ <row, column>

@ 5,5 && cursor position comes on row 5, column 5

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. Rows are numbered from top to bottom. The first column is number 0 on the desktop. Columns are numbered from left to right.

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.

@ <row, column> SAY <expr>

<expr> is evaluated and displayed or printed starting at <row, column>.

@ 5, 5 SAY 'Jignesh Dhol' && prints data on specified position of desktop

@ <row, column> SAY <expr> [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.

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.

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	rims leading and trailing blanks from <memvar> or <field>.
TZ	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.
$	CURRENCY. If CURRENCY is SET LEFT, the $ function code cannot be used. Use with numeric data only.

PICTURE

A PICTURE expression can include any characters, but only the characters listed below actively participate in display and editing.

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.

$	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.
.	A decimal point specifies the decimal point position.
,	A comma is used to separate digits to the left of the decimal point.

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$$'
```

[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]] [STYLE <expC4>]

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.

The FONT clause is ignored in FoxPro for MS-DOS.

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 SET COLOR Commands Overview or Color Table by Color Pair.

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.

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)

For 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
```

ATMIYA

0505. @ ... Get

Syntax

@ <row, column> GET <memvar> | <field>

[FUNCTION <expC1>] [PICTURE <expC2>]

[FONT <expC3>[, <expN1>]] [STYLE <expC4>]

[DEFAULT <expr1>] [ENABLE | DISABLE]

[MESSAGE <expC5>]

[RANGE [<expr2>] [, <expr3>]]

[SIZE <expN2>, <expN3>]

[VALID <expL1> | <expN4> [ERROR <expC6>]]

[WHEN <expL2>]

[COLOR SCHEME <expN5> | COLOR <color pair list>]

Use @ ... GET 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.

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. Rows are numbered from top to bottom. The first column is number 0 on the desktop. 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. A position in the main FoxPro window or in a user-defined window is determined by the font. you can use decimal fractions for row and column coordinates.

<memvar> | <field>

@ ... GET creates an editing region for the memory variable or array element specified in <memvar> or the field specified in <field>.

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

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.
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	rims leading and trailing blanks from <memvar> or <field>.
TZ	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.
$	CURRENCY. If CURRENCY is SET LEFT, the $ function code cannot be used. Use with numeric data only.

PICTURE

A PICTURE expression can include any characters, but only the characters listed below actively participate in display and editing.

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.
$	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.
.	A decimal point specifies the decimal point position.
,	A comma is used to separate digits to the left of the decimal point.

FONT <expC3>[, <expN1>]

The character expression <expC3> is the name of the font and the numeric expression <expN1> is the font size.

@ 2, 2 GET eBooK FONT 'ROMAN',16

READ

If you include the FONT clause but omit the font size <expN1>, a 10 point font is used.

STYLE <expC4>

You can include the STYLE clause to specify a font style for an @ ... GET editing region.

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.

@ 2, 2 GET eBooK STYLE 'BI'

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.

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.

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.

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.

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.

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>.

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.

<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.

<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. When <expN4> is positive, <expN4> specifies the number of objects to advance. When <expN4> is negative, <expN4> specifies the number of objects to move back.

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>

Including the number of an existing color scheme in the COLOR SCHEME clause or a set of color pairs in the COLOR clause can specify the color of the rectangular area. For example, this color pair specifies a red foreground on a white background: R/W

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)

0506. Read

Syntax

READ [CYCLE] [SHOW <expL3>] [NOMOUSE]

[OBJECT <expN2>] [TIMEOUT <expN3>]

The read command allows you to enter data in a field or memary variable. READ is used most commonly in foxpro programming to allow for input of data. Several @ ... GET commands like check boxes, lists, popups, push buttons, radio buttons, spinners and text are used to position the input areas on the screen, then the READ commands allows you to enter data in fields or memary variables.

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.

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.

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.

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.

```
STORE 1 TO eBooK

STORE SPACE(10) TO name

CLEAR

@ 2,2 SAY 'Enter a name: ' GET name

@ 4,2 GET eBooK PICTURE '@*R Foxpro;Oracle;Visual Basic'

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.

0507. If ... EndIf

Simple If

```
if (condition)
    statement in block
endif
```

If . . . Else

```
if (condition)
    statement in block
else
    statement in block
endif
```

Nested If . . . Else

```
if (condition)
    if (condition)
        statement in block
    else
        statement in block
    endif
else
    statement in block
```

ATMIYA

```
endif
```

Else . . . If Ladder

```
if (condition)
    statement in block
else
    if (condition)
        statement in block
    else
        if (condition)
            statement in block
        else
            statement in block
        endif
    endif
endif
```

0508. Do Case ... EndCase

DO CASE

CASE <expL1>

<statements>

[CASE <expL2>

<statements>

...

CASE <expLN>

<statements>]

[OTHERWISE

<statements>]

ENDCASE

Example

Store 0 to ch

@ 5,5 say "Mango"

@ 6,5 say "Apple"

@ 7,5 say "Enter your Choice&ldots;: " get ch

read

do case

ATMIYA

```
    case ch = 1
        @ 15,10 say "You select Mango"
    case ch = 2
        @ 15,10 say "You select Apple"
    otherwise
        @ 15,10 say "Invalid Choice"
endcase
```

0509. Do While ... EndDo

DO WHILE <expL>

<statements>

[LOOP]

[EXIT]

ENDDO

Example: 1

```
CLEAR
store 1 to a
do while a<10
    ?a
    a=a+1
enddo
```

Example: 2

```
CLEAR
store 1 to a
store .T. to xFlag
do while xFlag=.T.
    ?a
    a=a+1
```

ATMIYA

```
    if a=10
        xFlag=.F.
    endif
enddo
```

0510. For ... EndFor

FOR <memvar> = <expN1> TO <expN2> [STEP <expN3>]

<statements in block>

[EXIT]

[LOOP]

ENDFOR | NEXT

Example: 1

```
CLOSE DATABASES
CLEAR
FOR mcount = 1 TO 10
    ? mcount
ENDFOR
```

ATMIYA

0511. Scan ... EndScan

Syntax:

SCAN

[<scope>] [FOR <expL1>] [WHILE <expL2>]

[<statements>] [LOOP] [EXIT]

ENDSCAN

This command moves the record pointer through the current table and executes a block of commands for each record that meets the specified conditions. It automatically advances the record pointer to the next record that meets the specified conditions and executes the block of commands.

You can place comments after ENDSCAN on the same line. The comments are ignored during program compilation and execution.

<scope>

The scope clauses are: ALL, NEXT <expN>, RECORD <expN>, and REST. 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.

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.

ATMIYA

0512. User Define Function (UDF)

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).

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. 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.

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. 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:

```
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.

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 EXAMPLE

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.

In this 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
```

0513. Procedures

Syntax

PROCEDURE <procedure name>

In many programs, certain routines are frequently repeated. Define these commonly-used routines as separate procedures to reduce program size and complexity and easy for 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({11/18/75})

PROCEDURE longdate
PARAMETER mdate
RETURN CDOW(mdate) + ', ' + MDY(mdate)
```

OUTPUT :

Tuesday, November 18, 1975

ATMIYA

0514. Return

Syntax

RETURN [<expr> | TO MASTER | TO <program name>]

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.

<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>.

0515. Set Commands

SET ALTERNATE on/OFF	Enables or disables output to an Alternate file.
SET ALTERNATE TO	Creates a text file with default extension TXT for storing screen or printer output of most commands, except full-screen commands.
SET ANSI on/OFF	Specify -how comparison between strings of different lengths are made with the = operator with SQL commands.
SET AUTOSAVE on/OFF	Determines whether or not FoxPro flushes data buffers to disk on termination of READ, or the program returns to the command window.
SET BELL ON/off	Enables or disables sounding of bell during editing.
SET BELL TO	Sets the bell attribut&, i.e. frequency and duration.
SET BLINK ON/off	Determines whether or not the blinking or high-density attributes can be specified. (Applicable only for EGA and VGA monitors on FoxPro for DOS)
SET BLOCKSIZE	Specifies how FoxPro allocates disk space for the storage of memo fields.
SET BORDER	Defines a border for popups created with DEFINE POPUP and for windows created with DEFINE WINDOW. Border can be of various types - single- or double-line border, panel, none, etc.
SET BRSTATUS on/OFF	Enables or disables the display of the status bar in a Browse window. Status bar displays information, such as the current drive, database file, number of records and state of special keys.
SET CARRY on/OFF	Determines whether or not FoxPro carries data forward from the current record to the new record created with APPEND or INSERT.

SET CARRY TO	Specifies the fields from which the.field data is carried forward to a new record.
SET CENTURY on/OFF	Determines whether or not the century portim-of date expressions is displayed.
SET CLEAR off/ON	Determines whether or not FoxPro clears the screen on issuing SET FORMAT or QUIT.
SET CLOCK on/OFF/status	Determines whether or n-ol FoxPro displays the system clock. (STATUS - FoxPro for Windows only)
SET CLOCK TO	Specifies the clock location on the screen.
SET COLOR OF	Specifies the colours of user-defined menu systems and windows.
SET COLOR OF SCHEME	Specifies the colours of a color scheme or copies one color scheme to another color scheme.
SET COLOR SET	Loads colours from a a previously defined color set.
SET COLOR TO	Specifies the colours of user-defined menus and windows. it affects colour scheme 1 and 2.
SET COMPATIBLE on/OFF	Controls compatibility with FoxBASE+ and other Xbase languages. It affects response of FoxPro to comm3nds and functions, such as @ ... SAY .. GET, READ, LIKE(, PLAY MACRO, SELECT(, SET PRINT, etc.
SET CONFIRM on/OFF	Specifies whether or not Enter or Tab must be pressed to exit an input field and move to the next object.
SET CONSOLE ON/off	Enables or disables output to the screen or a window.
SET CURRENCY TO	Defines the currency symbol.
SET CURRENCY LEFT/right	Specifies the position of the currency symbol.
SET CURSOR ON/off	Determines whether the cursor is displayed during a pending @ ... GET.

SET DATE	Sets the format used to display date expressions. One can choose a date format, such as American, British, French, German, Italian, Japan, etc.
SET DEBUG ON/off	Enebles or disables menu access to the Debug and Trace -windows. If DEBUG is set to off, you cannot open these windows.
SET DECIMALS	Specifies the minimum number of decimal places displayed in the result of mathematical calculations.
SET DEFAULT TO	Specifies the default drive and directory for various file read and write operations.
SET DELETED on/OFF	Specifies whether or ftot FoxPro processes records marked for deletion. When set to ON, records marked for deletions are not included with commands, such as LIST, COPY, REPORT, etc.
SET DELIMITERS on/OFF	When set to ON, fields displayed with @..GET are enclosed within delimiter character(s).
SET DELIMITERS TO	Specifies the delimiter character(s). The default delimiter character is ":"
SET DEVELOPMENT ON/off	When DEVELOPMENT is ON (the default value), FoxPro compare the creation date and time of a program with those of its compiled object file when you run a program. If the source program is more current than the. compiled program, it recompiles the program before execution.
SET DEVICE TO SCREEN/printer/file	Directs output of @ ..I. SAY commands to the specified device (i.e. screen, printer, or file).
SET DISPLAY TO	Selects the specified display mode. The various display modes available to choose from include, CGA, COLOR, EGA25, EGA43, MONO, VGA25 and VGA50.
SET DOHISTORY on/OFF	Determines whether or not commands executed from a program are placed into the command window.

SET ECHO on/OFF	Activates the Trace window for program debugging. Breakpoints can also be set within the Trace window to suspend the program at a particular command.
SET ESCAPE ON/off	Determines whether or not the execution of a command or program interrupts on pressing the Esc key.
SET EXACT on/OFF	Specifies the rules used when comparing strings of different lengths. When set to ON, the expressions must match character for character, including blank spaces, for them to be equal.
SET EXCLUSIVE ON/off	Specifies whether database files are opened for exclusive or shared use on a network.
SET FIELDS on/OFF	Specifies whether all or only selected fields (as specified with SET FIELDS TO) in a database file are used with commands. With SET FIELDS OFF, all fields are accessible.
SET 'FIELDS TO	Specifies the field list that are accessible when FIELDS is set to ON.
SET FILTER	Specifies a condition for the current database file. This condition is automatically applied to all commands that use the database file.
SET FIXED on/OFF	Specifies whether or not the number of decimals places used in the display of numeric data is fixed. When set to ON, a fixed number of decimal places, as specified with SET DECIMALS are displayed for all numeric outputs.
SET FORMAT	Specifies a custom screen format file for use with APPEND, BROWSE, INSERT, etc. instead of the standard format.
SET FULLPATH ON/off	Specifies whether or not CDX(, DBF(, IDX(and NDX(return the path in a file name.
SET FUNCTION	Assigns a new value to a function key. Besides programming functioiis keys, you can also program combinations of functions keys with other keys, such as Ctrl, Shift, etc.

SET HEADING ON/off	Determines whether or not column headings (e.g. field names) are displayed for fields with commands, such as LIST, DISPLAY, AVERAGE, etc.
SET HELP ON/off	Enables or disables the FoxPro online help facility.
SET HELP TO	Specifies a new help file for use with the FoxPro online help facility.
SET HELPFILTER	Enables FoxPro to display a subset of help topics in the Help window. For example, you can set HELPFILTER to display only File Management or Printing commands.
SET HOURS	Sets the system clock to a 12- or 24-hour time, format.
SET INDEX	Opens one or more index files for use with the current database file. You can open both IDX and CDX files with SET INDEX,
SET INTENSITY ON/off	Determines whether or not FoxPro uses the enhanced screen colour attribute for the display of editing fields. On monochrome monitors, the editing fields are displayed in reverse video with INTENSITY set to ON.
SET KEYCOMP TO DOSIWINDOWS	Controls FoxPro keystroke navigation on different platforms. (FoxPro for Windows only)
SET LIBRARY	Opens an external Application Program Interface (API) routine library file to extend the capabilities of FoxPro and its user interface.
SET LOCK on/OFF	Enables or disables automatic file locking in certain commands, such as AVERAGE, CALCULATE, COPY, LIST, SORT, etc.
SET LOGERRORS ON/off	Determines whether or not FoxPro sends compilation error messages to a text file. When set to ON, the compilation errors are stored in a log file with the same name as the compiled program with extension ERR.

SET MACKEY	Specifies a key or key combination that displays the Macros dialog box. By default, it is Shift+FIO.
SET MARGIN	Specifies the left margin in printed output. For instance, SET MARGIN TO 1 0 leaves a margin of 1 0 characters on the left side of all printouts.
SET MARK OF	Specifies a menu pad or popup option mark character. The mark character is displayed on the left of the menu option. There is a separate command for setting mark character for Menu, Pad, Popup and Bar. Mark characters cannot be changed in FoxPro for Windows
SET MARK TO	Specifies a delimiter to use in the display of date expressions, e.g. SET MARK TO "-". The default delimiter is forward slash (1).
SET MEMOWIDTH	Specifies the displayed, width (in columns) of memo fields and character expressions. It affects output of commands, such as LIST, ?/??, DISPLAY, etc.
SET MESSAGE TO <expc>	Specifies a message for display under the status bar (when status bar is on).
SET MESSAGE TO <expn>	Specifies the row location where user-defined messages are displayed.
SET MESSAGE WINDOW	Defines a window in which the user-defined messages are displayed.
SET MOUSE ON/off	Enables or disables the mouse. (FoxPro for DOS only.)
SET MOUSE TO	Specifies the sensitivity of mouse in the range of 1 through 10 (default 5). (FoxPro for DOS only.)
SET MULTILOCKS on/OFF	Determines whether or not multiple records can be locked with LOCK( ) or RLOCK( ).
SET NEAR on/OFF	Determines where the record pointer is positioned after FIND or SEEK unsuccessfully searches fora record. When set to ON, it is positioned on the closest record. When set to OFF, it, is positioned at the end of file, i.e. EOF(returns T).

SET NOTIFY ON/off	Enables or disables the display of certain system messages. When set to OFF, a few system messages are not displayed.
SET ODOMETER	Determines the interval at which the record counter is updated in response to certain commands, such as INDEX, SORT, COPY. The default value is 100.
SET OPTIMIZE ON/off	Enables or disables Rushmore optimization. It affects commands,, such as BROWSE, CHANGE, COPY TO, EDIT, LIST, SORT, REPORT, TOTAL, etc.
SET ORDER	Selects a specified index file or tag as the controlling (master) index file/tag for the current or specified database file. Optionally, the index order, i.e. ascending or descending can also be specified.
SET PALETTE ON/off	Specifies whether or not the FoxPro color palette is used for displaying BMP pictures and OLE objects. (FoxPro for Windows only)
SET PATH	Specifies a path (a set of directories) for file searches. FoxPro searches for files in the path if it cannot find a file in the default directory.
SET PDSETUP	Loads a printer driver s@ etup or clears the current printer driver setup.
SET POINT	Determines the decimal point character used in the display of numericexpressions.Thedefault decimal point character is a period
SET PRINTER on/OFF	Enables or disables output to the printer However, the output of @..SAY is not routed to printer.
SET PRINTER TO	Print output is routed to the specified file or port. It can also send output to a network printer.
SET PROCEDURE	Opens the named procedure file. Only one procedure file can be open at a time.
SET READBORDER on/OFF	Determines if borders are placed around editing regions created with @ ... GET. (FoxPro for Windows only)

SET REFRESH	Determines whether or not the screen displays changes made to records by other users on the network while viewing records in the Browse window.
SET RELATION	Establishes relation between two or more open database files on a common field. Can also be used to clear relations.
SET RELATION OFF	Breaks an established relationship between the current database file and another open database file.
SET REPROCESS	Specifies how many times or for how long FoxPro attempts to lock a file or record after an unsuccessful locking attempt.
SET RESOURCE ON/off	When set to on, any change made to the FoxPro environment is saved in the resource file. Changes are not saved when RESOURCE is OFF.
SET RESOURCE TO	By default, FoxPro uses FOXUSER.dbf as the resource file. This con mand can be used to specify any other file as the resource file.
SET SAFETY ON/Off	Determines whether or not FoxPro'displays a warning before overwriting an existing file.
SET SCOREBOARD on/OFF	When set to ON, FoxPro displays the status of the NumLock, CapsLock, and Insert keys in row zero when the status bar is off. When both scoreboard and status bar are off, the status of these keyr is not displayed (FoxPro for DOS only.)
SET SEPARATOR	Specifies the character that separates each group of three digits to the left of the decimal point. Used with @..SAY PICTURE command. The default separator character is comma
SET SHADOWS ON/off	Globally displays or removes shadows from all windows and po-, pups. (FoxPro for DOS only.)
SET SKIP TO	Creates one-to-many Relationships between database files open in different work areas.

SET SKIP OF MENU	Enables or disables a menu bar based on a specified condition.
SET SKIP OF PAD	Enables or disables a menu pad based on a specified condition.
SET SKIP OF POPUP	Enables or disables a popup based on a specified condition.
SET SKIP OF BAR	Enables or disables a popup option (bar) based on a specified condition.
SET SPACE ON/off	Determines whether or not a space is displayed between fields or expressions with the ? or ?? command.
SET STATUS on/OFF	Displays or removes the status bar. The status bar displays information, such as current drive, database file, number of records, status of special keys, etc.
SET STATUS BAR ON/off	Displays or removes a Windows-style status bar in FoxPro for Windows. (FoxPro for Windows only)
SET STEP On/OFF	Opens the Trace window and suspends the program. Useful for debugging of programs.
SET STICKY ON/off	Specifies how the mouse displays menus in the FoxPro menu system. (FoxPro for DOS only.)
SET SYSMENU ON/off	automatic Enables or disables access to the FoxPro system menu bar during program execution.
SET TALK ON/off	Determines whether or not FoxPro displays response to certain commands on the screen or a window.
SET TEXTMERGE on/OFF	Enables or disables the evaluation of fields, functions and expressions that are surrounded by text merge delimiters.
SET TEXTMERGE DELIMITERS	Specifies the text merge delimiters. The default text merge characters are double angle brackets.
SET TOPIC	Specifies the initial help topic that is displayed on invoking the FoxPro help system.

SET TRBETWEEN ON/off	Enables or disables tracing between breakpoints in the Trace window.
SET TYPEAHEAD	Specifies the maximum size of the typeahead buffer, i.e. the number of characters that can be stored in the typeahead buffer.
SET UDFPARMS TO VALUE/reference	Specifies whether FoxPro passes-parameters to a user-defined function (UDF) by value or by reference.
SET UNIQUE on/OFF	Specifies whether or not records with duplicate index key values appear in an index file. For instance, a file indexed on the NAME field will not have two records with "Ritu Shartna" when UNIQUE is ON.
SET VIEW on/OFF	Opens or closes the View window that provides an easy way to open database files, establish relations, etc.
SET VIEW TO	Restores the FoxPro environment from a specified view file.
SET WINDOW OF MEMO	Specifies a window in which memo fields are edited.
SET(	Returns the status of a specified SET command. For example, ? SET("TALK") displays ON or OFF depending on the previous SET TALK command.

0604. @ ... Box

Syntax

@ <row1, column1>, <row2, column2> BOX [<expC>]

Draws a box using specified coordinates given in bracket.

[<expC>]

This command draws a box in the main FoxPro window in FoxPro for Windows, or in a user-defined window.

<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.

Example:

@ 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)
```

0605. @ ... Clear

Syntax

@ <row1,column1> [CLEAR | CLEAR TO <row2,column2>]

Clears a portion of the screen in FoxPro.

<row1, column1>

These are the coordinates of the upper-left corner of the area to clear.

<row2, column2>

These are the coordinates of the lower-right corner of the area to clear.

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, 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.

TO CLEAR ALL SCREEN

@ 1, 1, CLEAR TO 24, 80 OR @ 1, 1, CLEAR 24, 80

TO CLEAR HALF SCREEN

@ 1, 1, CLEAR TO 12, 40 OR @ 1, 1, CLEAR 12, 40

0606. @ ... Get

To refer this topic CLICK ME.

0607. @ ... Get - Check Box

Syntax

@ <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>]

Creates a check box or picture check box.

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.

In FoxPro for Windows, pressing the Spacebar or Enter or clicking the check box toggles the check box from one state to the other. Press Enter to move to the next object.

<row, column>

These arguments are useful for positioning check box control on specified screen.

GET <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.

FUNCTION <expC1> | PICTURE <expC2>

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:

```
STORE 1 TO CHOICE

@ 5,5 GET CHOICE PICTURE '@*C eBookMark'

READ
```

(or)

```
STORE 1 TO CHOICE

@ 5,5 GET CHOICE FUNCTION '*C' PICTURE 'eBookMark'
```

READ

PICTURE and FUNCTION Options

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.

Hot Keys

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 an underlined character for FoxPro Windows.

```
STORE 1 TO CHOICE;

@ 5,5 GET CHOICE PICTURE '@*C e\<BookMark'

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 CHOICE

@ 5,5 GET CHOICE PICTURE '@*C \\eBookMark' (OR)

@ 5,5 GET CHOICE PICTURE '@*C eBookMark'

DISABLE

READ
```

ATMIYA

FONT <expC3> [, <expN1>]

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.

```
STORE 1 TO CHOICE

@ 5,5 GET CHOICE FUNCTION '*C' PICTURE 'eBookMark'

FONT 'VERDANA',12

READ
```

STYLE <expC5>

The STYLE clause can be used with @&ldots;GET check box command to specify a font style. Most commonly used Font Style are described as follows:

Character	Font Style
B	Bold
I	Italic

N	Normal
O	Outline
Q	Opaque
S	Shadow
-	Strikeout
T	Transparent
U	Underline

```
CLEAR

STORE 1 TO CHOICE

@ 5,5 GET CHOICE FUNCTION '*C eBookMark' STYLE 'UBI'

READ
```

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.

```
CLEAR

STORE 1 TO CHOICE

@ 5,5 GET Choice FUNCTION '*C eBookMark' DEFAULT .T.

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.

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 ENABLE to enable a disabled check box.

MESSAGE <expC5>

The MESSAGE clause can be used with @ &ldots; GET check box command to display the character expression when the check box is selected.

VALID <expL1> | <expN4>

The VALID clause allows you to include an optional VALID expression that is evaluated when check box is chosen. Note that the VALID clause is not evaluated when you move to the check box.

ATMIYA

WHEN <expL2>

You can also use WHEN clause to allow/does not allows access to a check box based on the logical value you have specified. If it is true (.T.), the check box can be accessed. If it is False (.F.), the check box can not be accessed.

COLOR SCHEME <expN5> | COLOR <color pair list>

Including the number of an existing color scheme in the COLOR SCHEME clause or a set of color pairs in the COLOR clause can specify the color of the rectangular area. For example, this color pair specifies a red foreground on a white background: R/W

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)

0608. @ ... Get - Radio Button

Syntax

@ <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>]

Creates a set of radio buttons or picture radio buttons.

This variation of @ ... GET creates a set of radio buttons. As their name suggests, radio buttons are similar to the buttons of a car radiochoosing 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.

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.

<row, column>

The first button in a set of radio buttons is placed at the location designated by <row, column>.

GET <memvar> | <field>

When you choose a radio button, your choice is stored to the memory variable or array element.

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.

STORE 1 TO mchoice

@ 2,2 GET mchoice FUNCTION '*R None;Single;Double'

or

@ 2,2 GET mchoice PICTURE '@*R None;Single;Double'

or

@ 2,2 GET mchoice FUNCTION '*R' PICTURE ' None;Single;Double'

or

@ 2,2 GET mchoice FUNCTION '*R None;Single' PICTURE ';Double'

READ

Radio Buttons with Picture Prompts

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, followed by a space and the .BMP file name. If the .BMP file is not located in the default directory, include the directory with the .BMP file name.

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 the box is chosen. This is the default behavior.
T	Terminate the READ when the box 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:

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.

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.

STORE 1 TO mchoice

@ 2,2 GET mchoice FUNCTION '*R' PICTURE 'None;\\Single;Double'

READ

FONT <expC3>[, <expN1>]

Include FONT to specify a font and font size for the radio button prompt. The character expression <expC3> is the name of the font, and the numeric expression <expN1> is the font size.

STORE 1 TO CHOICE

@ 2,2 GET mchoice FUNCTION '*R' PICTURE '\<None;\<Single;\<Double' FONT 'VERDANA',12

READ

STYLE <expC5>

The STYLE clause can be used with @&ldots;GET radio button command to specify a font style. Most commonly used Font Style are described as follows:

Character	Font Style
B	Bold

I	Italic
N	Normal
O	Outline
Q	Opaque
S	Shadow
-	Strikeout
T	Transparent
U	Underline

CLEAR

STORE 1 TO CHOICE

@ 2,2 GET mchoice FUNCTION '*R' PICTURE '\<None;\<Single;\<Double' STYLE 'UBI'

READ

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.

CLEAR

STORE 1 TO CHOICE

@ 2,2 GET mchoice FUNCTION '*R' PICTURE 'None;Single;Double' DEFAULT 'Single'

or

@ 2,2 GET mchoice FUNCTION '* R' PICTURE '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).

The numeric expression <expN2> specifies the height of a Radio Button. 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.

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 ENABLE 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.

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.

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.

When <expN5> is negative, <expN5> indicates the number of objects to move back.

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>

Including the number of an existing color scheme in the COLOR SCHEME clause or a set of color pairs in the COLOR clause can specify the color of the rectangular area. For example, this color pair specifies a red foreground on a

white background: R/W

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)

0609. @ ... Get Push Button

Syntax

@ <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>]

Creates a set of push buttons or picture push buttons.

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.

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.

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.

<row, column>

The <row, column> is assigned position of the control.

<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.

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.

FUNCTION clause only:

STORE 1 TO mchoice

@ 2,2 GET mchoice FUNCTION '* OK;Cancel' SIZE 1, 8

or

@ 2,2 GET mchoice PICTURE '@* OK;Cancel' SIZE 1, 8

or

@ 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.

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, followed by a space and the .BMP file name. If the .BMP file is not located in the default directory, include the directory with the .BMP file name.

PICTURE and FUNCTION Options

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 the box is chosen. This is the default behavior.
T	Terminate the READ when the box 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.

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.

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.

```
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.

```
STORE 1 TO mchoice

@ 2,2 GET mchoice FUNCTION '* \!OK;Cancel'

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.

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.

To create an escape push button, place a backslash and a question mark (\?) before the push button's prompt.

STORE 1 TO mchoice

@ 2,2 GET mchoice FUNCTION '* OK;\?Cancel'

READ

FONT <expC3>[, <expN1>]

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.

STYLE <expC4>

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.

Character	Font Style
B	Bold
I	Italic
N	Normal
O	Outline
Q	Opaque
S	Shadow
-	Strikeout
T	Transparent
U	Underline

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.

@ 2,2 GET mchoice FUNCTION '* OK;Cancel' DEFAULT 'OK'

or

@ 2,2 GET mchoice FUNCTION '* OK;Cancel' DEFAULT 2

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.

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 ENABLE 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 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.

A VALID clause that includes a numeric expression is used to specify which object is activated after a push button is chosen.

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.

When <expN5> is negative, <expN5> indicates the number of objects to move back.

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>

Including the number of an existing color scheme in the COLOR SCHEME clause or a set of color pairs in the COLOR clause can specify the color of the rectangular area. For example, this color pair specifies a red foreground on a white background: R/W

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)

0610. @ ... Get List

Syntax

@ <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>]

This command 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.

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 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.

<row, column>

The <row, column> is assigned position of the control.

<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.

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.

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.

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.

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.

FONT <expC3> [, <expN3>]

The character expression <expC3> is the name of the font, and the numeric expression <expN3> is the font size.

STYLE <expC4>

In FoxPro for Windows, include the STYLE clause to specify a font style for the items in the list.

Character	Font Style
B	Bold
I	Italic
N	Normal
O	Outline
Q	Opaque
S	Shadow
-	Strikeout
T	Transparent
U	Underline

DEFAULT <expr>

When you choose an item from the list, your choice is saved in a memory variable, array element or field you specified.

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.

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 ENABLE to enable a disabled list.

MESSAGE <expC5>

The MESSAGE clause character expression <expC5> is displayed when a list 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.

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. When <expN6> is negative, <expN6> specifies the number of objects to move back.

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 <expN6> | COLOR <color pair list>

Including the number of an existing color scheme in the COLOR SCHEME clause or a set of color pairs in the COLOR clause can specify the color of the rectangular area. For example, this color pair specifies a red foreground on a white background: R/W

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)

0611. @ ... Get Popup

Syntax

@ <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>]

This commadnd 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 popupa caret (^).

To open a popup press the Spacebar, Alt+Up Arrow, Alt+Down Arrow or click on the popup.

ATMIYA

<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. 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>.

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.

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 PICTURE character expression <expC2> uses the same syntax as the FUNCTION character expression except the PICTURE expression must begin with @ followed by ^.

STORE 1 TO eBook

@ 2,2 GET eBook FUNCTION '^ FoxPro;Oracle;VIsualBasic'

or

@ 2,2 GET eBook PICTURE '@^ FoxPro;Oracle;VIsualBasic'

or

@ 2,2 GET eBook PICTURE '@^' + ' FoxPro;Oracle;VIsualBasic'

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 the box is chosen. This is the default behavior.
T	Terminate the READ when the box 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

To assign a hot key, place a backslash and a less than sign (\<) before the desired character of the popup option. The following example assigns the hot keys 'F' and 'r' to the Foxpro and Oracle options, respectively:

```
STORE 1 TO mchoice

@ 2,2 GET mchoice FUNCTION '^ \<Foxpro;O\<racle'

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.

FONT <expC3>[, <expN1>]

The character expression <expC3> is the name of the font, and the numeric expression <expN1> is the font size.

If you include the FONT clause but omit the font size <expN1>, a 10 point font is used.

STYLE <expC4>

In FoxPro for Windows, include the STYLE clause to specify a font style for the options in the popup. 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

DEFAULT <expr>

When you choose an option from a popup, your choice is saved in the memory variable, array element or field you specify.

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 eBook FUNCTION '^ Oracle;VB;C;Fox' DEFAULT 'Fox'

or

@ 2,2 GET eBook FUNCTION '^ Oracle;VB;C;Fox' DEFAULT 4

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 eBook_arr PICTURE '@^' ...

... FROM eBook_arr FUNCTION '^' ...

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.

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.

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>.

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.

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.

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. When <expN6> is negative, <expN6> specifies the number of objects to move back.

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> | COLOR <color pair list>

Including the number of an existing color scheme in the COLOR SCHEME clause or a set of color pairs in the COLOR clause can specify the color of the rectangular area. For example, this color pair specifies a red foreground on a white background: R/W

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)

0612. @ ... Menu

Syntax

@ <row, column> MENU <array>, <expN1>

[, <expN2>] [TITLE <expC>] [SHADOW]

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.

<row, column>

These are numeric expressions that specify where the top-left corner of the menu popup is placed on the desktop or a user-defined window.

<array>

This 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.

0613. @ ... Prompt

Syntax

@ <row, column> PROMPT <expC1> [MESSAGE <expC2>]

Using a series of @ ... PROMPTs you can create the menu bar pads and specify their positions on the desktop 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.

<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.

ATMIYA

0614. @ ... Pad

Syntax

DEFINE PAD <pad name> OF <menu name>

PROMPT <expC1> [AT <row, column>]

[BEFORE <pad name> | AFTER <pad name>]

[KEY <key label>[, <expC2>]]

[SKIP [FOR <expL>]] [MESSAGE <expC4>]

[COLOR SCHEME <expN> | COLOR <color pair list>]

This command 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.

<pad name>

This is the name of the menu pad you want to create.

OF <menu name>

This 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 eBook

DEFINE PAD fox OF eBook PROMPT "\<FoxPro"

DEFINE PAD oracle OF eBook PROMPT "Or\<acle"

ACTIVATE MENU eBook

The first pad, fox, is chosen by pressing F; the second pad, oracle, is chosen by pressing a.

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 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.

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.

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.

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>.

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.

Menus you create that include SKIP FOR expressions may not behave properly when the spellchecker or wizards are active.

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.

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 <expN4> | COLOR <color pair list>

Including the number of an existing color scheme in the COLOR SCHEME clause or a set of color pairs in the COLOR clause can specify the color of the rectangular area. For example, this color pair specifies a red foreground on a white background: R/W

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)

0615. @ ... Bar

Syntax

DEFINE BAR <expN1> 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>]

This command 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.

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.

<expN1>

<expN1> is the bar number of the popup option you create with

OF <popup name>

Specify the name of the popup on which the options are placed with <popup name>.

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 eBook

DEFINE BAR 1 OF receive PROMPT '\<FoxPro'

DEFINE BAR 2 OF receive PROMPT 'Or\<acle'

ACTIVATE POPUP eBook

The first popup option, BAR 1, is chosen by pressing F; the second option, BAR 2, is chosen by pressing a.

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>.

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.

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>.

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.

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 pop-up option by placing a backslash (\) before the text of the prompt. For example:

DEFINE BAR 1 OF eBook PROMPT '\eBookMark for C Language'

The first menu option, BAR 1, appears in disabled colors and cannot be selected or chosen.

COLOR SCHEME <expN4> | COLOR <color pair list>

Including the number of an existing color scheme in the COLOR SCHEME clause or a set of color pairs in the COLOR clause can specify the color of the rectangular area. For example, this color pair specifies a red foreground on a white background: R/W

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)

ATMIYA

0616. Define Window

Syntax

DEFINE WINDOW <window name1> FROM <row1, column1> TO

<row2, column2> | AT <row3, column3> SIZE <row4, column4>

[IN [WINDOW] <window name2>]

[FONT <expC1>[, <expN1>]]

[STYLE <expC2>]

[FOOTER <expC3>]

[TITLE <expC4>]

[HALFHEIGHT]

[DOUBLE | PANEL | NONE | SYSTEM | <border string>]

[CLOSE | NOCLOSE]

[FLOAT | NOFLOAT]

[GROW | NOGROW]

[MINIMIZE]

[ZOOM | NOZOOM]

[ICON FILE <expC5>]

[COLOR SCHEME <expN2> | COLOR <color pair list>]

Define window command is used to define a new named window. You also need to provide the top-left and bottom-right co-ordinates for the window.

<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>.

AT <row3, column3> SIZE <row4, column4>

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.

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.

FONT <expC1>[, <expN1>]

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.

STYLE <expC2>

In FoxPro for Windows, include the STYLE clause to specify a font style for output directed to the window. Windows determines the styles that are available for a font.

TITLE <expC4>

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

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.

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.

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. 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. 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. Include NOGROW or omit GROW to prevent a window's size from being changed.

MINIMIZE

A user-defined window can be minimized if you include MINIMIZE in DEFINE WINDOW.

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.

COLOR SCHEME <expN2> | COLOR <color pair list>

Including the number of an existing color scheme in the COLOR SCHEME clause or a set of color pairs in the COLOR clause can specify the color of user-defined windows. For example, this color pair specifies a red foreground on a white background: R/W 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:

```
DEFINE WINDOW eBookMark FROM 5,5 TO 25,75 TITLE
'eBookMark for FoxPro';

CLOSE FLOAT GROW ZOOM FONT "ROMAN",18 STYLE "BI"
COLOR B+/Rr

ACTIVATE WINDOW eBookMark

@ 2,2 SAY "eBookMark is nice concept to learn yourself"
```

0617 Activate Window

Syntax

ACTIVATE WINDOW [<window name1>[, <window name2>...]] | ALL

[IN [WINDOW] <window name3> | SCREEN]

[BOTTOM | TOP | SAME] [NOSHOW]

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

Activating a window makes it the front most windows 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.

<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.

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

Filer

Calculator

Calendar/Diary

Special Characters*

ASCII Chart*

Puzzle

System Windows

Command

Debug

Trace

View

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.

BOTTOM | TOP | SAME

By default, a window becomes the front most 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.

0618. Deactivate Window

Syntax

DEACTIVATE WINDOW <window name1>[, <window name2> ...] | ALL

Deactivates specific user-defined windows or FoxPro system windows and removes them from the screen but not from memory.

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.

<window name1>[,<window name2>...]

These clauses specify one or more windows to deactivate.

ALL

Including ALL deactivates all active windows.

0619. Hide Window

Syntax

HIDE WINDOW [<window name1>[, <window name2>] ...]
|

ALL [IN [WINDOW] <window nameN> | IN [WINDOW]

SCREEN] [BOTTOM | TOP | SAME] [SAVE]

Hides an active user-defined window or FoxPro system window. HIDE WINDOW removes a window or a set of windows from the desktop in FoxPro for window or a user-defined window. You can use HIDE WINDOW to hide system windows, such as the Command window, the Calculator, the Calendar/Diary etc.

<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.

BOTTOM | TOP | SAME

Include BOTTOM, TOP or SAME to specify where windows are hidden with respect to other windows. Including BOTTOM places a window behind all other windows; including TOP places it in front of all other windows. Issuing HIDE WINDOW with SAME hides a window without affecting its front to back placement. The default is TOP. To preserve the relative positions of multiple hidden windows when they are redisplayed with SHOW WINDOW ALL, include the SAME keyword when you hide them.

ATMIYA

0620. Show Window

Syntax

SHOW WINDOW <window name1>[, <window name2> ...] | ALL

[IN [WINDOW] <window name3> | IN SCREEN]

[REFRESH] [TOP | BOTTOM | SAME]

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

<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 instead of in another window. Windows are placed on the desktop 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. The work area for the Browse window table 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 for additional information on how data is refreshed in tables 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 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.

0621. Release Window

Syntax

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.

RELEASE <expr> is Releases memory variables and arrays, libraries, menus, menu pads, popup, user-defined or system windows from memory.

Syntax

RELEASE <memvar list>

or

RELEASE ALL [LIKE <skel> | EXCEPT <skel>]

or

RELEASE LIBRARY <library name>

or

RELEASE MENUS [<menu list> [EXTENDED]]

or

RELEASE PAD <pad name> OF <menu name> | ALL OF <menu name>

or

RELEASE POPUPS [<popup list> [EXTENDED]]

or

RELEASE BAR <expN> OF <popup name>| ALL OF <popup name>

or

RELEASE WINDOWS [<window list>]

0401. Introduction to Work Area

To manipulate multiple database files, one would require multiple work area. Default work area is current say 1. Fox Pro can have maximum 255 different work areas. Available work area can be found for new database access using SELECT() statement. you can give work area a symbolic name called an alias.

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.

Syntax:

SELECT([0 | 1])

Returns:

Numeric

ATMIYA

0402. Select Statement

Syntax

SELECT <expN> | <expC>

Activates the specified work area.

You can specify a different work area with <expN>. When a table is opened in a work area, you can select the table's work area by specifying the table's alias with <expC>.

Fields in tables in any work area can be included in FoxPro commands and functions. Use the following formats to access fields in a table open in a work area other than the current work area:

<alias>.<field>

or

<alias> -> <field>

Example:

```
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

0403. Set Relation

Syntax

SET RELATION TO

[<expr1> INTO <expN1> | <expC1>

[, <expr2> INTO <expN2> | <expC2> ...]

[ADDITIVE]]

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...

customer number, credit limit and payment terms.

You can use SET RELATION to relate these two tables on their common fieldthe 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.

<expr1>

The general relational expression <expr1> establishes the relationship between the parent and child tables.

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

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.

0404. Memory Variables

Variable is a named memory block which stores data. FoxPro provides multiple way of using memory variables. Variables are temporary storage space which is automatically erase data after execution of program is over. Three types to declare memory variables are...

1) Store Command

2) Public

3) Private

---

1 Store Command

Syntax

STORE <expr> TO <memvar list> | <array list>

STORE sets the value of a memory variable. 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 {11/18/75} TO BDT

<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.

Examples:

STORE DATE( ) TO mdate && stores system data to date datatype variable mdate

STORE 50 TO mnum && stores value 50 to number type variable mnum

STORE 'Hello' TO mchar && stores 'Hello' to character variable

STORE .T. TO mlog && stores true to boolean variable

---

2 Public

Syntax

PUBLIC <memvar list>

or

PUBLIC [ARRAY] <array1>(<expN1>[, <expN2>])

[, <array2>(<expN3>[, <expN4>])] ...

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.).

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.

<memvar list>

specifies the names of one or more memory variables to be initialized and designated as global.

PUBLIC a,b,c

[ARRAY] <array1>(<expN1>[, <expN2>) [, <array2>(<expN3>[, <expN4>]) ...

The array clause specifies one or more arrays to be initialized and designated as global.

PUBLIC Array a(5) && generates array named 'a' with 5 elements.

---

3 Private

Syntax

PRIVATE <memvar list>

or

PRIVATE ALL [LIKE <skel> | EXCEPT <skel>]

Hides specified memory variables or arrays that were defined in a calling program from the current program. PRIVATE doesn't create variables; it simply hides variables declared in higher-level programs from the current program.

<memvar list>

<memvar list> specifies the memory variables or arrays to be declared

private.

PRIVATE a,b

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 *.

0405. Copy To

Syntax

COPY TO <file> [FIELDS <field list>]

[<scope>]

[FOR <expL1>]

[WHILE <expL2>]

[[WITH] CDX]

COPY TO <file> [FIELDS <field list>]

Copies the contents of the current table in use to a new file as specified in file argument. If you include FIELDS and a field list, you can specify which fields are copied to the new file.

USE Stud_Reg

LIST

COPY To abc && generates new dbf file with structure of Stud_Reg

COPY To efg FIELDS RollNo, Name && generates database file as efg with fields rollno, name only.

COPY TO <file> [FIELDS <field list>] [<scope>]

The scope clauses are: ALL, NEXT <expN>, RECORD <expN>, and REST. The default scope for COPY TO is ALL records. Scope argument is useful for selection of record desire.

COPY TO abc ALL

COPY TO <file> [FIELDS <field list>]

[<scope>]

[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.

COPY To aaa NEXT 3 RollNo<3

COPY TO <file> [FIELDS <field list>]

[WHILE <expL2>]

If WHILE <expL2> is included, records are copied as long as the logical expression <expL2> evaluates to true (.T.).

ATMIYA

0406. Append From

Syntax

APPEND FROM <file> | ?

[FIELDS <field list> | FIELDS LIKE <skel> | FIELDS EXCEPT <skel>]

[FOR <expL>]

[[TYPE] [DELIMITED [WITH TAB | WITH <delimiter> | WITH BLANK] | DIF | FW2 | MOD | PDOX | RPD | SDF | SYLK | WK1 | WK3 | WKS | WR1 | WRK | XLS]]

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.

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.

<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.

USE efg

APPEND FROM abc FIELDS rollno, name

Appends records in efg.dbf file from abc.dbf for the fields rollno, name.

FIELDS LIKE <skel> | FIELDS EXCEPT <skel>

You can selectively append data to fields by including the LIKE or EXCEPT clause or both. If you include LIKE <skel>, FoxPro appends data to the fields that match <skel>. If you include EXCEPT <skel>, FoxPro appends data to all fields except those that match <skel>.

The skeleton <skel> supports wildcard characters. For example, to append data to all fields that begin with the letters A and P, use:

APPEND FROM FIELDS LIKE A*,P*

The LIKE clause can be combined with the EXCEPT clause:

APPEND FROM FIELDS LIKE A*,P* EXCEPT PARTNO*

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.

ATMIYA

0407. Copy Structure

Syntax

COPY STRUCTURE TO <file>

[FIELDS <field list>]

[[WITH] CDX]

COPY STRUCTURE creates an empty table/.DBF named <file> with the same structure as the current table/.DBF.

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.

COPY STRUCTURE TO MyNewTable

FIELDS LIKE <skel> | FIELDS EXCEPT <skel>

You can selectively copy fields to the new table/.DBF by including the LIKE or EXCEPT clause or both. If you include LIKE <skel>, FoxPro copy fields that match <skel> to the new table/.DBF. If you include EXCEPT <skel>, FoxPro copies all fields except those that match <skel> to the new table/.DBF.

The skeleton <skel> supports wildcard characters. For example, to copy all fields that begin with the letters A and P to the new table/.DBF, use:

COPY STRUCTURE TO mytable FIELDS LIKE A*,P*

The LIKE clause can be combined with the EXCEPT clause:

COPY STRUCTURE TO mytable FIELDS LIKE A*,P* EXCEPT PARTNO*

[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.

0408. Join

Syntax

JOIN WITH <expN> | WITH <expC>

TO <file> FOR <expL>

[FIELDS <field list> | FIELDS LIKE <skel> | FIELDS EXCEPT <skel>]

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.

WITH <expN> | WITH <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.

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.

FIELDS LIKE <skel> | FIELDS EXCEPT <skel>

You can selectively specify fields to be included in the new table/.DBF by including the LIKE or EXCEPT clause or both. If you include LIKE <skel>, the fields that match <skel> are included in the new table/.DBF. If you include EXCEPT <skel>, all fields except those that match <skel> are included in the new table/.DBF.

The skeleton <skel> supports wildcard characters. For example, to include all fields that begin with the letters A and P, use the following clause:

FIELDS LIKE A*,P*

The LIKE clause can be combined with the EXCEPT clause:

FIELDS LIKE A*,P* EXCEPT PARTNO*

ATMIYA

0409. Update

Syntax

UPDATE ON <field> FROM <expN> | <expC>

REPLACE <field1> WITH <expr1>

[, <field2> WITH <expr2> ...]

[RANDOM]

Updates fields in the current table/.DBF with data from another table/.DBF open in another work area.

UPDATE 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.

ATMIYA

0410. Dos Commands

COPY FILE <file1> TO <file2>

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>.

RENAME <file1> TO <file2>

Use RENAME to change the name of a file, <file1>, to a new name, <file2>. 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.

ERASE <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.

RUN <MS-DOS command > or

! <MS-DOS command >

Executes external operating commands or programs. RUN can be issued from within the Command window or from a program.

ATMIYA

0411. Scatter and Gether

Syntax

SCATTER [FIELDS <field list> | FIELDS LIKE <skel> | FIELDS EXCEPT <skel>]

[MEMO]

TO <array> | TO <array> BLANK | MEMVAR | MEMVAR BLANK

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

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. General and picture fields are always ignored in SCATTER, even if the MEMO keyword is included.

FIELDS LIKE <skel> | FIELDS EXCEPT <skel>

You can selectively transfer fields to memory variables or an array by including the LIKE or EXCEPT clause or both. If you include LIKE <skel>, fields that match <skel> are transferred to the memory variables or the array. If you include EXCEPT <skel>, all fields except those that match <skel> are transferred to the memory variables or the array.

The skeleton <skel> supports wildcard characters. For example, to transfer all fields that begin with the letters A and P to the memory variables or the array, use: SCATTER FIELDS LIKE A*,P* TO myarray

The LIKE clause can be combined with the EXCEPT clause:

SCATTER FIELDS LIKE A*,P* EXCEPT PARTNO* TO myarray

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 \ MEMVAR \ MEMVAR 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.

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. 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. Important Don't include TO with MEMVAR. An array named MEMVAR is created if TO is included.

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.

Syntax

GATHER FROM <array> | MEMVAR

[FIELDS <field list> | FIELDS LIKE <skel> | FIELDS EXCEPT <skel>] [MEMO]

Stores data from a memory variable array or a set of memory variables to the current record of the active table/.DBF.

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 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.

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.

FIELDS LIKE <skel> | FIELDS EXCEPT <skel>

You can selectively replace fields with the contents of array elements or memory variables by including the LIKE or EXCEPT clause or both. If you include LIKE <skel>, FoxPro replaces the fields that match <skel>. If you include EXCEPT <skel>, FoxPro replaces all fields except those that match <skel>.

The skeleton <skel> supports wildcard characters. For example, to replace all fields that begin with the letters A and P, use:

GATHER FROM myarray FIELDS LIKE A*,P*

The LIKE clause can be combined with the EXCEPT clause:

GATHER FROM myarray FIELDS LIKE A*,P* EXCEPT PARTNO*

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. General and picture fields are always ignored in GATHER, even if the MEMO keyword is included.

EXAMPLE

SET TALK OFF

USE ebookmark

COPY TO temp

USE temp

SCATTER MEMVAR MEMO

DEFINE WINDOW me FROM 5,10 to 20,70 PANEL

ACTIVATE WINDOW me

```
@ 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

@ 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

    WAIT WINDOW 'Changes saved' NOWAIT

ELSE

    WAIT WINDOW 'Changes not saved' NOWAIT

ENDIF

DEACTIVATE WINDOW me

BROWSE

USE
```

INDIANA TECHNOLOGIES

EbookMark for FoxPro

04. Function Index

- 0412. Numeric Functions
- 0412. String Functions
- 0412. Date Functions
- 0412. Array Functions

ATMIYA

0301. Sorting

Syntax

SORT TO <file> ON <field1> [/A | /D] [/C]

[, <field2> [/A | /D] [/C] ...] [ASCENDING | DESCENDING]

[<scope>]

[FOR <expL1>]

[WHILE <expL2>]

[FIELDS <field list> | FIELDS LIKE <skel> | FIELDS EXCEPT <skel>]

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

Note:

One or more specified fields in the current table/.DBF determine the order in which the records appear in the new table/.DBF.

- 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.
- 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.

SORT TO <file> ON <field1>

TO <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.

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.

SORT TO <file> ON <field1> <Scope>

The scope clauses are: ALL, NEXT <expN>, RECORD <expN>, and REST. These are explained in the Overview of the FoxPro Language chapter of the FoxPro Language Reference. 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.

SORT TO <file> ON <field1> 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.

SORT TO <file> ON <field1> 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.).

SORT TO <file> ON <field1> 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.

FIELDS LIKE <skel> | FIELDS EXCEPT <skel>

You can specify the fields that appear in the new table by including the LIKE or EXCEPT clause or both. If you include LIKE <skel>, the fields that match <skel> are included in the new table. If you include EXCEPT <skel>, all fields except those that match <skel> are included in the new table.

The skeleton <skel> supports wildcard characters. For example, to specify that all fields that begin with the letters A and P are included in the new table, use:

```
SORT TO mytable ON myfield FIELDS LIKE A*,P*
```

The LIKE clause can be combined with the EXCEPT clause:

```
SORT TO mytable ON myfield FIELDS LIKE A*,P* EXCEPT PARTNO*
```

ATMIYA

0302. Indexing

Syntax

INDEX ON <expr> TO <idx file> | TAG <tag name> [OF <cdx file>]

[FOR <expL>]

[COMPACT]

[ASCENDING | DESCENDING]

[UNIQUE]

[ADDITIVE]

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 SET TALK is ON, FoxPro reports how many records are indexed during the indexing process.

The number of index files (.IDX or .CDX) you can open is limited only by memory and system resources.

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.

Index Order and Updating

Only one index file 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.

INDEX ON <expr> TO <idx file>

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.

<expr>

<expr> can be a field or field expression from a table/.DBF in any 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.

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.

USE Stud_Reg

INDEX ON Roll_No TO Stud_Reg_File

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 system 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. Names of compound index files are given a .CDX extension.

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. 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.

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.

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.

0303. Locate

Syntax

LOCATE FOR <expL1>

[<scope>]

[WHILE <expL2>]

CONTINUE

If LOCATE finds a matching record, its record number is displayed if SET TALK is ON. 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. You can issue CONTINUE repeatedly until the end of the scope or the end of the table/.DBF is reached.

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.

FOR <expL1>

LOCATE sequentially searches the current table/.DBF for the first record that matches the logical expression <expL1>.

<scope>

The scope clauses are: ALL, NEXT <expN>, RECORD <expN>, and REST. 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.).

Examples:

LOCATE for Gender='M' ALL

LOCATE for Gender='M' Next 3

LOCATE While Age<22

CONTINUE

ATMIYA

0304. Find

Syntax

FIND <expC>

Searches an indexed table. FIND moves the record pointer to the first record in the table whose index key matches the character expression <expC>. FIND requires the selected table 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 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

Note : First create database file PAY and add Record according to field.

```
USE PAY

INDEX ON NAME TO PAYIDX

    32 records indexed

FIND KALIND

? RECNO()

    OUTPUT ----> 8

DISPLAY NAME

    Record# Name
```

ATMIYA

8 KALIND

If record is not found in table then massage will be displayed in statusbar "no find"

0305 Seek

Syntax

SEEK <expr>

SEEK searches the current indexed table for the first occurrence of a record whose index key expression matches a general expression. You can use SEEK only with indexed tables, and you can search only on the index key expression. The match must be exact unless SET EXACT is OFF. Specify the index key expression you want SEEK to search for with <expr>.

If SEEK finds a matching record, RECNO( ) returns the record number of the matching record, FOUND( ) returns true (.T.) and EOF( ) returns false (.F.).

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.

EXAMPLE

Note : First create database file PAY and add Record according to field.

```
USE PAY

INDEX ON NAME TO PAYIDX

    32 records indexed

SEEK 'KALIND'

? RECNO()

    OUTPUT ----> 8

DISPLAY NAME

    Record# Name
```

8 KALIND

If record is not found in table then massage will be displayed in statusbar "no find"

0201. Creating Table

To create a new database file in FoxPro user has multiple ways as,

- Write "Create" in FoxPro command window. Generates file save dialog box and suggests to give file name before proceed.
- Write "Create <table_Name>" in command window, to create table in default directory. As a result table structure window will come.
- Write "Create <\\Path\table_name.dbf>" in command window, to create table in specified path.
- Select New option from file menu to have New option dialog box. Select Table/DBF and click "New" button. As a result untitled table generates.

FoxPro table has extension as *.DBF. You can't use library reserve word (say keywords) for the file name like com1, list, clear etc. Figure shows blank structure displayed for table creation dialog box.

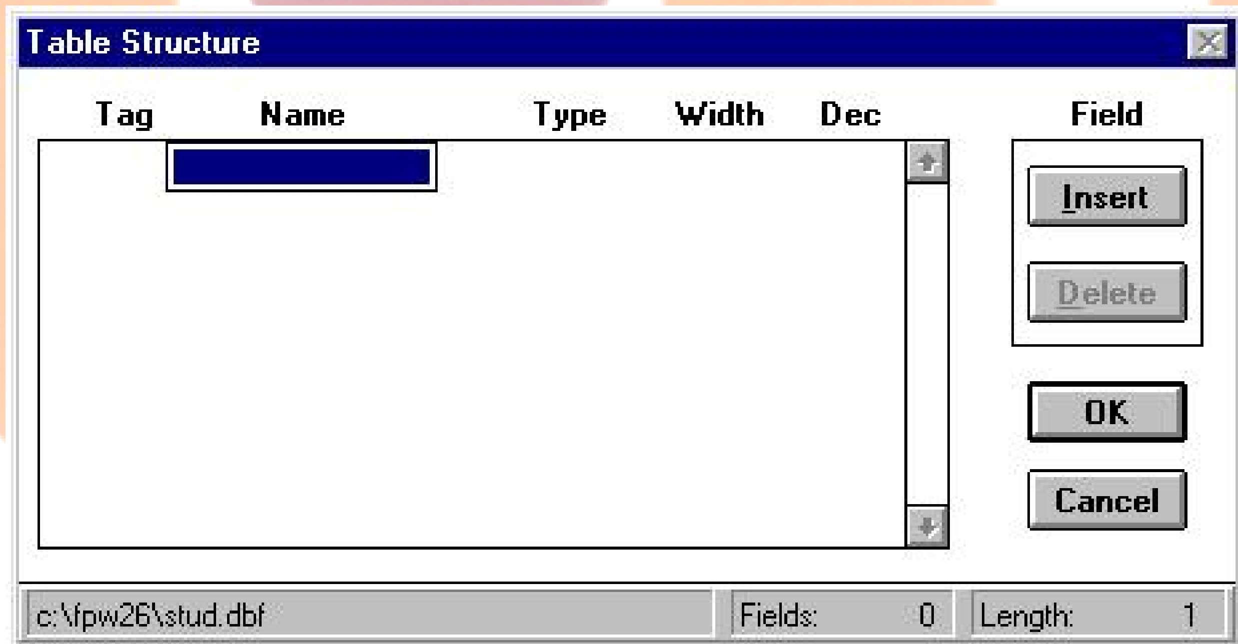


Table structure consist of following features...

Field Name:

To define a field name we follow few following rules.

- Each database fieldname can be up to 10 characters long.

  illegal: Student_Name, CLient_address

  legal : Student_Nm, CLient_add

- Fieldname must begin with a letter (a to z)

  illegal: 1name, @address

  legal: Name, Address1, Address2

- should not be the same as any command or even a valid abbreviation of a command.

  illegal: Create, Modify, Change, Brouse

  legal: Create_nm, Modify_dt

- Fieldname may include letters, numbers & the Special character underscore (_).

  illegal: #Address, 100Data

  legal: Address, Data100

- It may not contain blank spaces.

  illegal: Roll No, Mark 1, Mark 2

  legal: Roll_No, Mark_1, Mark_2

(To move next column press tab key...)

Type:

This column is useful to provide data type for each field set in earlier column. Data type is nature of data will come in field. Using this one can manage memory. Select data type from drop down list. To know more detail of each data type go to "Field types" topic in earlier chapter.

(To move next column press tab key...)

Width:

To set a width for field. This respects with earlier field (data type). Maximum field size for character field is 254. For number 20, date 8, Logical 1 and so on...

(To move next column press tab key...)

Decimal:

To set a decimal point for numeric fields. Maximum size of decimal field is one less than size of width of field. Decimal argument is not valid with character of any non number (as well float) field.

Enter as many fields you wish in database table. **Insert** button is useful to insert field in between two field at desired place. **Delete** button removes field which is pointed. **OK** button saves changes and creates *.dbf file on secondary storage device. **Cancel** button undo changes made by user and keep file as is was earlier.

Entering Records in database:

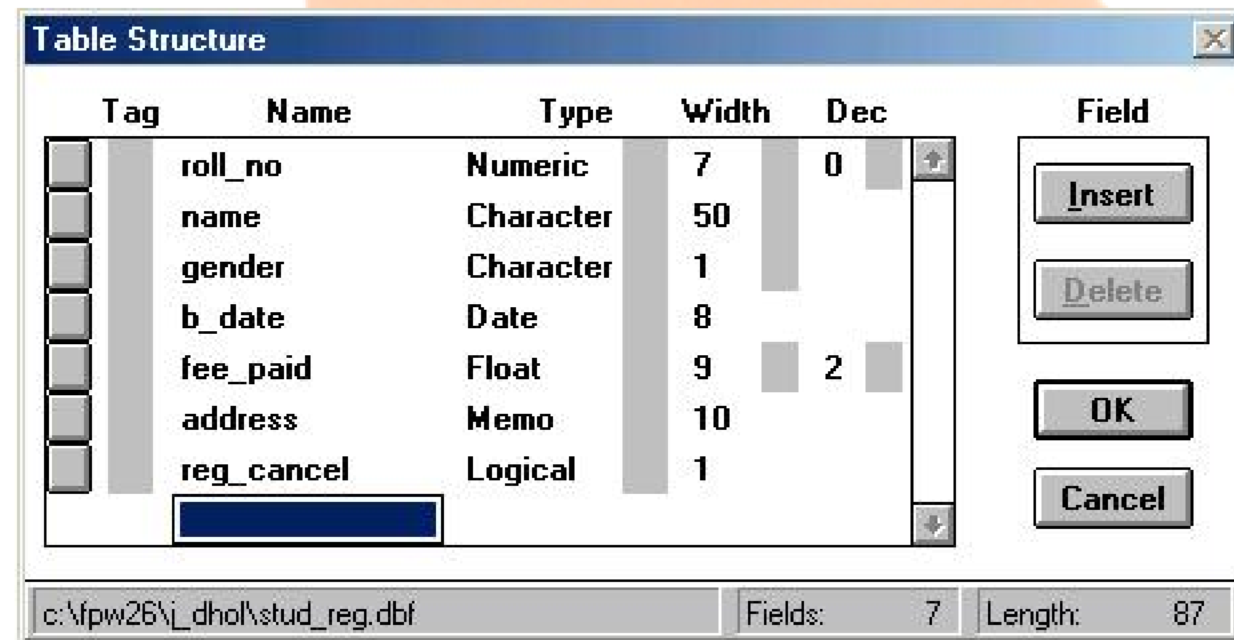
After entering a structure press CTRL + W or <<OK>> button to save structure. The following dialog box will be displayed on a screen which consist

two push button <<Yes>> and <<No>>

By default it will be <<Yes>>. When you press Enter, it allows to add new records into database file and when you select <<No>> it directly moves to command window.

Example of Database Structure:

Below example shows use of Table structure possible values with field name, type, width and decimal.



0202. Opening and Closing Table

Opening Table:

Syntax

Use [[\\path\] | filename | ?] [<index file name> | ?>]

Opens a DBF file and associated index files in the current work area. Here [\\path\] argument is optional and if given, FoxPro opens database on non default directory also. If USE is issued without a DBF file name, the file which is open in the current work area gets closed. The name of the DBF file to open is specified with <file>. If you include ? instead of a DBF file name, The Open dialog appears with a list of available DBF files to choose a file.

You can open index files with the DBF file by including INDEX and specifying a set of indexes with <index file list>. If a DBF file has a structural compound index file, the index file is automatically opened with the DBF file. You can open a single index file by issuing "INDEX ?". The Open dialog appears with a list of available index files.

Closing Table:

Syntax

CLOSE ALL (or) CLOSE DATABASES

When user opens any database file (*.dbf), it opens in current work area. That means to open multiple dbf files user requires more than one work area. So to close database two option are there…

CLOSE DATABASES:

CLOSE DATABASES closes all open DBF, index, memo and format files and selects work area 1.

CLOSE ALL:

Writting "CLOSE ALL" in command window, closes all files in all work areas and selects work area 1. CLOSE ALL also closes the following windows: Label, Project, Report, RQBE, Screen.

Example showing Open and Close option of Database:

ATMIYA

0203. List

Syntax

LIST [FIELDS <expr list>] [<scope>]

[FOR <expL1>] [WHILE<expL1>] [OFF]

[TO PRINTER [PROMPT] | TO FILE <file name>]

[NOCONSOLE]

LIST

List command display all record from the database file in use. For example, write below mentioned statement in FoxPro Command Window.

List

LIST [FIELDS <expr list>]

List command can display selected fields value, only when we specify the field option. To display only name, city and pin fields value use this...

List fields name,city,pin

LIST [<scope>]

<scope> operate only on the active work area's DBF file. The scope clauses are:

All	All records in the database file. The default scope for List command is **all.** For Example, **List All**
Next	Specified number of records from the current record position. Example will show next 3 records **List Next 3**
Record	Lists Specified record number. Example will show second record output of all records. **List Record 2**
Rest	Lists Current record and all subsequent records in table. **List Rest**

LIST [FOR <expL1>]

If the FOR clause is included, only the records that satisfy the logical condition <expL1> are displayed. Example will show all columns but sorts record where city name is "rajkot". "For <exp1>" applies to whole database records.

List for city = "rajkot"

LIST [WHILE<expL1>]

If the WHILE clause is included, records are displayed for as long as the logical expression <expL1> evaluates to true (.T.). This means by, starting from existing cursor position to end of file. If the current cursor position is on record 3 then condition will be checked from record 3 and previous 2 records will be ignored. This is the difference between "For <exp>" and "While <exp>". Below example will show records for Roll_no less than 30 from current cursor position.

List while roll_no < 30

LIST [OFF]

Include OFF to suppress record number display. If OFF is omitted, the record number is displayed before each record. Record is maintained by FoxPro itself. Record no is sequence for all the record, and it is maintained automatically while user manipulate (insert, update, delete) the records. Example will show all the records and columns except Record No column.

List off

LIST [TO PRINTER [PROMPT]]

Below mentioned statement prints all the record given by List command as well echo to desktop of FoxPro.

List TO PRINTER

Below mentioned statement gives print dialog box before you command it to print.

List TO PRINTER PROMPT

LIST [TO FILE <file name>]

This command directs all the data retrived by list command to the file user has specified in compulsory argument <File name>. As well it echo to screen for all the records retrieved by List command.

List TO FILE Jignesh.txt

LIST [NOCONSOLE]

Include [NOCONSOLE] to suppress output to the desktop. This is useful while user seek for output on Printer and File.

List TO FILE Jignesh.txt NOCONSOLE

NOTE:

Above mentioned arguments can be used all together. For simplicity these arguments are explained individually here. In practical use these arguments can be combined as per requirement and use. Other kind of List commands are given here...

LIST FILES

[ON <drive>]

[LIKE <skel>]

[TO PRINTER [PROMPT] | TO FILE <file>]

Example Lists available database (*.dbf) files on current path...

LIST files

Example Lists available database (*.dbf) files on path given...

LIST files On D:

Example Lists available database (*.dbf) files on path given with specified name type i.e. filename starts with f...

LIST files On D:\ LIKE f*

LIST MEMORY

[LIKE <skel>]

[NOCONSOLE]

[TO PRINTER [PROMPT] | TO FILE <file>]

Command gives list of memory variables, Menu definitions, Popup definitions and windows definitions which uses memory. Content can be stored directly in file as well can be send directly to a printer.

LIST STATUS

[NOCONSOLE]

[TO PRINTER [PROMPT] | TO FILE <file>]

Command gives list of flags set for environment of FoxPro. Also provides information about system, work directory, processor, environment settings and memory utilization by application.

LIST STRUCTURE

[NOCONSOLE]

[TO PRINTER [PROMPT] | TO FILE <file>]

Command gives structure of table which is in use. Below example gives table structure on screen along with Field No, Field name, type, width and Decimal.

List STRUCTURE

structure of table can be send to any file or printer using...

List STRUCTURE TO FILE abc.txt

0204. Display

Syntax

DISPLAY [FIELDS <expr list>] [<scope>]

[FOR <expL1>] [WHILE<expL1>] [OFF]

[TO PRINTER [PROMPT] | TO FILE <file name>]

[NOCONSOLE]

These DISPLAY commands are identical to the LIST command except for the following differences:

DISPLAY	LIST
The default scope of DISPLAY command IS NEXT.	The default scope of LIST command is ALL.
Display prompts you after filling the desktop with possible result of your query. This is more useful while you have no of records in database to display to the user on FoxPro Desktop.	List does not prompt you before it scroll no of record which are not possible to show on desktop at a time. This carries incosistency in displaying records to the user.

Below example will show only current cursor position record.

Display

Example will show all records from database.

Display All

Example will show next 2 record including current positioned record.

Display next 2

Example will show all records from database having city name 'rajkot'.

Display all for city = "rajkot"

Example will show all records from current cursor position to EOF which satisfies Roll_No value less than 30.

Display all while roll_no < 30

Example will show all records from database except Record No column.

Display all off

Example will show all records to desktop as well prints from printer.

Display all TO PRINTER

Example will show all records to desktop as well saves record in abc.txt file.

Display all TO FILE abc.txt

ATMIYA

0205. Goto

GO / GO TOP / GO BOTTOM

Syntax

GO [RECORD] <expN1> [IN <expN2> | IN expC>]

(or)

GO TOP | BOTTOM [IN <expN2> | IN <expC>]

(or)

GOTO [RECORD] <expN1> [IN <expN2> | IN expC>]

(or)

GOTO TOP | BOTTOM [IN <expN2> | IN <expC>]

GO [RECORD] <expN1> [IN <expN2> | IN expC>]

Positions the record pointer on the specified record number in a DBF file. You can use either GO or GOTO. For Example,

GO 1

GOTO 1

RECORD <expN1> Moves the record pointer to the physical record number <expN1>.

GO RECORD 2

GOTO RECORD 2

IN <expN2> | IN <expC> Moves the record pointer in a DBF file in another work area, include the work area number <expN> or the DBF file alias <expC>.

GO RECORD 2 IN Invoices

GO BOTTOM IN Invoices

GO TOP or GOTO TOP

position the record pointer on the first records, in the current DBF file.

GO BOTTOM or GOTO BOTTOM

position the record pointer last record, in the current DBF file.

ATMIYA

0206. Skip

Syntax

SKIP [<expN1>] [IN <expN2> | <expC>]

Moves the record pointer forward or backward in the current or specified DBF file. For Example,

SKIP 1 (skips on record forward if not EOF)

SKIP (skips on record forward if not EOF)

SKIP -1 (skips on record backward if not BOF)

Include a work area number **<expN2>** or work area alias **<expC>** to move the record pointer in a work area other than the current work area.

SKIP 4 IN Student (Skips 4 records in work area student)

ATMIYA

0207. Edit

Syntax

EDIT [FIELDS <field list>] [<scope>]

[FOR <expL1>] [WHILE <expL2>]

[FONT <expC1>[, <expN1>]] [STYLE <expC2>]

While manipulating data in database, user may require certain field values to be change or say EDIT. EDIT command display fields for editing values in database.

Below example gives edit record screen to edit all record.

Edit

If FIELDS argument is included in EDIT command, the listed fields are displayed in the order they appear in <field list>. If FIELDS isn't included, all fields in the DBF file are displayed.

Edit FIELDS roll_no, name, gender

ATMIYA

<scope> operate only on the active work area's DBF file. The scope clauses are:

All	All records in the database file. The default scope for EDIT command is **all.** For Example, **EDIT All**

Next	Specified number of records from the current record position. Example will show next 3 records **EDIT Next 3**
Record	Lists Specified record number. Example will show second record output of all FIELDS. **EDIT Record 2**
Rest	Gives Current record and all subsequent records in table. **EDIT Rest**

If the FOR clause is included, only the records that satisfy the logical condition <expL1> are displayed. Example will show all columns but sorts record where city name is "rajkot". "For <exp1>" applies to whole database records.

EDIT for city = "rajkot"

EDIT FIELDS roll_no, reg_cancel FOR reg_cancel = .T.

If the WHILE clause is included, records are displayed for as long as the logical expression <expL1> evaluates to true (.T.). This means by, starting from existing cursor position to end of file. If the current cursor position is on record 3 then condition will be checked from record 3 and previous 2 records will be ignored. This is the difference between "For <exp>" and "While <exp>". Below example will show records for Roll_no less than 30 from current cursor position.

EDIT while roll_no < 30

Fonts can be set in EDIT command window. The font style is specified with

<expC2>. If the STYLE clause is omitted, the standard font style is used.

EDIT ALL FONT 'ROMAN',16

EDIT ALL FONT 'ROMAN',16 STYLE 'BI'

EDIT FIELDS roll_no, name WHILE roll_no<3 FONT 'ROMAN', 20 STYLE 'I'

0208. Append

Syntax

APPEND [BLANK]

APPEND command is useful when user wish to insert new record in the database. APPEND allows new records to be added to the bottom of the database currently in use.

APPEND

Insert one blank record in database and gives insertion dialog box for field values.

APPEND BLANK

Adds one blank record to the end of the current table. When a blank record is appended, each field in the record is initially empty unless CARRY is SET ON. APPEND BLANK does not provide record insertion screen to user, It just add blank record in database.

ATMIYA

0209. Browse

Syntax

BROWSE [FIELDS <field list>]

[FONT <expC1>[, <expN1>]] [STYLE <expC2>]

[FOR <expL1>] [FORMAT]

[FREEZE <field>]

[NOAPPEND] [NODELETE] [NOEDIT | NOMODIFY]

[TITLE <expC4>]

[COLOR SCHEME <expN6> | COLOR <color pair list>]

BROWSE is one of the most useful commands available in FoxPro. Use BROWSE to open a Browse window. It displays record from a table into GRID form. You can then easily edit and append records. To add new record in table press **ctrl+N**. So BROWSE command is useful in manipulation of database. Certain validation can be taken in BROWSE command like freeze field, Noappend, Nodelete etc. 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.

BROWSE

BROWSE command displays all the records available in database into Grid.

BROWSE [FIELDS <fields 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 are displayed in the order they appear in the table structure.

BROWSE FIELDS roll_no, name

You can include fields from other related tables in the field list. When you include a field from a related table 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]	Read Only
[:column width]	Column Width
[:V = <expr1> [:F] [:E = <expC1>]]	Verify Forced Validation Error Message
[:P = <expC2>]	Picture
[:B = <expr2>, <expr3> [:F]]	Boundaries

[:H = <expC3>]	Headings
[:W = <expL1>]	Conditionally Prohibit

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.

```
BROWSE FIELD LOCATION = ALLTRIM(city) + ', ' + state
```

FONT <expC1>[, <expN1>]

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:

```
BROWSE FONT 'ROMAN',16
```

STYLE <expC2>

To specify a font style for fields displayed in the Browse window. Windows determines the styles that are available for a font. 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.

```
BROWSE FIELDS contact FONT 'System', 15 STYLE 'NU'
```

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.

```
BROWSE FOR RollNo < 3
```

FORMAT

Include the FORMAT clause to use a format file which helps you format the display in a Browse window. A format file must first be opened with SET FORMAT command. This is more useful while you wish to represent data entities in specified format or character set. These has several clauses and can be used while programming with FoxPro to create user friendly screens. This clause can be viewed in next topics as and when required.

FREEZE <field>

This command provide data manipulation for the column given along with FREESE clause. Programmer can specify one column name at a time, remaining fields are displayed and cannot be edited.

```
BROWSE FREEZE NAME
```

Above command will show all database field having editing facility with NAME column and rest columns as FREESE (no edit facility).

NOAPPEND

If you include NOAPPEND in BROWSE command, user can't add records to the DBF file by pressing Ctrl+N or choosing Append Record from the Browse menu.

```
BROWSE NOAPPEND
```

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.

```
BROW NODELETE
```

NOEDIT | NOMODIFY

Including NOEDIT or NOMODIFY prevents you from modifying the table content. NOEDIT and NOMODIFY are identical. If you include either clause, you can browse or search the table, but you cannot edit it. However, you can append and delete records.

```
BROWSE NOEDIT
```

TITLE <expC4>

By default, the name of the table 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.

```
BROW TITLE "eBookMark for FoxPro"
```

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.

If VALID returns a true value (.T.), you can move the cursor to another record.

```
BROW VALID :F city = "rajkot" ERROR "Enter city equals rajkot only"
```

COLOR SCHEME <expN6> | COLOR <color pair list>

Including the number of an existing color scheme in the COLOR SCHEME clause or a set of color pairs in the COLOR clause can specify the color of a Browse window.

For example, this color pair specifies a red foreground on a white background: R/W

```
BROW COLOR R+/B
```

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) (this gives red forecolor and blue backcolor)

```
BROW COLOR RGB(255,0,0,255,255,255)
```

ATMIYA

0210. Delete

Syntax

DELETE [<scope>] [FOR <expL1>] [WHILE <expL2>]

DELETE command marks record with * character that shows deleted record. Records marked (*) for deletion aren't physically removed from the table.

DELETE

DELETE [<scope>]

The scope clauses are: ALL, NEXT <expN>, RECORD <expN>, and REST. The default scope for DELETE is the current record. <scope> operate only on the active work area's DBF file. The scope clauses are:

All	All records in the database file. For Example, **DELETE All**
Next	Specified number of records from the current record position. Example will delete next 3 records **DELETE Next 3**
Record	Delete Specified record number. Example will delete second record in all records. **DELETE Record 2**
Rest	Delete Current record and all subsequent records in table. **DELETE Rest**

DELETE for <expl1>

If the FOR clause is included, only the records that satisfy the logical condition <expL1> are marked for deletion. For example, to delete record from student register having rollno greater than 50....

DELETE for rollno>50

If condition value is of type character, one could write

DELETE for Name='Jignesh'

DELETE While <expl2>

If the WHILE clause is included, records are marked for deletion for as long as <expL2> evaluates to true (.T.) from cursor current position.

DELETE While rollno<50

ATMIYA

0211. Recall

Syntax

RECALL [<scope>] [FOR <expL1>] [WHILE <expL2>]

RECALL Unmark records marked for deletion in the selected table. Once a file has been packed, all records marked for deletion are gone forever.

RECALL

This command recalls deleted records if current record on which cursor reside is deleted.

RECALL

RECALL <scope>

The scope clauses are: ALL, NEXT <expN>, RECORD <expN>, and REST. The default scope for RECALL is the current record. <scope> operate only on the active work area's DBF file. The scope clauses are:

All	Recalls (unmark deleted) All records in the database file. For Example, **RECALL All**
Next	Specified number of records from the current record position. Example will recall next 3 records **RECALL Next 3**
Record	RECALLs Specified record number. Example will recall second record in all records. **RECALL Record 2**

Rest	Recall Current record and all subsequent records in table. **RECALL Rest**

RECALL for <expl1>

If the FOR clause is included, only the records that satisfy the logical condition <expL1> are unmarked if deleted. For example, undo deleted record from student register having rollno greater than 50....

RECALL for rollno>50

If condition value is of type character, one could write

RECALL for Name='Jignesh'

RECALL While <expl2>

If the WHILE clause is included, records are unmarked if deleted for as long as <expL2> evaluates to true (.T.) from cursor current position.

RECALL While rollno<50

0212. Pack

Syntax

PACK

PACK permanently removes all records marked for deletion in the current table or table in use. PACK also reduces the size of a memo file associated with a dbf.

- DELETE command marks deleted record and does not remove records from database.
- RECALL command rollback deleted record which is marked as delete (*)
- PACK command removes delete marked records from database and empty memory space. Once records removed from database using PACK command can not be RECALL.

ATMIYA

0213. Zap

Syntax

ZAP

Removes all the records from table. Before removing all the records from database FoxPro ask for Zapping record as given in the figure.

Zap command leave DBF structure when you press yes button. Records zapped from the current table cannot be recalled.

- PACK command deletes only marked records.
- ZAP command removes all the records.

0214. Replace

Syntax

REPLACE <field1> WITH <expr1>

[ADDITIVE]

[, <field2> WITH <expr2>] ...

[<scope>]

[FOR <expL1>]

[WHILE <expL2>]

REPLACE <field1> WITH <expr1>

REPLACE replaces data in a field with the value in an expression. No replacement occurs if the record pointer is at the end of the file.

REPLACE City with 'Rajkot'

REPLACE <field1> WITH <expr1>

[ADDITIVE]

[, <field2> WITH <expr2>] ...

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.

REPLACE City with 'Rajkot' ADDITIVE City with ' District'

REPLACE <field1> WITH <expr1>

[<scope>]

The scope clauses are: ALL, NEXT <expN>, RECORD <expN>, and REST. The default scope for REPLACE is the current record (NEXT 1). This works same with all command as given in earlier commands.

REPLACE <field1> WITH <expr1> [FOR <expL1>]

If the FOR clause is included, the specified fields are replaced only in records for which <expL1> evaluates to true.

REPLACE <field1> WITH <expr1> [WHILE <expL1>]

If the WHILE clause is included, fields in records are replaced for as long as the logical expression <expL2> evaluates to true (.T.).

When you want to replace any formula at that time this command is very useful like if you want to increase salary with 10% then you use following command after using your database file.

```
REPLACE ALL SAL WITH SAL+SAL*.10
```

After replace data you get message into statusbar that, * record is replaced.

Other Ex. as follow

```
REPLACE ALL COMM WITH 200 FOR SAL > 5000

REPLACE ALL CITY WITH "RAJKOT"
```

```
REPLACE NEXT 3 SAL WITH SAL+COMM
```

* means record no which is affected with replace command.

0215. Sum

Syntax

SUM [<expr list>]

[<scope>]

[FOR <expL1>]

[WHILE <expL2>]

[TO <memvar list> | TO ARRAY <array>]

Totals all or specified **numeric field** values in the current table. All numeric fields are totaled if the expression list is omitted. Below example shows total fees paid by student.

SUM fees

The scope clauses are: ALL, NEXT <expN>, RECORD <expN>, and REST. The default scope for SUM is ALL records. Below example will show total fees paid by student because scope is all given that mean all the records.

SUM fees ALL

Below example will total of fees from current position to last record..

SUM fees REST

Below example will total of fees for record mentioned with number...

SUM fees RECORD 3

Below example will total of fees from current position to next no of records..

ATMIYA

SUM fees NEXT 3

If the FOR clause is included, only the records for which the logical condition <expL1> evaluates to true (.T.) out of all the records.

SUM fees FOR gender='M'

Above example will total sum of fees for sex 'male'.

If the WHILE clause is included, records from the current table are included in the total for as long as the logical expression <expL2> evaluates to true (.T.).

SUM fees WHILE gender='M'

Use the TO <memvar list> clause to store each sum to a memory variable.

Use TO ARRAY clause to store totals to a memory variable array.

[To have output on screen **set talk** command must be on].

Examples:

SUM

SUM SAL

SUM SAL FOR DEPT = 10

SUM SAL TO m_sal [m_sal is declared numeric memory variable]

SUM TO arr [arr is declared numeric array variable]

0216. Total

Syntax

TOTAL TO <file> ON <expr> [FIELDS <field list>] [<scope>] [FOR <expL1>] [WHILE <expL2>]

TOTAL computes totals for numeric fields in the current table. The table 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. One record is created in the second table for each unique index key value.

<file> is the name of the output table.

<expr> is the expression on which the source table is indexed or sorted.

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.

The scope clauses are: ALL, NEXT <expN>, RECORD <expN>, and REST. The default scope for TOTAL is ALL records.

If the FOR clause is included, only the records that satisfy the logical condition <expL1> are included in the totals.

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.).

Example "month.dbf" following daata file is created and have records.

recno	month	qty
1	feb	10
2	jan	20
3	feb	05

4	jan	08
5	july	07
6	july	08
7	feb	05

Apply the following command

Index on month to mon_idx

7 records are indexed.

Total to Tdb on month

3 records are totalled

create 3 records in Tdb.dbf then use this file

USE Tdb

List

output look like the following

recno	month	qty
1	feb	20
2	jan	28
3	July	15

0217. Average

Syntax

AVERAGE [<expr list>] [<scope>]

[FOR <expL1>] [WHILE <expL2>]

[TO <memvar list> | TO ARRAY <array>]

Computes the arithmetic mean of numeric expressions or fields. All numeric fields in the selected table are averaged unless you include an optional expression list.

The scope clauses are: ALL, NEXT <expN>, RECORD <expN>, and REST. The default scope for AVERAGE is ALL records.

If FOR <expL1> is included, only the records that satisfy the logical condition <expL1> are included in the average.

If WHILE <expL2> is included, records are included in the average for as long as the logical expression <expL2> evaluates to true (.T.).

The results of AVERAGE can optionally be stored to a list of memory variables or array elements or to a one-dimensional ARRAY.

Eg. [To see output on screen set talk command is must on]

AVERAGE

AVERAGE SAL

AVERAGE SAL FOR DEPT = 10

AVERAGE SAL TO m_sal [m_sal is declared numeric memory variable]

AVERAGE TO arr [arr is declared numeric array variable]

ATMIYA

0218. Change

SYNTAX

CHANGE [FIELDS <field list>] [<scope>]

[FOR <expL1>] [WHILE <expL2>]

[FONT <expC1>[, <expN1>]] [STYLE <expC2>]

[FREEZE <field name>] [NOAPPEND] [NODELETE]

[NOEDIT | NOMODIFY] [NOMENU] [NOWAIT]

[TIMEOUT <expN4>] [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>]

To see all the Options of change command refer following command

-> Edit

-> Browse

0101. Introduction to DBMS and RDBMS

What is a Database

Anything can be data a number, name of a person, city etc. When data is meaningful, it's called information.

A simple definition of a database is

"The database is an organised collection of related information "

Any unorganised information can't be called a database. You can use the database for

- taking meaningful decision,
- reorganizing information,
- retrieving information and
- processing information.

Some **example** of database file is telephone directory, student information in college and dictionary.

Database system, as computer base record keeping system whose overall purpose to record and maintain information. In other word, database is a collection of related records and a set of programs to access this data. Because it is an entire system and enables data to be enter, store and manage that is why it is call Database Management System.

Modern Database Management System comes in many different classifications, and with many different capabilities, but in general they try or accomplish two things.

Data Sharing

Data sharing refer to the ability of the system to allow multiple user concurrent access to the individual pieces of data in the database. You can

think of the database as a 'pool' of sharable information.

Data Protection

Data protection refers to the ability of a database management system to maintain integrity of its data, protecting against crashes, program failures, etc. If this type of events occurs, the DBMS must have the ability to back out (or undo) changes to data stored in the database.

What is Relational Database Management System ?

A relational database is a database structured on the relational model. A Relational Database Management System or RDBMS is a suite of software programs that can be used for creating, maintaining, modifying and manipulating a relational database. It can also be used to create the application that a user will require for interacting with the data stored within the database. Three important factors in any RDBMS exist are...

Base Tables

A base table is a table with a name that physically exists in a database. It is a created by the user. A base table can be created, altered and removed from a database. All these tasks are accomplished using SQL (Structured Query Language) statement.

Query Results

When a 'question' is asked to a table, the resultant data can be despayed from data stored in tables. Such tables are called query results.

Views

A view is a virtual table. Some columns of a base table may not be required by the user. In such cases, a view is created. This view will consist only of those columns of the base table that the user is interested in seeing. This format can be saved as a view by giving it a name.

ATMIYA

0102. Importance of DB and DBMS

Using Computers for database

A computer is more suited for database application because of two reasons.

- It can hold a large amount of data in its storage device
- It operates at a very high speed.

For building database, one normally uses a generalized software package, called the Data Base Management System (DBMS). As the name suggests, it is used to build and manage the database, i.e. add, edit, delete and sort (arrange in order) information in the database, to keep the database up- to-date.

How is a Database System Beneficial?

The amount of redundancy in the stored data can be reduced.

No more inconsistencies.

The store data can be shared.

Standards can be set and followed.

Data integrity is maintained.

Security of data can be implemented.

Data independence.

DBMS Users

The Database Designers

The Database Administrator or DBA

The Application Programmer

The actual End-Users of the application

0103. Database Models

Database Management Systems organize data in way is known as database model. You can think of a data model as the infrastructure of the data organization, in other word how the data is presented to the user. There are three basic data models:

- The Hierarchical Model
- The Network Model
- The Relational Model

(i) The Hierarchical Model

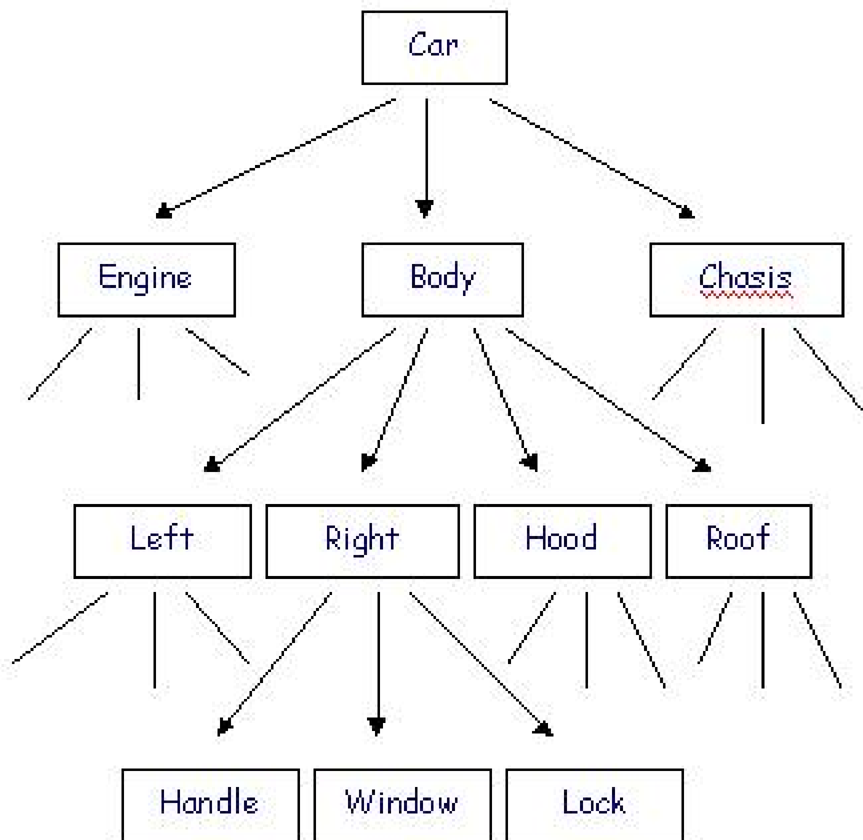
One of the earliest database management systems was based on the Hierarchical Model. In a hierarchical data model the records have a parent-child relationship. The application used was Production planning for automobile manufacturing companies. The model of database is shown in following figure. An automobile manufacture may manufacture various model of car. Each car model was decomposed into its assemblies (Engine, Body and Chassis). Each assembly is further decomposed into sub-assemblies (valves, spark plugs&ldots;) and so on. If manufacturer wanted to generate the Bill of Materials for a particular model of an automobile the hierarchical data model would be very suitable because the bill of materials for a product has hierarchical structure. Each record represents a particular part and since the records have a parent-child relationship each part is linked to its sub-part. The hierarchical model of support multiple occurrences of the same record type.

One of the most popular hierarchic database management system was IBM's Information Management System (IMS) introduced in 1968. IMS is still most widely used DBMS in IBM mainframes.

The Characteristics of DBMS are:

- **Data is represented as hierarchical trees.**

The hierarchical database is characterized by parent-child relationship between records. A record type, R1, is said to the parent of record type, R@, if R1 is one level higher than R2 in the hierarchic tree. The root of the hierarchy is the most important record type and all records at different levels of the hierarchy are dependent of the root. Each child record has only one parent record. The parent record can have one or more children record type.



(Fig. Shows Hierarchical Model)

- **Represents a set of related records.**

  There can be one or more than one record occurrences for given record type. When writes into database, one occurrence of record of the record type is written. Similarly, whenever a record is retrieved from the database, one occurrence if the record type is retrieved.

- **Hierarchy is established through pointers.**

In the hierarchic database, the pointers link the records. Pointers determine whether a particular record occurrence is a parent of child record and path from parent to the child.

- **Simple structure**

  The database is simple hierarchical tree. The parent and child records can be stored close to each other on the disk, minimizing disk input and output. The hierarchic data model is simpler than a network model.

- **High performance**

  The parent-child relationship is stored as a pointer from one record to another; hence navigation through the database is very fast resulting in high performance.

- **Relationships between record types are pro-defined**

  The hierarchical DBMS is based on the hierarchic tree structure in which the parent-child relationship is supported. A record type, R!, is said to be the parent of record type, R2, if R1 is one level higher than R2 in the hierarchical tree. Records types at different level of the hierarchy are dependent on the root, which is most important record type in the hierarchy. Since the relationships are predefined, flexibility is lost but a high performance compared to other data models is achieved.

- **Tedious to reorganize.**

  It is tedious to reorganize the database because the hierarchy has to be maintained. Each time a record type is inserted of deleted, the pointer have to be manipulated to maintain the parent-child relationship. The reorganization is static and appropriate changes have to be made to the application programs.

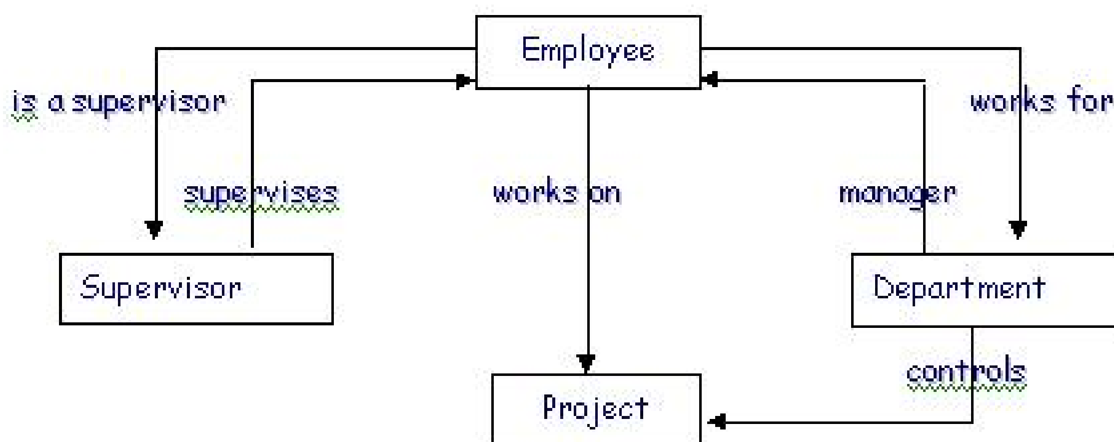
- **Real life requirement are more complex**

  The hierarchic DBMS is based on a simple parent-child relationship, but real life applications are more complex and cannot be

represented by a hierarchic structure. In an order-processing database, a single order might participate in three different parent-child relationships linking the order to the customer who placed the order, the items ordered and the sales person who took the order. This complex structure cannot be represented in a hierarchical structure.

(ii) The Network Model

To overcome the problem posed by the hierarchical data model, the network model was developed. The network model modified the hierarchical model by allowing multiple parent-child relationships. This relation is known as set in network model was developed. The network model together with the hierarchical data model was major a data model for implementing numerous commercial DBMSs. The network model structure and language construct were defined by the CODASYL (Conference on Data Systems Language).



(Fig. Shows Network Model)

The characteristics of a network DBMS are:

- Data record types are represented as a network.
- A network is used when hierarchy is not established or when a record participates is more than one relationship.

- Each sub-module can have one or more super-ordinate modules. Since each multiple parent child relationship is supported child record type could have one than more parent record types.

- Represents a set of related records.

  The sets that support multiple parent-child relationships and the structure of the record have to be specified in advance.

- Complex structure

  Since multiple parent-child relationship is supported, database structure is very complicated. The network database implements sets that support multiple parent-child relationships. The sets have to be specified in advance. In the tradeoff between flexibility and performance, a network model is not very flexible to reorganize but has high performance level.

- Difficult to reorganize

  The network database is very difficult to reorganized because insert and deleting a record would trace the pointers and changing the appropriate links.

- Navigation done by the programmer

  The programmer will have to write 3-GL programs specifying the relationship and direction in which to navigate in the database.

- 3-GL needed to program database

  To access records the programmer has to navigate the database record-by-record. Program will have to be written specifying to which relationship to navigate and the direction.

- 3-GL inadequate for handling sets.

  The records network model are processed one set at a time. 3-GLs handle only one record at a time and hence are inadequate for handling sets.

- Query facility not available

  Network database management system do not have any query facility and hence 3-GL programs will have to written specifying the path and the relationship.

(iii) The Relational Model

An IBM research scientist Dr. E. F. Codd, was unhappy with the way the DBMSs available in those day handled large volumes of data. He felt the need to apply the rules and decline of mathematics to help address the problems associated with the earlier models as

Data integrity

Data redundancy

In June 1970, he presented his paper titled ' A Relational Model of Data for Large shared Databanks'. This paper actually laid down 12 rules. Which a true RDBMS would have to satisfy.

The term 'Relation' is derived from the set theory of mathematics. The basis characteristics of a relational model are discussed here.

First, in a relation model, data is stored in relations. 'What are relations?' will be the next question that we will answer. Before that, consider the following example. Given below are two different lists. One is a list of countries and their capitals. The other lists countries and the local currencies used by them.

ATMIYA

Country	Capital
Greece	Athens
Italy	Rome
India	New Delhi
China	Beijing
Japan	Tokyo
Australia	Sydney
France	Paris
New Zealand	Auckland
Spain	Madrid
Peru	Lima
Portugal	Lisbon

Country	Currency
Greece	Drachma
Italy	Lira
India	Rupee
China	Remnimbi(Quan)
Japan	Yen
Australia	Australian Dollar
France	Francs
New Zealand	Dollars
Spain	Peso
Peru	Nuevo sols
Portugal	Escudo

You will notice that their two different lists shown here. However, there is a column, which is the common to both lists. This the column, which contains names of the country. Now if someone wants to know the currency used in Rome, first one should find out the name of the country. Next that country should be looked up the next list to find out the currency.

It is possible to get this information because it is possible to establish relation between the two lists through a common column called "country".

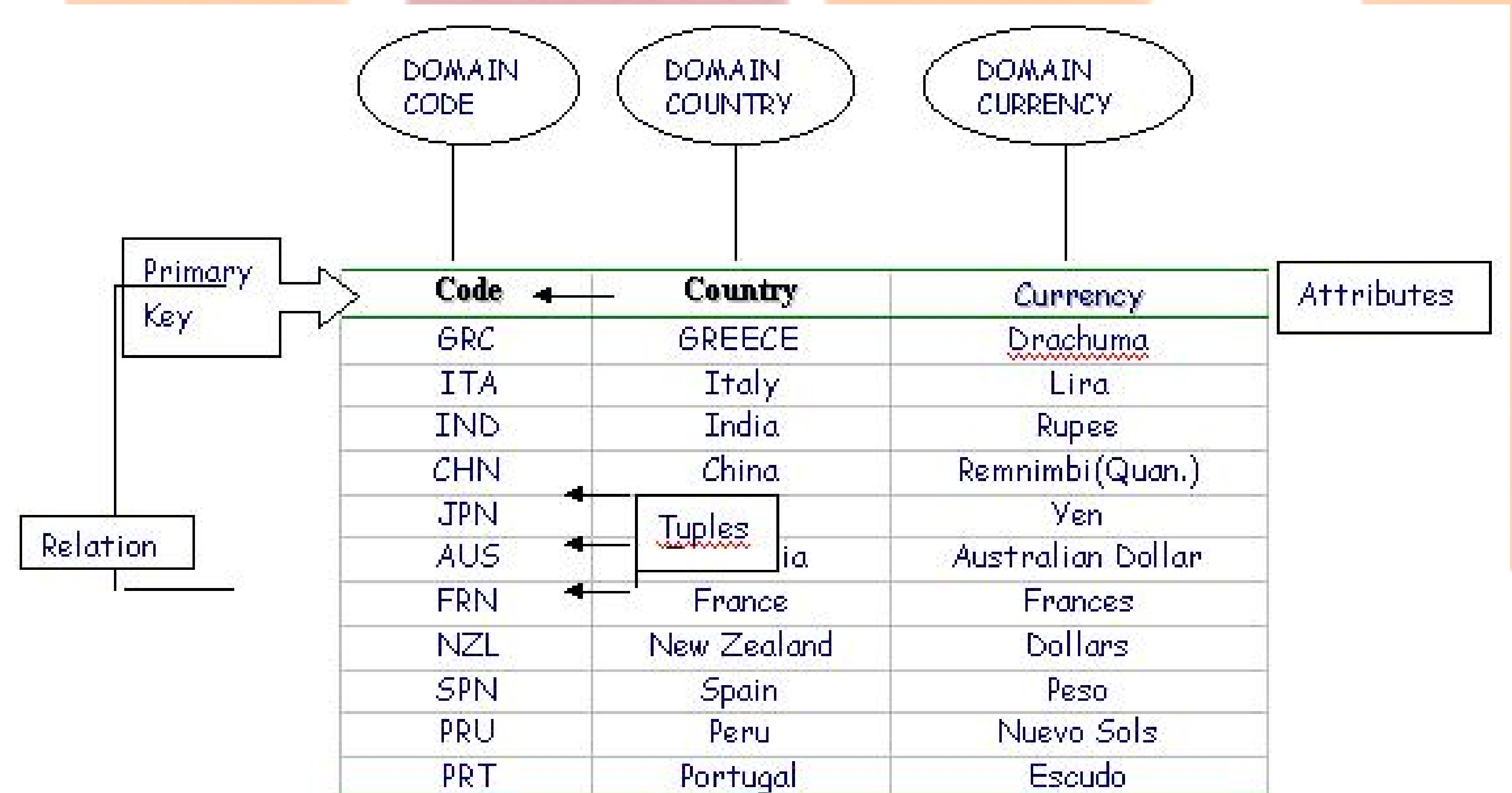
In the relational model data is stored in relations, relation is a formal term for the table. In the example above we have stored information about countries as a table. A table in a database as a unique table name that identifies its contents. Each table can be called an intersection of row and columns. One of the most important properties of a table is that the rows are unordered. A row cannot be identified by its position in the table. Every table must have a column that uniquely identifies by each row in the table. It is essential that

no two rows should contain identical information. This is prevented by the use of primary key.

Now we will exam the relational model in detail.

The relational model - the details

Let us consider any of the tables we have considered in the example above less us take the currency table.



Code	Country	Currency
GRC	GREECE	Drachuma
ITA	Italy	Lira
IND	India	Rupee
CHN	China	Remnimbi(Quan.)
JPN		Yen
AUS	ia	Australian Dollar
FRN	France	Frances
NZL	New Zealand	Dollars
SPN	Spain	Peso
PRU	Peru	Nuevo Sols
PRT	Portugal	Escudo

(Fig. Shows Relational Model)

a new column called "codes" has been introduced as the primary key. The table show above consist of the components listed below, according to a relational model:

Domain

Domain is pool of values from where one or more attributes (columns) can draw their actual values. For example, the values in the field "country" are available from the name of all the countries in the world. Hence, the domain

name for this field is country.

Tupple

according to the relational model, every relation or table is made up of many tuples. They are called records- a term that we are already familiar with. They are the rows that a table is made up of. Given below are some of the tuples that are part of the currency table.

CHN	China	Remnimbi (quan)
FRN	France	Francs
PRT	Portugal	Escudo

The number of tuples in a table is the cardinality of the tuple.

Attributes

The term "attributes" refers to characteristic. The characteristic of the tuple is reflected by its attributes or field. This simply means that what the column contains will be define by the attributes of that column. The number of attributes is called the degree of that table. Look at some of the attributes shown below.

Peso
Australia
New Zealand

Although are relational model prescribe these above terms they do not appear in the daily usage. The terms "records" and "fields" are commonly used.

Advantages of a Relational Database Model

Some of the salient advantages of a relational database model have been listed below:

Built in integrity at various levels.

Allow data integrity to be incorporated at the field level to ensure data accuracy; integrity at the table level to avoid duplication of records and to detect records with missing primary key values;

At the relationship level to ensure that relationships between tables are valid.

Logical and Physical data Independence from database applications

Changes made in the logical design of the database or changes made in the database software will adversely affect the implementation of the database.

Data consistency and accuracy

Due to the various levels, at which data integrity can be built in, data is accurate and consistent.

Easy data retrieval and data sharing

Data cab be easily extracted from one or more than one tables. Data can also be easily shared users.

0104. Data and File Concept

What is Data, Database File, Field & Record

Data: Data is a collection of information.

Database: The database is an organised collection of related information. Database is useful for...

- taking meaningful decision,
- reorganizing information,
- retrieving information and
- processing information.

Database File: When information is stored in a computer file using a database management system (DBMS). DBMS such as FoxPro or dBASE III Plus are available in market.

Field: A field is a collection of data of same type and nature. Each such column in database is also called a field. Field is a basic part of any database. This is also known as **Attribute**.

Record: Data is stored in one physical (horizontal) line of the database is called record. i.e. "Record is a collection values in fields of table". This is also known as **Tupple**.

ATMIYA

0105. Difference between DBMS v/s RDBMS

DBMS	RDBMS
The concept of relationships is missing in a DBMS. If it exists it is very less	It is based on the concept of relationships.
Speed of operation is very slow.	Speed of operation is very fast.
Hardware and software requirements are less.	Hardware and software requirements are high.
Facilities and Utilities offered are limited.	Facilities and Utilities offered are many.
Platform is used is normally DOS	Platform used can by any DOS, UNIX, VAX,VMS etc.
Uses concept of a file.	Uses concept of a table.
DBMS normally use a 3GL.	RDBMS normally use a 4GL.
Examples are dBASE, FOXBASE etc.	Examples are ORACLE, INGERS etc.

ATMIYA

0107. Special Feature of FoxPro

FoxPro for windows is a relational database management system, which allows you to work with several related groups or tables of data at once. Each table consists of data about a single topic. A database table contains Rows and Columns. One row int he table is equvalent to one record and one column is equivalent to one field.

The major features of FoxPro:

- Creating Tables.
- Viewing Data.
- Querying Data with RQBE.
- Retrieving Data from Multiple Tables.
- Designing Labels.
- Designing Reports.

HARDWARE AND SOFTWARE REQUIREMENTS

The hardware and software requirements for installing FoxPro for Windows are as follows:

- 80486 or higher.
- Minimum 8 MB RAM if virtual memory is set to none OR minimum 4 MB RAM if Virtual memory is set to temporary or permanent.
- Microsoft Windows 3.0 or higher.
- Mouse.
- VGA Monitor or higher.
- MS-DOS verson 3.1 or higher.

ATMIYA

- The hard disk whose space requirements would depend upon the type of installation to be done:
    - Complete Installation - It installs all FoxPro files including the main FoxPro files, help files, sample files, tutorials, etc.
    - Custom Installation - It installs only the options and files you specify.
    - Minimum Installation - It installs only the files required to run FoxPro for Windows.

ATMIYA

0109. Field Types

DATA TYPES

We have 7 data type available in FoxPro. Every data object has a data type associated with it, but at any identifier or memory variable has only one data type.

1. Numeric

It consist only numeric data. You can store digits (0 to 9), Decimal point and optional leading plus (+) or minus (-) sign. It can be up to 20 digits wide include decimal point and decimal part up to 0 to 16 digits.

2. Float

It is similar to numeric type, but it store more significant digits for the float type data as compared to the numeric data. The float type is better for scientific calculations.

3. Character

It consist the text made up of an alphabetical characters, numbers, spaces and special symbols. A character type is a string of 0 to 254 characters from the ASCII character set. Ordinary character variables store name, city, address etc.

4. Date

It is used to store a dates. Width of data field is 8 characters. The dates are stored in a specify default American format mm/dd/yy. You can change this format as and when required from specify format.

5. Logical

It consist only a single character long. That is used to store T,t,F,f,Y,y,N,n in it, which indicates that given text is logically True / False or Yes / No.

6. Memo

The memo field is special field where you can store any amount of data. This is particularly useful for storing long textual information. The data to be stored in the memo field is not stored in the data file but in an auxiliary database file with the same name but extension is ".fpt". FoxPro allots a ten byte space in the main database file to store the location of the memo data in the auxiliary database file. Memo fields are useful in reducing the size of a database file.

7. General

This type enables you to embed objects created in one windows application within FoxPro database. For instance, you can use a general field to include a complete spreadsheet, a pie chart or even a word-processed document. Through OLE (Object Linking and Embedding) feature of windows, FoxPro even updates the object in the database when the original object is changed in the other windows application.

0110. Operators

1. Arithmetic Operators

Sign	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
^ or **	Power or Exponent
( )	Grouping - (Brackets)

2. Relational Operators

Sign	Description
<	Less than
>	Grater than
<=	Less than Equal to
>=	Grater than Equal to
=	Equal to
!= or <> or #	Not Equal to
$	Sub string Comparison
==	Character String Comparison

3. Logical Operators

Sign	Description
AND	All the condition must be satisfied
OR	One of the conditions must be satisfied
NOT	Negative condition must be satisfied

4. String Operators

Sign	Description
,	Union operator to join two database fields
+	Concatenate two database fields or variable when displayed or printed

5. Literal Operators

Sign	Description
" " or ' ' or [ ]	Enclose string constants
{ }	Enclose date constants

ATMIYA

0410. Numeric Functions

Syntax: ABS(number)

Return Type: number

Description:

Returns the absolute value of the specified numeric expression.

Example:

? Abs(-45) -> 45

? Abs(10-30) -> 20

? Abs(30-10) -> 20

Syntax: Max(<expr1>, <expr2>[, <expr3> ...])

Return Type: Character, date or number

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.

Example:

? Max(54, 39, 40) -> 54

? Max(2^8, 10*12, PI( )) -> 256

? Max({07/12/99},DATE( )) -> Current Date

Syntax: **Min(<expr1>, <expr2>[, <expr3> ...])**

Return Type: Character, date or number

Description:

Min( ) (minimum) evaluates a set of numeric 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.

Example:

? Min(54, 39, 40) -> 39

? Min(2^8, 10*12, PI( )) -> 3.1416

? Min({07/12/18},DATE( )) -> 07/12/18

? Min('a','abc') -> a

? Min('a','XYZ') -> XYZ

Syntax: **Mod(<expN1>, <expN2>)**

Return Type: Numeric

Description:

Mod( ) (modulus) divides two numeric expressions and returns the remainder. Mod( ) and the % operator return identical results.

Example:

? Mod(36, 10) -> 6

? Mod(4*9, 90/9) -> 6

? Mod(25.250, 5.0) -> 0.250

Syntax: Between(<expr1>, <expr2>, <expr3>)

Return Type: Boolean

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.).

Example:

? Between(1250,1250,1260) -> .T.

? Between(1260,1250,1260) -> .T.

? Between(1261,1250,1260) -> .F.

? Between ('b','a','c') -> .T.

Syntax: Exp(<expN>)

Return Type: Numeric

Description:

The value of e, the base of natural logarithms, is approximately 2.71828.

Example:

? EXP(0) -> 1.0000

? EXP(1) -> 2.7183

Syntax: **Difference(<expC1>, <expC2>)**

Return Type: Numeric

Description:

Returns an integer, 0 through 4, representing the relative phonetic difference between two character expressions.

Example:

? Difference('Smith', 'Smyth') -> 4

? Difference('Smith', 'Stt') -> 3

? Difference('Smith', 'mtt') -> 2

? Difference('Smith', '') -> 1

Syntax: **Sign(<expN>)**

Return Type: numeric

Description:

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.

Example:

? Sign(10) -> 1

? Sign(-10) -> -1

? Sign(0) -> 0

Syntax: **Val(<expC>)**

Return Type: Numeric

Description:

Returns a numeric expression from a specified character expression composed of numbers.

Example:

? Val('12') + Val('13') -> 25

? 2 * Val('25E') -> 50

Syntax: **Int(<expN>)**

Return Type: Numeric

Description:

Int( ) returns the integer portion of the numeric expression <expN>.

Example:

? Int(12.5) -> 12

? Int(-12.5) -> -12

? Int(6.25 * 2) -> 12

Syntax: Sqrt(<expN>)

Return Type: Numeric

Description:

Returns the square root of the specified numeric expression. <expN> can't be a negative number.

Example:

? Sqrt(4) -> 2

? Sqrt(57.6*14.3) -> 28.6998

Syntax: Floor(<expN>)

Return Type: Numeric

Description:

Returns the nearest integer that is less than or equal to the specified numeric expression.

Example:

? Floor(10.9) -> 10

? Floor(-10.1) -> -11

? Floor(10.0) -> 10

? Floor(-10.0) -> -10

Syntax: **Ceiling(<expN>)**

Return Type: Numeric

Description:

Returns the nearest integer that is greater than or equal to the specified numeric expression.

Example:

? Ceiling(10.9) -> 11

? Ceiling(-10.1) -> -10

? Ceiling(10.0) -> 10

? Ceiling(-10.0) -> -10

Syntax: **Round(<expN1>, <expN2>)**

Return Type: Numeric

Description:

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.

Example:

? ROUND(1234.1962, 3) -> 1234.196

? ROUND(1234.1962, 2) -> 1234.20

? ROUND(1234.1962, 0) -> 1234

? ROUND(1234.1962, -1) -> 1230

? ROUND(1234.1962, -2) -> 1200

? ROUND(1234.1962, -3) -> 1000

ATMIYA

0410. String Functions

Syntax: Ltrim(<expC>)

Return Type: Character

Description:

Returns the specified character expression with leading blanks removed.

Example:

? Ltrim(' Computer') -> Remove space from string

Syntax: Rtrim(<expC>)

Return Type: Character

Description:

Returns the specified character expression with all trailing blanks removed.

Example:

? Rtrim('Computer ') -> Remove space from string

Syntax: Alltrim(<expC>)

Return Type: Character

Description:

Returns the specified character expression with leading and trailing blanks

removed.

Example:

? Alltrim(' Computer ') -> Remove both space from string

Syntax: **FULLPATH(<file name1>[, <expN> | <file name2>])**

Return Type: Character

Description:

Returns the MS-DOS path for a specified file or the full path relative to another file.

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.

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.

Example:

? fullpath("stud.dbf")

Syntax: **Row( )**

Return Type: Numerics

Description:

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:

? 'HELLO'

? Row() -> Row no 2.000 is displayed

Syntax: Col( )

Return Type: Numeric

Description:

Col( ) is especially useful for directing screen output to a position relative to the current col position of the cursor. The special operator $ can be used in place of the ROW( ) function.

Example:

@ 5,5 SAY ''

@ ROW(), COL()+12 SAY 'Computer'

Syntax: Soundex(<expC>)

Return Type: Character

Description:

Soundex ( ) returns a four-character string. 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.

Example:

? SOUNDEX('Smith')=SOUNDEX('aaa') -> .F.

? SOUNDEX('Smith')=SOUNDEX('smyth') -> .T.

Syntax: Empty(<expr>)

Return Type: Logical

Description:

Empty( ) returns true (.T.) if the expression <expr> is empty; false (.F.) if the expression isn't empty. Empty( ) returns true (.T.) when expressions of the following data types contain the indicated data:

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:

? Empty('hello') -> .F.

? Empty(0) -> .T.

? Empty('0') -> .F.

Syntax: **Len(<expC>)**

Return Type: Numeric

Description:

Returns the number of characters in a character expression.

Example:

? Len('computer') -> 8

Syntax: **Space(<expN>)**

Return Type: Character

Description:

Returns a character string composed of a specified number of spaces.

Example:

Store Space(15) TO blank

? Len(blank) -> 15

Syntax: **Replicate(<expC>, <expN>)**

Return Type: Character

Description:

Returns a character string that contains a specified character expression repeated a specified number of times.

Example:

? Replicate('*',5) -> *****

Syntax: **INKEY([<expN>] [, <expC>])**

Return Type: Numeric

Description:

Returns a number corresponding to the first key pressed or mouse click in the typeahead buffer.INKEY( ) returns 0 if a key isn't pressed.

<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.

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.

Syntax: **Fsize(<expC1>)**

Return Type: Numeric

Description:

Returns the size, in bytes, of a specified field.

Example:

USE STUDENT

? Fsize('cno') -> Display Field Size

Syntax: Str(<expN1>[, <expN2>[, <expN3>]])

Return Type: Character

Description:

Returns the character string equivalent to a specified numeric expression.

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.

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.

Example:

? Str(123.456,10,4) -> 123.4560

? Str(123.456,2,4) -> **

Syntax: Chr(<expN>)

Return Type: Character

Description:

CHR( ) returns a single character corresponding to a numeric expression. CHR( ) can be used to send printer control codes to a printer.

Example:

? Chr(65) -> A

? Chr(97) -> a

? Chr(7) -> Rings the bell

Syntax: Asc(<expC>)

Return Type: Numeric

Description:

Every character has a unique ASCII value in the range from 0 to 255.

Example:

? Asc('A') -> 65

? Asc('a') -> 97

Syntax: Version()

Return Type: Character

Description:

Returns a character string containing the FoxPro version number you are using.

Example:

? Version() -> FoxPro 2.6 for Windows

Syntax: Inlist(<expr1>, <expr2>[, <expr3> ...])

Return Type: Boolean

Description:

Inlist() searches for an expression in a set of expressions. Inlist () returns true (.T.) if it finds the expression in the set of expressions, else returns False (.F.). It can include up to 24 expressions and can All the expressions in the list must be of the same data type (character, numeric, logical or date).

Example:

? Inlist('b','a','z') -> .F.

? Inlist('b','a','b') -> .T.

Syntax: **Upper(<expC>)**

Return Type: Character

Description:

Each lower-case letter (az) in the character expression is converted to upper-case (AZ) in the returned string. All other characters remain unchanged.

Example:

? Upper('computer') -> COMPUTER

Syntax: **Lower(<expC>)**

Return Type: Character

Description:

Lower( ) converts all upper-case letters (AZ) in the character expression to

lower-case (az). All other characters in the character expression remain unchanged.

Example:

? Lower('COMPUTER') -> computer

Syntax: **Proper(<expC>)**

Return Type: Character

Description:

Returns the specified character expression with each word having the initial letter capitalized and the remaining characters lowercase.

Example:

? Proper('COMPUTER') -> Computer

Syntax: **Left(<expC>, <expN>)**

Return Type: Character

Description:

Returns a specified number of characters from a character expression, starting with the leftmost character.

Example:

? Left('computer',3) -> com

Syntax: **Right(<expC>, <expN>)**

Return Type: Character

Description:

Returns the specified number of rightmost characters from a character string.

Example:

? Right('computer',3) -> ter

Syntax: **Substr(<expC>, <expN1>[, <expN2>])**

Return Type: Character

Description:

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.

Example:

? Substr('computer',4,3) -> put

Syntax:

Padl(<expr>, <expN>[, <expC>])

or

Padr(<expr>, <expN>[, <expC>])

or

Padc(<expr>, <expN>[, <expC>])

Return Type: Character

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.

Example:

? PADL('TITLE', 10, '=') -> =====TITLE

? PADR('TITLE', 10, '=') -> TITLE=====

? PADC('TITLE', 10, '=') -> ==TITLE===

ATMIYA

0410. Date Functions

Syntax: Date( )

Return Type: Date

Description:

Returns the current system date, which is controlled by the operating system.

Example:

? Date()-> Display System Current date

Syntax: Time([<expN>])

Return Type: Character

Description:

Returns the current system time in 24-hour, eight-character string (HH:MM:SS) format.

Example:

? Time()->Display System Current time

Syntax: Day(<expD>)

Return Type: Numeric

Description:

Returns the numeric day of the month for a given date expression.

Example:

? Day({1/26/2001})->26

Syntax: Cdow(<expD>)

Return Type: Character

Description:

Cdow( ), the character day of the week function, returns from a date expression the name of the day of the week.

Example:

? Cdow({1/26/2001})->Friday

Syntax: Dow(<expD>)

Return Type: Character

Description:

Returns the day of the week from a given date expression. The value returned by DOW( ) ranges from 1 (Sunday) through 7 (Saturday).

Example:

? Dow({01/26/2001})->6

? Cdow({01/26/2001})->Friday

Syntax: Month(<expD>)

Return Type: Numeric

Description:

Returns the numeric month for a given date.

Example:

? Month({1/26/2001})->1

Syntax: Cmonth(<expD>)

Return Type: Character

Description:

Returns the name of the month from a given date expression.

Example:

? Cmonth({1/26/2001})->January

Syntax: Year(<expD>)

Return Type: Numeric

Description:

Year( ) always returns the year with the century. The CENTURY setting (ON or OFF) doesn't affect the returned value

Example:

? Year({1/26/2001})->2001

Syntax: **Dtoc(<expD>[, 1])**

Return Type: Character

Description:

DTOC( ) returns a character string corresponding to the date expression <expD>. It can be a date type memory variable, an array element, the name of a field, or a function that returns a date. 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.

Example:

? 'Your 90-day warranty expires ' + Dtoc({01/26/2001}+90)

Syntax: **Ctod(<expC>)**

Return Type: Date

Description:

Ctod( ), the character to date function, returns a date type value from a character expression.

Example:

? CTOD('01/26/2001')->01/26/2001

0410. Array Functions

DIMENSION <array1>(<expN1>[, expN2>]) [, <array2>(<expN3>[, <expN4>])] ...

This command Creates one- or two-dimensional arrays of memory variables.

<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)

ACOPY(<array1>, <array2>[, <expN1>[, <expN2>[, <expN3>]]])

Copies elements from one array to another array and returns numeric value.

<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.

<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

```
dime a(5)

store 0 to a

for i = 1 to 5

    a(i) = i
```

```
next

=acopy (a,b)

for i = 1 to 5

      ? b(i)

next
```

ADEL(<array>, <expN>[, 2])

Deletes an element from a one-dimensional array or a row or column from a two-dimensional array and returns numeric.

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.

Example

```
dime a(5)

store 0 to a

for i = 1 to 5

      a(i) = i

next

=acopy (a,b)

=adel(b,2)
```

```
for i = 1 to 5

    ? b(i)

next
```

AINS(<array>, <expN>[, 2])

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 toward 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.

<array>, <expN>

To insert an element into a one-dimensional array, include the array name <array> and 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>. and 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

```
dime a(5)

store 0 to a

for i = 1 to 5
```

```
    a(i) = i
next
=ains(a,3)
a(3) = 10
for i = 1 to 5
    ? a(i)
next
```

ADIR(<array>[, <expC1>[, <expC2>]])

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.

<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:

Column Information	Data Type
File names	Character

File sizes	Numeric
File dates	Date
File times	Character
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.

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.

EXAMPLE

```
dime a(5)

=aDIR(A,'README.TXT')

for i = 1 to 5

    ? a(i)

next
```

ASORT(<array>[, <expN1>[, <expN2>[, <expN3>]]])

Sorts elements in an array in ascending or descending order.

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.

<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>).

<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

```
dime a(5)

for i = 1 to 5

      a(i) = I

next

=aINS(A,3)

A(3) = 10

=ASORT(A,1,5,0)

for i = 1 to 5

      ? a(i)

next

=ASORT(A,1,5,1)

for i = 1 to 5

      ? a(i)

next
```

AFIELDS(<array>)

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.

<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.

Col # , Field Info	Data Type
1 Field names	Character
2 Field types (C D L M N F G)	Character
3 Field lengths	Numeric
4 Decimal places	Numeric

```
USE STUDENT

=aFIELD(A)

for i = 1 to 20

      ? a(I)

next
```

AELEMENT(<array>,<EXPN1>[,<EXPN2>])

Returns the number of an array element from the element's subscripts. 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.

An element can be referred by its subscripts or its element number. The commands STORE 'EBOOKMARK' TO X(1, 2) and STORE 'EBOOKMARK' TO X(3) both store the character string EBOOKMARK 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.

<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

DIME A(5)

STORE 0 TO N

for i = 1 to 5

```
    a(I) = (I*10)
next
N=AELEMENT(A,3)
? N
```

ASCAN(<array>,<EXP>[,<EXPN1>[,<EXPN2>]])

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.

<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>).

EXAMPLE

```
DIME A(5)

STORE 0 TO N

for i = 1 to 5

      a(I) = (I*10)

next

N=ASCAN(A,20)

? N
```

ASUBSCRIPT(<array>,<EXPN1>,<EXPN2>)

Returns the row or column subscript of an element from the element's number.

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 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.

<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.

EXAMPLE

```
DIME A(5,2)

STORE 0 TO k

for i = 1 to 5

        for j = 1 to 2

                a(i,j) = k

                k=k+1

        next

next

for i = 1 to 5

        for j = 1 to 2

                ?? a(i,j)
```

```
    next
    ?
next
N=ASUBSCRIPT(A,4,1)
? N
```