

RETURN TO MAIN MENU

Recital/4GL Commands

Recital/4GL Commands

Recital Corporation,
100 Cummings Center, Suite 318J
Beverly, MA 01915

Recital may have patents and/or patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents.
COPYRIGHT ©1988-2006 Recital Corporation. All rights reserved. All Recital products are trademarks or registered trademarks of Recital Corporation, Inc. Other brand and product names are trademarks or registered trademarks of their respective holders.

Last Updated August, 2006

INDEX

\ and \\	1
++	2
--	3
&&	4
*	5
//	6
:=	7
;	8
? ??	9
???	10
@	12
@...BOX	13
@...CLEAR	14
@...EDIT	15
@...FILL	17
@...GET	18
@...GET - Check Boxes	24
@...GET - Lists	27
@...GET - Popups	30
@...GET - Push Buttons	33
@...GET - Radio Buttons	36
@...GET...SPINNER	38
@...MENU	41
@...PROMPT	44
@...SAY	46
@...SAY...BITMAP	48
@...SCROLL	49
@...TO	50
ACCEPT	52
ACTIVATE MENU	53
ACTIVATE POPUP	54
ACTIVATE SCREEN	55
ACTIVATE WINDOW	56
ALIAS	57
APPEND	58
APPEND AUTOMEM	60
APPEND BLANK	61
APPEND FROM	62
APPEND FROM ARRAY	64
APPEND MEMO	65
ASSERT	66
AVERAGE	67
BEGIN SEQUENCE	68
BEGIN TRANSACTION	69
BLANK	71
BROWSE	72
BUILD	76
CALCULATE	78
CALCULATOR	80
CALENDAR	81

CANCEL	82
CHANGE	83
CLASS	86
CLASS - Methods	90
CLASS - Parameters	92
CLASS - Properties	94
CLASS - Scoping	96
CLEAR	98
CLEAR ALL	99
CLEAR AUTOMEM	100
CLEAR FCACHE	101
CLEAR GETS	102
CLEAR IOSTATS	103
CLEAR KEYS	104
CLEAR LOCKS	105
CLEAR MEMORY	106
CLEAR MENUS	107
CLEAR POPUPS	108
CLEAR PROGRAM	109
CLEAR PROMPT	110
CLEAR READ	111
CLEAR SCREEN	112
CLEAR TYPEAHEAD	113
CLEAR WINDOWS	114
CLOSE	115
CLOSE ALL	116
CLOSE ALTERNATE	117
CLOSE DATABASES	118
CLOSE FORMAT	119
CLOSE INDEX	120
CLOSE PROCEDURE	121
CLOSE TABLES	122
COMPILE	123
COMPILE DATABASE	124
CONTINUE	125
CONVERT	126
COPY DICTIONARY TO	128
COPY FILE	129
COPY INDEXES	130
COPY MEMO	131
COPY STRUCTURE	132
COPY STRUCTURE EXTENDED	133
COPY TAG	134
COPY TO	135
COPY TO ARRAY	138
COUNT	139
CREATE	140
CREATE BRIDGE	146
CREATE DICTIONARY FROM	149
CREATE FROM	151
CREATE GATEWAY	151
CREATE LABEL	152
CREATE REPORT	153

CREATE SCREEN	156
CREATE VIEW	164
DEACTIVATE MENU	166
DEACTIVATE POPUP	167
DEACTIVATE WINDOW	168
DEBUG	169
DECLARE	171
DECRYPT	173
#DEFINE	174
DEFINE BAR	175
DEFINE CLASS	176
DEFINE MENU	178
DEFINE PAD	179
DEFINE POPUP	181
DEFINE TABLE	183
DEFINE WINDOW	187
DELETE	191
DELETE FILE	192
DELETE TAG	193
DESIGN	194
DIALOG BOX	196
DIALOG FIELDS	197
DIALOG FILES LIKE	198
DIALOG GET	199
DIALOG MESSAGE	201
DIALOG QUERY	202
DIALOG SCOPE	204
DIMENSION	205
DIR	207
DISPLAY	208
DISPLAY DATABASE	210
DISPLAY DICTIONARY	212
DISPLAY FILES	213
DISPLAY HISTORY	214
DISPLAY INDEXES	215
DISPLAY MEMORY	217
DISPLAY PROCEDURE	218
DISPLAY PROTECTION	219
DISPLAY REPORT	220
DISPLAY STATUS	221
DISPLAY STRUCTURE	222
DISPLAY TABLES	223
DISPLAY TRIGGERS	224
DISPLAY USERS	225
DO	226
DO CASE	227
DO WHILE	228
EDIT	230
EJECT	233
ENCRYPT	234
END TRANSACTION	235
ENDFUNC	237
ENDPROC	238

ERASE	239
ERROR	240
EXIT	241
EXTERNAL	242
FIND	243
FLUSH	244
FOR ... NEXT	245
FUNCTION	246
GATHER	248
GENERATE	249
GOTO	250
HELP	251
HIDE MENU	252
HIDE POPUP	253
HIDE WINDOW	254
IF	256
#IF...#ENDIF	257
#IFDEF...#ENDIF	258
#INCLUDE	259
INDEX ON	260
INFO	262
INPUT	264
INSERT	265
INSTALL	267
JOIN	268
KEYBOARD	269
KEYWORD	270
LABEL	271
LINK	273
LIST	275
LIST DATABASE	277
LIST DICTIONARY	279
LIST FILES	280
LIST HISTORY	281
LIST INDEXES	282
LIST MEMORY	284
LIST PROCEDURE	285
LIST PROTECTION	286
LIST REPORT	287
LIST STATUS	288
LIST STRUCTURE	289
LIST TABLES	290
LIST TRIGGERS	291
LIST USERS	292
LOCAL	293
LOCATE	294
LOCKF	295
LOCKR	296
LOGIN	297
LOGOUT	298
LOOP	299
LPARAMETERS	300
MENU	301

MENU AT	303
MENU BAR	305
MENU BROWSE	307
MENU COMMAND	309
MENU FIELDS	310
MENU FILES LIKE	312
MENU FORMAT	314
MENU FRAME	316
MENU FROM	317
MENU QUERY	319
MENU SCOPE	322
MENU TO	324
MESSAGE	325
METHOD	326
MODIFY BRIDGE	327
MODIFY COMMAND	328
MODIFY FILE	329
MODIFY LABEL	331
MODIFY MEMO	332
MODIFY REPORT	333
MODIFY SCREEN	334
MODIFY STRUCTURE	335
MODIFY VIEW	336
MOVE WINDOW	337
NOEXIT	338
NOTE	339
ON BAR	340
ON ERROR	341
ON ESCAPE	342
ON EXIT BAR	343
ON EXIT MENU	344
ON EXIT PAD	345
ON EXIT POPUP	346
ON FINISH	347
ON KEY	348
ON MAIL	350
ON MENU	352
ON MOUSE	352
ON PAD	353
ON PAGE	354
ON POPUP	355
ON READERROR	356
ON SELECTION BAR	357
ON SELECTION MENU	358
ON SELECTION PAD	359
ON SELECTION POPUP	360
ON TERMINATION	361
ON TIMEOUT	362
PACK	363
PACK DATABASE	364
PARAMETERS	365
POP KEY	366
POP MENU	367

POP POPUP	368
PRINT	369
PRIVATE	370
PROCEDURE	371
PUBLIC	373
PUSH KEY	374
PUSH MENU	375
PUSH POPUP	376
QUERY	377
QUIT	379
READ	380
READ MENU TO	382
READ MENU BAR TO	383
RECALL	384
RECOVER	385
REINDEX	386
REINDEX DATABASE	387
RELEASE	388
RELEASE LIBRARY	389
RELEASE MENUS	390
RELEASE POPUPS	391
RELEASE WINDOWS	392
RENAME	393
REPLACE	394
REPLACE AUTOMEM	396
REPLAY	397
REPORT	398
RESET IN	401
RESIZE WINDOW	402
RESTORE	403
RESTORE COLOR	404
RESTORE GETS	405
RESTORE KEYS	406
RESTORE MENU	407
RESTORE RECORDVIEW	408
RESTORE SCREEN	409
RESTORE WINDOW	410
RESUME	411
RETRY	412
RETURN	413
ROLLBACK	414
RUN	416
SAVE COLOR	417
SAVE ERROR	418
SAVE GETS	419
SAVE KEYS	420
SAVE MENU	421
SAVE RECORDVIEW	422
SAVE SCREEN	423
SAVE TO	424
SAVE WINDOW	425
SCAN	426
SCATTER	427

SCROLL	428
SEEK	429
SELECT	430
SHOW GET	431
SHOW GETS	433
SHOW MENU	434
SHOW OBJECT	435
SHOW POPUP	436
SHOW WINDOW	437
SKIP	438
SLEEP	439
SORT	440
SPAWN	441
STORE	442
STORE AUTOMEM	444
SUM	445
SUSPEND	447
TEXT	448
TOTAL	450
TREPORT	452
TRY...ENDTRY	458
TYPE	461
#UNDEF	462
UNLOCK	463
UPDATE	464
USE	465
WAIT	469
WITH...ENDWITH	470
ZAP	471

Commands by Category

Applications

\\	&&	*
//	:=	;
ALIAS	CANCEL	CLEAR PROGRAM
CLOSE PROCEDURE	COMPILE	CONTINUE
DISPLAY PROCEDURE	DO	DO CASE
DO WHILE	ENDFUNC	ENDPROC
EXIT	FOR...NEXT	FUNCTION
IF	#IF...#ENDIF	#IF...#ENDIF
#INCLUDE	KEYWORD	LINK
LIST PROCEDURE	LOOP	LPARAMETERS
METHOD	NOTE	ON ERROR
PARAMETERS	PROCEDURE	QUIT
RELEASE LIBRARY	RETURN	SCAN
SLEEP		

Array Processing

DECLARE	DIMENSION	GATHER
SCATTER		

Data Connectivity

CREATE BRIDGE	CREATE GATEWAY	LOGIN
LOGOUT	MODIFY BRIDGE	MODIFY GATEWAY

Databases

CLOSE DATABASES	CLOSE TABLES	COMPILE DATABASE
DISPLAY DATABASE	DISPLAY TABLES	LIST DATABASE
LIST TABLES	PACK DATABASE	REINDEX DATABASE

DES3 Encryption

DECRYPT	ENCRYPT	
---------	---------	--

Disk and File Utilities

COPY FILE	COPY TO	DELETE FILE
DIR	DISPLAY FILES	ERASE
LIST FILES	RENAME	RUN
SPAWN		

Environment

CLEAR ALL	DISPLAY MEMORY	DISPLAY STATUS
DISPLAY TRIGGERS	DISPLAY USERS	LIST MEMORY
LIST STATUS	LIST TRIGGERS	LIST USERS
ON TERMINATION		

Error Handling and Debugging

ASSERT	BEGIN SEQUENCE	DEBUG
DISPLAY HISTORY	ERROR	LIST HISTORY
ON FINISH	RESUME	RETRY
SAVE ERROR	SUSPEND	TRY

Fields and Records

APPEND BLANK	APPEND FROM ARRAY	AVERAGE
BLANK	CALCULATE	COPY TO ARRAY
COUNT	DELETE	DISPLAY
GOTO	LIST	LOCATE
PACK	RECALL	REPLACE
RESTORE RECORDVIEW	SAVE RECORDVIEW	SKIP
SORT	SUM	TOTAL
UPDATE	ZAP	

Information Center

DESIGN	INFO	
--------	------	--

Indexing

CLOSE INDEX	COPY INDEXES	COPY TAG
DELETE TAG	DISPLAY INDEXES	FIND
INDEX ON	LIST INDEXES	REINDEX
SEEK		

Input / Output

? ??	@	@...SAY
@...SAY...BITMAP	CLOSE ALTERNATE	INPUT
LABEL	ON MAIL	REPORT
TEXT	TYPE	

Keyboard Events

CLEAR KEYS	CLEAR TYPEAHEAD	KEYBOARD
ON ESCAPE	ON KEY	ON TIMEOUT
POP KEY	PUSH KEY	REPLAY
RESTORE KEYS	SAVE KEYS	WAIT

Manual Locking

CLEAR LOCKS	LOCKF	LOCKR
UNLOCK		

Memory Variables

++	--	APPEND AUTOMEM
CLEAR AUTOMEM	CLEAR MEMORY	#DEFINE
LOCAL	PRIVATE	PUBLIC
RELEASE	REPLACE AUTOMEM	RESTORE
SAVE TO	STORE	STORE AUTOMEM

Memos

APPEND MEMO	COPY MEMO	MODIFY MEMO
-------------	-----------	-------------

Menus

@...MENU	@...PROMPT	ACTIVATE MENU
ACTIVATE POPUP	CLEAR MENU	CLEAR POPUPS
CLEAR PROMPT	DEACTIVATE MENU	DEACTIVATE POPUP
DEFINE BAR	DEFINE MENU	DEFINE PAD
DEFINE POPUP	DEFINE TABLE	HELP
HIDE MENU	HIDE POPUP	MENU
MENU AT	MENU BAR	MENU BROWSE
MENU COMMAND	MENU FIELDS	MENU FILES LIKE
MENU FORMAT	MENU FRAME	MENU FROM
MENU QUERY	MENU SCOPE	MENU TO
NOEXIT	ON BAR	ON EXIT BAR
ON EXIT MENU	ON EXIT PAD	ON EXIT POPUP
ON MENU	ON PAD	ON POPUP
ON SELECTION BAR	ON SELECTION MENU	ON SELECTION PAD
ON SELECTION POPUP	POP MENU	POP POPUP
PUSH MENU	PUSH POPUP	READ MENU TO
READ MENU BAR TO	RELEASE MENUS	RELEASE POPUPS
RESTORE MENU	SAVE MENU	SHOW MENU
SHOW POPUP		

Objects

CLASS	CLASS – METHODS	CLASS – PARAMETERS
CLASS – PROPERTIES	CLASS – SCOPING	DEFINE CLASS
WITH...ENDWITH		

Performance and Optimization

CLEAR FCACHE	CLEAR IOSTATS	
--------------	---------------	--

Printing

???	EJECT	ON PAGE
PRINT		

Screen Dialogs

CALCULATOR	CALENDAR	DIALOG BOX
DIALOG FIELDS	DIALOG FILES LIKE	DIALOG GET
DIALOG MESSAGE	DIALOG QUERY	DIALOG SCOPE

Screen Forms

@...BOX	@...CLEAR	@...EDIT
@...FILL	@...GET	@...GET – CHECK BOXES
@...GET – LISTS	@...GET – POPUPS	@...GET – PUSH BUTTONS
@...GET – RADIO BUTTONS	@...GET...SPINNER	@...SCROLL
@...TO	ACCEPT	ACTIVATE SCREEN
APPEND	BROWSE	CHANGE
CLEAR	CLEAR GETS	CLEAR READ
CLEAR SCREEN	CLOSE FORMAT	EDIT
FLUSH	INSERT	MESSAGE
ON READERROR	QUERY	READ
RESTORE COLOR	RESTORE GETS	RESTORE SCREEN
SAVE COLOR	SAVE GETS	SAVE SCREEN
SCROLL	SHOW GET	SHOW GETS
SHOW OBJECT		

Screen Windows

ACTIVATE WINDOW	CLEAR WINDOWS	DEACTIVATE WINDOW
DEFINE WINDOW	HIDE WINDOW	MOVE WINDOW
RELEASE WINDOWS	RESIZE WINDOW	RESTORE WINDOW
SAVE WINDOW	SHOW WINDOW	

Table Basics

APPEND FROM	BUILD	CLOSE
CLOSE ALL	CLOSE DATABASES	CONVERT
COPY DICTIONARY TO	COPY STRUCTURE	COPY STRUCTURE EXTENDED
CREATE DICTIONARY FROM	CREATE FROM	DISPLAY DICTIONARY
DISPLAY PROTECTION	DISPLAY STRUCTURE	GENERATE
INSTALL	JOIN	LIST DICTIONARY
LIST PROTECTION	LIST STRUCTURE	RECOVER
SELECT	USE	

Terminal Developer Development Tools

CREATE	CREATE BRIDGE	CREATE GATEWAY
CREATE LABEL	CREATE REPORT	CREATE SCREEN
CREATE VIEW	ED	MODIFY BRIDGE
MODIFY COMMAND	MODIFY FILE	MODIFY GATEWAY
MODIFY LABEL	MODIFY REPORT	MODIFY SCREEN
MODIFY STRUCTURE	MODIFY VIEW	VI

Transaction Processing

BEGIN TRANSACTION	RESET IN	ROLLBACK
-------------------	----------	----------

Xbase Compatibility

EXTERNAL		
----------	--	--

\ | \

Class

Applications

Purpose

Output lines of text

Syntax

\ <text> | \ \ <text>

See Also

SET TEXTMERGE TO, SET TEXTMERGE DELIMITERS, SET TEXTMERGE ON|OFF, TEXT...ENDTEXT, TREPORT

Description

The \ and \ \ commands are used to output lines of text to the screen, to a file, or to window. <text> may also include expressions which are bracketed by < and >. These expressions will be evaluated before the text is output.

Any spaces, text and delimited expressions following the \ command are preceded by a new line character in UNIX and a carriage return/line feed pair in OpenVMS. The <text> line is then output. Any spaces, text and delimited expressions following the \ \ command are not preceded by an end of line marker. The <text> line is then output. Delimited expressions may include table field names, memory variables, and functions. The default text delimiters are < and >.

The SET TEXTMERGE DELIMITERS command may be used to change this default. The SET TEXTMERGE TO command is used to merge text to a file, or a window. The SET TEXTMERGE ON|OFF command determines whether delimited expressions will be evaluated, or output literally. Text is output when it is placed after the \ and \ \ commands, or when it is between the TEXT...ENDTEXT commands.

Example

```
use test
set textmerge to balance.txt
set textmerge to delimiters to “{“,“}”
set textmerge on
go top
do while balance > 0
    \Date: {{date()}}
    \Name: {{last_name}} {{first_name}}
    \
    \Account Number : {{account_no}}
    \Current Balance : {{balance}}
    \Credit Limit:    : {{limit}}
    \
enddo
set textmerge off
set textmerge to
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

++

Class

Memory Variables

Purpose

Increment memory variable by one

Syntax

++ <memvar>

See Also

--, STORE, PRIVATE, PUBLIC, PARAMETERS, SUM, COUNT, AVERAGE

Description

The ++ command is used to automatically increment a previously declared numeric memory variable by one. The ++ command must be placed at the beginning of the command line.

Example

```
i=0
do while i <100
  ++ i
enddo
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

--

Class

Memory Variables

Purpose

Decrement memory variable by one

Syntax

-- <memvar>

See Also

++, STORE, PRIVATE, PUBLIC, PARAMETERS, SUM, COUNT, AVERAGE

Description

The -- command is used to automatically decrement a previously declared numeric memory variable by one. The -- command must be placed at the beginning of the command line.

Example

```
i=100
do while i > 0
  --I
enddo
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

&&

Class

Applications

Purpose

Add comments to a program to improve its readability and maintainability

Syntax

&& <expC>

See Also

*, // NOTE

Description

The && command allows all characters following it on a line to be treated as a comment and to be ignored by the Recital/4GL. The && command differs from the NOTE command, and the * command, in that it can be placed anywhere on a line, even following an executable command.

Example

&& open the table

use patrons index names && view in name order

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

Class

Applications

Purpose

Comment line

Syntax

* <expC>

See Also

&&, NOTE, //

Description

The * command allows comment lines to be inserted in programs to enhance their readability and maintainability. The '*' character must be the first non-space character on the line. The '*' command differs from the && command in that comments that commence with && can be placed on the same line as an executable command whereas those beginning with * cannot.

Example

open the table
use patrons index names

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

//

Class

Applications

Purpose

Comment line

Syntax

// <expC>

See Also

* &&, NOTE

Description

The // command allows comment lines to be inserted in programs to enhance their readability and maintainability. Like the && command, the // command allows all characters following it on a line, to be treated as a comment and to be ignored by the Recital/4GL. The // command differs from the NOTE command, and the * command, in that it can be placed anywhere on a line, even following an executable command.

Example

// open the table

use patrons index names // view in name order

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

:=

Class

Applications

Purpose

Inline assignment

Syntax

<alias>-><field> | [m->]<memvar> := <exp>

See Also

REPLACE, STORE

Description

The := inline assignment operator is used to assign a value to a field or memory variable. Assignment can also be achieved using the STORE command or '=' operator in the case of memory variables and the REPLACE command for fields. The tables alias name and alias operator, '->' must be specified when assigning a value to a field. If these are missing, the target is assumed to be a memory variable and is created as a PRIVATE memory variable if it does not already exist. The 'm->' memory alias pointer can be specified, but is not required.

Example

```
use names
3
? "mid_name=",mid_name, ""
//Create memvar 'mid_name'
mid_name := "James"
replace names->mid_name with ""
//Assign value to field 'mid_name'
names->mid_name := "James"
m_name := "James"
? "mid_name=",mid_name, ""
? "mid_name=",m->mid_name, ""
? "m_name=",m_name, ""
?
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

;

Class

Applications

Purpose

Command separator /continuation character

Syntax

<command> [; <command>]

See Also

*, //, &&, NOTE

Description

Multiple commands can be placed on a line by separating each command with a ';' character. The ';' character may also be used to continue a literal character string to the next line. When the ';' character is used in this way, a space is added to the string. If ';' is the last character on a line, it can be used as a continuation character to extend commands over more than one line.

Example

```
// Used as a command separator
if deleted()
    dialog box "Record deleted."; clear; return
else
    dialog box "Record not deleted."
endif
// Used as a continuation character
menu browse accounts->name;
    label "Account Names"
// Used in a character string
dialog message "That file does not exist. Do you wish to;
    continue?"
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

? | ??

Class

Input/Output

Purpose

Evaluate expression and output result

Syntax

? <exp> [,<exp>] | ?? <exp> [,<exp>]

[AT <expN>]

[FUNCTION <expC1>]

[PICTURE <expC2>]

[STYLE <expC3>]

See Also

@...GET, @...SAY, ACCEPT, INPUT, WAIT, SET SPACE, TRANSFORM()

Description

The ? command outputs a carriage return, then a line feed, then evaluates each expression in turn and displays the result. The ?? command evaluates the expression and displays the results on the same line, no carriage return/line feed sequence is output. If more than one expression is specified and SET SPACE is ON, a single space is output between each expression. If SET SPACE is OFF, the expressions are output with no space in between.

Clause	Description
AT <expN>	You may optionally specify the column at which the expression will be output with the AT <expN> clause.
PICTURE <expC1>	The PICTURE clause supports all picture template symbols listed in the @...SAY command.
FUNCTION <expC2>	The FUNCTION qualifier allows picture functions to be specified. Normally this can be done with the PICTURE qualifier by preceding the picture with '@'. If the FUNCTION qualifier is used the '@' is not needed.
STYLE <expC3>	The STYLE qualifier has been added for compatibility with other Xbase languages.

In Recital Mirage, the following style qualifiers are supported in <expC3>:

<expC3>	Style
B	Bold
I	Italic
U	Underline
-	Strikeout

Example

? "price" at 01 picture "@!",1234.56 picture "\$99,999.99"

PRICE \$1,234.56

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

???

Class

Printing

Purpose

Evaluate expression and output result to printer

Syntax

??? <expC>

See Also

?, ??, SET DEVICE TO PRINT, PRINT, SET PRINTER ON, SET PRINT TO \\SPOOLER

Description

The ??? command evaluates the specified expression and sends the output to the printer. This command is particularly useful for sending initialization strings to a printer. Initialization strings contain control characters that prescribe certain printing attributes. If your printer driver does not support a certain printing capability, the ??? command can be used to circumvent the driver and send printer control codes directly to the printer. Your printer manual contains a table of control codes and the correct use of each. Printer control codes may contain any printable character with the exception of the double quote mark "".

Printable character codes may be expressed in a variety of ways, however strings that name non-printable characters must include curly braces {}. The <expC> can contain directives that are translated from the following table:

ASCII Code	Control Character Specification
0	{NULL} or {CTRL @}
1	{CTRL-A}
2	{CTRL-B}
3	{CTRL-C}
4	{CTRL-D}
5	{CTRL-E}
6	{CTRL-F}
7	{BELL} or {CTRL-G}
8	{BACKSPACE} go {CTRL-H}
9	{TAB} or {CTRL-I}
10	{LINEFEED} or {CTRL-J}
11	{CTRL-K}
12	{CTRL-L}
13	{RETURN} or {CTRL-M}
14	{CTRL-N}
15	{CTRL-O}
16	{CTRL-P}
17	{CTRL-Q}
18	{CTRL-R}
19	{CTRL-S}
20	{CTRL-T}
21	{CTRL-U}
22	{CTRL-V}
23	{CTRL-W}
24	{CTRL-X}
25	{CTRL-Y}
26	{CTRL-Z}

ASCII Code	Control Character Specification
27	{ESCAPE} or {ESC} or {CTRL-[}
28	{CTRL-\}
29	{CTRL-]}
30	{CTRL-^}
31	{CTRL-_}
32	{DEL} or {DELETE}

Example

??? "{ESCAPE}24"

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

@

Class

Input/Output

Purpose

Position the cursor on the screen and clear to the end of the line

Syntax

@<expN1>,<expN2>

See Also

@...BOX, @...CLEAR, @...FILL, @...GET, @...MENU, @...PROMPT, @...SAY, @...SCROLL, @...TO, SET ALTERNATE, SET DEVICE, CREATE REPORT, TREPORT, ROW(), COL(), PROW(), PCOL()

Description

The @ command is used to position the cursor at the specified row, <expN1> and column, <expN2> on the screen, and then clear to the end of the row. Rows on the screen are addressable from 0 to the height of the screen display, and columns are addressable from 0 to 79. If SET DEVICE TO PRINT has been issued, any range is addressable and the print head of the printer is positioned at the row, column specified.

Example

@0,0

@i+ 1, j

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

@...BOX

Class

Screen Forms

Purpose

Draw a box using the specified coordinates

Syntax

@<expN1>,<expN2>,<expN3>,<expN4> BOX [<expC>]

See Also

@...CLEAR, @...FILL, @...GET, @...MENU, @...PROMPT, @...SAY, @...SCROLL, @...TO

Description

This command can be used to draw a box on the screen, or on a user-defined window. <expN1> and <expN2> define the row and column coordinates of the upper left-hand corner of the box, and <expN3> and <expN4> define the row and column coordinates of the lower right hand corner of the box. If <expN1> and <expN3> contain the same value, a horizontal line will be drawn. If <expN2> and <expN4> are the same value, a vertical line will be drawn.

The optional argument <expC> defines the characters to fill the box. Up to nine characters may be used and if fewer than nine characters are used, a partial box will be drawn. The first character of the <expC> fills the top left corner of the box. The second character of the <expC> fills the top row of the box. The third character fills the top right corner of the box. The fourth character fills the right side of the box. The fifth character fills the bottom right corner of the box. The sixth character fills the bottom row of the box. The seventh character fills the bottom left corner of the box. The eighth character fills the left side of the box, and the ninth character fills the remaining center of the box.

Example

@0,0,10,10 box "123456789"

```
1222222223
8999999994
8999999994
8999999994
8999999994
8999999994
8999999994
8999999994
8999999994
8999999994
7666666665
```

Products

Recital Mirage Server, Recital Terminal Developer

@...CLEAR

Class

Screen Forms

Purpose

Position the cursor on the screen and clear to the end of the screen

Syntax

@<expN1>,<expN2> CLEAR

@<expN1>,<expN2> CLEAR TO <expN3>,<expN4> [BOLD] [REVERSE]

See Also

@...BOX, @...FILL, @...GET, @...MENU, @...PROMPT, @...SAY, @...SCROLL, @...TO, ROW(), COL()

Description

The @...CLEAR command positions the cursor at the specified row <expN1> and column <expN2> on the screen and then clears the rest of the screen or window from that point downwards.

The @...CLEAR TO command clears a rectangular area of the screen or window, whose upper-left corner is at <expN1> and <expN2> and bottom left corner is at <expN3> and <expN4>. If the REVERSE keyword is specified, the area is displayed in reverse video. If the BOLD keyword is specified, the area is displayed in highlighted reverse video.

Example

@10,0 clear

Products

Recital Mirage Server, Recital Terminal Developer

@...EDIT

Class

Screen Forms

Purpose

Create a FoxPro style text-editing region for data input

Syntax

```
@<expN1>,<expN2> EDIT <memvar> | <field>  
SIZE <expN3>,<expN4> [,<expN5>]  
[COLOR SCHEME <expN6> | COLOR <expC1>]  
[DEFAULT <exp>]  
[ENABLE | DISABLE]  
[FONT <expC2>]  
[FUNCTION <expC3>]  
[MESSAGE <expC4>]  
[NOMODIFY]  
[SCROLL]  
[STYLE <expC5>]  
[TAB]  
[VALID <expL1> [ERROR <expC6>]]  
[WHEN <expL2>]
```

See Also

@...GET, @...SAY, APPEND, CHANGE, EDIT, SET COMPATIBLE, SET FILETYPE, READ

Description

The @...EDIT command can be used to create a FoxPro style text-editing region. The @...EDIT command can be used to edit a memory variable, a memo field or a character field. The text-editing region is displayed at the row and column coordinates specified by <expN1> and <expN2>.

SIZE <expN3>,<expN4> [,<expN5>]

The SIZE <expN3>,<expN4> clause defines the size of the text-editing region. <expN3> is the height and <expN4> is the width. The SIZE clause must be used. The optional <expN5> specifies the number of characters that can be edited.

COLOR SCHEME <expN6> | COLOR <expC1>

The COLOR SCHEME | COLOR clause is used to define the foreground and background colors of the edit region. The <expN6> is the number of a color scheme. The <expC1> is a color pair in the format foreground / background.

DEFAULT <exp>

The DEFAULT clause is used to specify a default value for the variable being edited in the edit region.

ENABLE | DISABLE

The DISABLE keyword can be included to prevent the edit region from being selected. By default, edit regions are enabled.

FONT <expC2>

The FONT clause specifies the font to be used.

FUNCTION <expC3>

The FUNCTION clause determines the text justification. The following options are supported in <expC3>:

<expC3>	Style
I	Centers text.
J	Right-justifies text.

MESSAGE <expC4>

The MESSAGE clause is used to specify a text message, <expC4>, of up to 80 characters, which will be displayed in the message line when the edit region is selected.

NOMODIFY

The NOMODIFY keyword specifies that the edit region can be displayed, but the contents cannot be edited.

SCROLL

If the SCROLL keyword is included, a scroll bar will be displayed on the edit region when the text exceeds the displayed size.

STYLE <expC5>

The STYLE clause defines the font style. The following style qualifiers are supported in <expC5>:

<expC5>	Style
B	Bold
I	Italic
U	Underline
-	Strikeout

VALID <expL1> [ERROR <expC6>]

If the VALID clause is included and changes are made to the text in the edit region, <expL1> must evaluate to .T. (true) before the changes are saved. The optional ERROR clause causes the character expression <expC6> to be displayed when <expL1> evaluates to .F. (false).

TAB

If the TAB keyword is included, tabs can be inserted into the edit region text using the tab key. The default behavior is for the tab key to cause the cursor to move to the next control.

WHEN <expL2>

The text-editing region can only be used when the <expL2> condition evaluates to true (.T.).

Example

```
@01,01 edit customer->notes;
    size 3,10
read
```

Products

Recital Mirage Server, Recital Terminal Developer

@...FILL

Class

Screen Forms

Purpose

Change the colors of a specified region

Syntax

@<expN1>,<expN2> FILL TO <expN3>,<expN4>[COLOR <color>]

See Also

@...BOX, @...CLEAR, @...GET, @...MENU, @...PROMPT, @...SAY, @...SCROLL, @...TO, SET COLOR, SAVE COLOR, RESTORE COLOR, ISCOLOR(), SETCOLOR()

Description

The @...FILL command changes the colors of the text in the specified region of the screen, or window. Only the standard foreground and background colors in the area can be changed. This command affects the display already on the screen and any subsequent commands that write to this area will use the default colors, not the colors set with the @...FILL command. If the COLOR <color> option is omitted, the @...FILL command clears the specified region of the screen. If SET DEVICE TO PRINT is in effect, then this command is ignored.

The region on the screen is specified using <expN1> and <expN2> as the top left corner row and column coordinates, and <expN3>, <expN4> as the bottom right corner row and column coordinates.

Clause	Description
COLOR <color>	The <color> is a color pair. 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.

Example

@05,00 fill to 08,09 color gr+/b

Products

Recital Mirage Server, Recital Terminal Developer

@...GET

Class

Screen Forms

Purpose

Data entry

Syntax

@ <expN1>,<expN2> GET <memvar> | <field>
[BUTTON <expC1>
 [LABEL <expC2>
 [GROUP <expC3>
 [HELP <expC4>
 [TRIGGER <expC5>]]
[CALCULATE]
[CALCULATED BY <expr1>]
[CHOICELIST <expC6>
[COLOR <color> | COLOR SCHEME <expN3>]
[DEFAULT <expr2>]
[DISABLE | ENABLE]
[ERROR <expC7>]
[FONT <expC8>[, <expN4>]]
[FUNCTION <expC9> | PICTURE <expC10>]
[HELP <expC11> | MESSAGE <expC12>]
[LOOKUP IN <alias>]
[MUST_ENTER]
[NOECHO]
[NAME <expC13>]
[POSTFIELD <program | procedure>]
[PREFIELD <program | procedure>]
[PROPERTIES <expC14>]
[RANGE <expr3>,<expr4>]
[READ_ONLY [IF <expL1>] | WHEN <expL2>]
[RELATION [INTO <alias>]
[VALID <expL3> | <expN5> | IN <expC14> | VALIDATE WITH <program | procedure>]

See Also

@...BOX, @...CLEAR, @...FILL, @...MENU, @...PROMPT, @...SAY, @...SCROLL, @...TO, CLEAR GETS, SHOW GETS, SAVE GETS, FUNCTION, SET CONFIRM, SET FORMAT, APPEND, CHANGE, CREATE SCREEN, EDIT, INSERT, ROW(), COL(), MENU AT, SET UPDATE, SET QUERYMODE, READ, USE

Description

The @...GET command is used to create an editing region for the contents of a memory variable, array element or field. @...SAY and @...GET can be combined into a single command. If both the SAY and GET are combined into single statement, then specify only one set of screen coordinates where the @...SAY output begins. The @...GET editing region will be placed one character after the end of the @...SAY.

The Recital Forms Designer can be used to create data entry forms containing @...GET commands. See CREATE SCREEN for full details.

When a READ command is issued, all GETS which have been processed since the last READ or CLEAR GETS command become active.

<expN1>,<expN2>

The <expN1>, <expN2> are numeric expressions specifying the row and column coordinates. Row numbers start at 0 and are incremented from top to bottom. Column numbers start at 0 and are incremented from left to right.

<memvar> | <field>

@...GET creates an editing region for the memory variable or array element specified in <memvar> or the field specified in <field>. Fields in open tables in other (not currently selected) workareas must be prefixed with an alias pointer. An alias pointer is an alias name followed by the symbol '.' or '->'. Fields have precedence over memory variables with the same name. Memory variables having the same name as a field in the active table file must be prefixed by a memory variable alias pointer: 'm.' or 'm->'.

@...GET <memvar> | <field> BUTTON <expC1>

[LABEL <expC2>]

[GROUP <expC3>]

[HELP <expC4>]

[TRIGGER <expC5>]

The optional BUTTON keyword is used to define the <memvar> | <field> as a check box or radio button.

Check boxes correspond to logical fields or memory variables and can be selected or deselected using the [SPACEBAR]. When a check box is selected, its corresponding <memvar> or <field> is set to true (.T.). When it is deselected, its corresponding <memvar> or <field> is set to False (.F.). The check box edit region is represented by square brackets: []. When a check box is deselected, the brackets are empty, when the check box is selected an asterisk is displayed inside the brackets.

Radio buttons form groups, whereby each radio button corresponds to a possible value <expC1> for a character field or memory variable. Only one radio button may be selected at any one time, since this signifies the <memvar> or <field> value. The radio button edit region is represented by round brackets: (). When a radio button is deselected, the brackets are empty, when the radio button is selected, an asterisk is displayed inside the brackets.

The optional LABEL <expC2> keyword specifies a label for the check box or radio button. If no LABEL is specified, the character expression specified in the BUTTON clause is used.

The GROUP <expC3> keyword is used to group radio buttons. Each radio button that you specify in a group of radio buttons must be given the same GROUP <expC3>.

The optional HELP <expC4> keyword specifies a message that will appear in the message line of the screen when the cursor moves onto the check box or radio button.

The optional TRIGGER <expC5> keyword specifies the name of an event driven procedure to be called when the button is selected or deselected. The TRIGGER procedure is called with the following three parameters:

Parameter	Check Box	Radio Button
<group name>	Null string	The name of the group as specified in the GROUP clause.
<button name>	The name of the button as specified in the BUTTON clause.	The name of the button as specified in the BUTTON clause.
<value>	.T. if selected, .F. if not selected.	.T. if selected, .F. if not selected.

CALCULATE

Whenever data is input into an @...GET, which has CALCULATE specified, all of the CALCULATED BY <expr> @...GETS are recalculated and redisplayed.

CALCULATED BY <expr1>

Used to calculate and display the expression in the @...GET edit region. The expression is calculated when the @...GET is first activated, or when data is entered in another active GET which has the CALCULATE clause. Any valid expression may be used (including User Defined Functions). The CALCULATED BY clause makes the @...GET read only.

CHOICELIST <expC6>

The optional CHOICELIST <expC6> keyword allows a pop-up choicelist or a User Defined Function (UDF) to be associated with an @...GET when the [HELP] key is pressed. The CHOICELIST <expC6> can be in one of the following three forms:

A static choicelist made up of a series of character expressions separated by commas: <expCi> [,<expCi>].... Each set of characters between the commas becomes a separate menu option. This choicelist is non-scrollable and can take up to 19 options.

A dynamic choicelist made up of records from tables other than the current table: "@<alias>,<expC>". The <alias> can refer to a workarea or to a table in the current directory. If the table is not opened it will automatically be opened and then closed again once a selection has been made. The <expC> must include a reference to at least one field in the <alias> table. The <expC> can also include literal values and alias pointers to related tables. The dynamic choicelist is scrollable with the [NEXT SCREEN], [PREVIOUS SCREEN] and [ARROW] keys. The [FIND] key can also be used to find menu items in the <alias> table being browsed. When the [FIND] key is selected, a popup DIALOG GET box is displayed, allowing the search string to be entered. If the <alias> table is indexed, a SEEK will be performed. If not, a LOCATE FOR will be performed. A filter condition may be defined on the <alias> table specified in the choicelist to limit the records that will appear as menu items.

A UDF can be used to return a character expression to be placed in the field. To specify a UDF to execute when the [HELP] key is pressed, the UDF name is prefixed with a question mark (?). For example:

```
@10,3 get account_name;  
choicelist "?helpproc()"
```

The CHOICELIST option can only be used on character fields/variables. There are default help objects on dates (calendar), numerics (calculator), memos (editor) and logicals (fixed choicelist). If a popup choicelist is required on non-character fields/variables, this can be achieved using the VALIDATE WITH clause in combination with one of the many MENU commands. If the VALIDATE WITH option is used, the CHOICELIST option will be ignored. The CHOICELIST specified on an @...GET will override any pre-defined Applications Data Dictionary choicelist.

COLOR <color code> | COLOR SCHEME <expN3>

The color of @...GET output can be specified by including a set of color pairs in the COLOR clause or specifying a numeric COLOR SCHEME <expN3>. If you do not specify the COLOR | COLOR SCHEME clause then the @...GET is displayed in the default Color of Fields. The color that is specified overrides the SET COLOR command, but only for the output of the current @...GET command.

DEFAULT <expr2>

The DEFAULT clause is included for FoxPro compatibility.

DISABLE | ENABLE

If the DISABLE clause is included, the GET is not active and cannot be selected or modified. GETs are enabled by default. READ will exit immediately if all GETs are marked DISABLE.

ERROR <expC7>

The optional ERROR <expC7> keyword causes an error message to be displayed if LOOKUP IN, RELATION, VALIDATE WITH or the VALID <condition> fail. The error message <expC7> is displayed in an ALERT message box, and can be a maximum of 80 characters in length.

FONT <expC8>[, <expN4>]

The name of the font is specified in <expC8> and, optionally, the font size in <expN4>.

FUNCTION | PICTURE

The optional PICTURE or FUNCTION clause allows for specialized formatting to be performed on the data before it is input or output. It causes each individual character entered in the field to be validated as it is entered.

FUNCTION codes can be included in a PICTURE clause. In this case, the PICTURE clause must start with @. Since a FUNCTION clause affects the entire expression, it can only contain FUNCTION codes.

Function Code	Description
A	Allows only letters (A-Z, a-z) to be entered.
B	Left justifies a number.
D	Displays dates in the current SET DATE format.
E	Displays dates in European format dd/mm/yy
I	Centers the text.
J	Right justifies the text.
K	Selects the entire field for editing when it gets focus.
L	Displays leading zeros in numeric output.
R	Displays literal characters in the template, but does not store them in the data.
S<n>	Allows horizontal scrolling of a wide character field.
T	Trims leading and trailing blanks.
Z	Replaces leading zeroes in a number with spaces.
!	Converts all alphabetic characters entered in the field to upper-case

A PICTURE template can include any characters, but only valid picture codes affect the editing, any other value is stored in the data. Each Picture Code applies to the character at that position only, not to the entire input.

Picture Code	Description
A	Allows lower-case or uppercase letters only to be entered.
L	Allows only logical data to be entered, (Y, N, T, or F).
N	Allows letters and digits only.
U	Allows only A-Z to be entered and automatically converted to upper case.
X	Allows any character to be entered.
Y	Allows only Y or N to be entered.
#	Allows digits (0-9), '.', '+', or '-' to be entered
!	Converts any characters entered to upper case.
9	Allows digits (0-9), '.', '+', or '-' to be entered.
.	Specifies the position of a decimal point in a number.
,	Displays commas in 'thousands' places if the number is large enough.
\$	Displays leading \$ in front of numbers instead of spaces.
*	Displays leading * in front of numbers instead of spaces.
Others	Other characters included in the picture for character fields are added to the field.

HELP <expC11> | MESSAGE <expC12>

The HELP and MESSAGE clauses are synonymous. The character expression <expC11> or <expC12>, of up to 80 characters in length, is displayed in the message line when the @...GET has focus.

LOOKUP IN <alias>

The LOOKUP IN clause defines a validity check using a cross-table lookup. The alias name of the lookup table must be specified as <alias>. Changing the field value causes the lookup table to be scanned for the new value. If the value does not exist in the lookup table, a validation failed error message is displayed. See ERROR for user-defined error messages.

MUST_ENTER

The MUST_ENTER clause is used to enforce data entry in empty GETS. You cannot commit any changes until the GET with a MUST_ENTER clause has had data entered into it.

Data Type	Empty when
Character	No text entered
Numeric	Equal to zero
Logical	Equal to .F.
Date	No data entered
Memo	No text entered

NAME <expC13> (Recital Mirage only)

The NAME clause is used to create a 'wrapper' object for the GET. The <expC13> must be a unique name for the current READ. Once defined, the NAME clause allows properties to be set and methods to be called on the client for that GET. For additional information on object properties and methods, please see the Recital Mirage documentation.

NOECHO

The NOECHO option is used to disable the echoing of the input characters. It is particularly useful for entering passwords. As each character is entered, "*" is displayed.

POSTFIELD <program | procedure>

The POSTFIELD clause is used to specify a procedure name for the POSTFIELD trigger. The POSTFIELD trigger procedure is called as the field is exited.

PREFIELD <program | procedure>

The PREFIELD clause is used to specify a procedure name for the PREFIELD trigger. The PREFIELD trigger procedure is called as the field is entered.

PROPERTIES <expC14> (Recital Mirage only)

The PROPERTIES clause can be used to specify properties for the specified get. For information on the available properties, please see the Recital Mirage documentation.

RANGE<expr3>,<expr4>

The RANGE option can be specified for numeric or date fields. The lowest value is represented by <expr3> and the highest value by <expr4>. Input data is checked to verify that it lies within the specified range. If it is out of range, a message will be displayed in the message line.

READ_ONLY [IF <expL1>] | WHEN <expL2>

The READ_ONLY option disables editing of the specified GET field. When this option is used, the contents of the field will be redisplayed as each new record is displayed on the screen. The optional IF <expL1> can be used to make the field READ_ONLY when the specified <expL1> is true (.T.). If the <expL1> evaluates to false (.F.), the field can be edited. The WHEN <expL2> has the same effect, but the opposite syntax. It is used to make the field read only when the specified <expL2> is false (.F.).

RELATION [INTO <alias>]

The RELATION option causes the data entered in the field to be used as a key field for a related table. The key is searched for in the master index of the specified table, and the record associated with that key read. The optional INTO <alias> clause specifies the target table. The INTO <alias> is not needed if relationships have already been established with the SET RELATION command. When the target table has any relationships to other tables specified with the SET RELATION command, the 'relationship chain' is traversed, satisfying the relationships.

VALID <expL3> | <expN5> | IN <expC14>

The VALID clause is used to validate data entry. When the GET is exited, the validation is called. If the <expL3> evaluates to true (.T.), the input is considered correct and the editing region is exited. If it evaluates to false (.F.), the data entered is rejected and a default error message, or the ERROR <expC7> is displayed. VALID <expL3> cannot be used in conjunction with Application Data Dictionary VALIDATION entries.

The VALID <expN5> option may be used with a function that returns a numeric value to select a GET for input. GET numbers are assigned in the order that they appear on the screen. The GETNO() function returns the number of the currently active get. The numeric value returned can have three different effects. When <expN5> is 0, the current GET remains in focus. When <expN5> is positive, the value is used to advance the GET focus the number of GETS specified. When <expN5> is negative, the value is used to move back the GET focus the number of GETS specified. VALID <expN5> cannot be used in conjunction with Application Data Dictionary VALIDATION entries.

The VALID IN option causes the input data to be checked against that specified in the character expression <expC14>. VALID IN is only active with character fields. It checks that the input data is contained as a sub string in <expC12>. Note that the <expC12> should consist of multiple strings of the same length as the field. VALID IN cannot be used in conjunction with the CHOICELIST clause. The VALID IN clause can be used in conjunction with Application Data Dictionary VALIDATION entries. The VALID IN check will be evaluated first.

VALIDATE WITH <program | procedure>

The VALIDATE WITH clause causes the specified <program | procedure> to be called, passing the input data as a parameter. The <program | procedure> must have a PARAMETERS statement with a single parameter defined. The parameter passed is a character string, regardless of the data type of the field in question. To signify that the input data has passed the validation checks the command SET VALIDATE ON should be issued. To reject the input data, the command SET VALIDATE OFF should be issued. The data itself may be modified using the SET FIELDVAL TO <expC> command. If the [HELP] key is pressed on a field that has a VALIDATE WITH specified, the <program | procedure> will be called with the parameter of "HELP". In this way, default popups and choice lists can be bypassed. VALIDATE WITH cannot be used in conjunction with the CHOICELIST clause. The VALIDATE WITH clause can be used in conjunction with Application Data Dictionary VALIDATION entries. The VALIDATE WITH check will be evaluated first.

Example

```
@3,3 get m_name;  
  picture "XXXXXXXXXX";  
  color gr+/b, r/w  
read
```

Products

Recital Mirage Server, Recital Terminal Developer

@...GET – Check Boxes

Class

Screen Forms

Purpose

Create a FoxPro style check box for data input

Syntax

```
@<expN1>,<expN2> GET <memvar> FUNCTION <expC1> | PICTURE <expC2>
[COLOR SCHEME <expN3> | COLOR <expC3>]
[DEFAULT <expr>]
[DISABLE | ENABLE]
[FONT <expC4>[, <expN4>]]
[MESSAGE <expC5>]
[PROPERTIES <expC6>]
[SIZE <expN5>,<expN6>]
[STYLE <expC7>]
[VALID <expL1> | <expN7>]
[WHEN <expL2>]
```

See Also

@...GET, @...SAY, SET COMPATIBLE, SET FILETYPE, READ

Description

The @...GET command can be used to create FoxPro style check box controls. Check boxes have two states, checked, which corresponds to a logical true (.T.) or a numeric 1, or unchecked, which corresponds to a logical false (.F.) or a numeric 0. Check boxes are displayed on the screen at the row <expN1>, column <expN2> coordinates as a pair of square brackets followed by a character string label. If the check box is checked, an asterisk is displayed inside the brackets. If the check box is unchecked, the brackets are empty. The check box can be checked/unchecked using the [SPACEBAR] or the [RETURN] key. To move off a check box, use the cursor keys. The status of the check box is returned to the <memvar>, which must be of logical or numeric data type.

The FUNCTION or PICTURE clauses are used to define the GET as a check box. The definition must include the check box code, “*C”, and the text label for the check box. The check box code must be preceded with an “@” in the PICTURE clause. The following options may also be included, immediately after the “*C”:

Picture/Function Option	Description
N	READ is not terminated when check box is checked.
T	READ is terminated when check box is checked.

The following clauses can optionally be used:

Keyword	Description
COLOR SCHEME <expN3> COLOR <expC3>	The color of @...GET output can be specified by including a set of color pairs in the COLOR clause or specifying a numeric COLOR SCHEME <expN3>. If you do not specify the COLOR COLOR SCHEME clause then the @...GET is displayed in the default Color of Fields. The color that is specified overrides the SET COLOR command, but only for the output of the current @...GET command.
DEFAULT <expr>	The DEFAULT clause specifies a default value for <memvar>.
DISABLE ENABLE	If the DISABLE clause is included, the GET is not active and cannot be selected or modified. GETs are enabled by default. READ will exit immediately if all GETs are marked DISABLE.
FONT <expC4>[, <expN4>]	The name of the font is specified in <expC4> and, optionally, the font size in <expN4>.
MESSAGE <expC5>	Defines a message to be displayed in the message line when the checkbox is the active GET.
PROPERTIES <expC6>	The PROPERTIES clause can be used to specify properties for the specified checkbox. For information on the available properties, please see The Mirage Object Model in the Recital Mirage documentation.
SIZE <expN5>,<expN6>	Defines the size of the control, <expN5> is the height and <expN6> is the width. Check boxes always have a height of one.
STYLE <expC7>	The STYLE clause can include the following in <expC7>: B = Bold I = Italic U = Underline - = Strikeout
VALID <expL1> <expN7>	The VALID clause is used to validate data entry. When the selected checkbox changes, the validation is called. If the <expL1> evaluates to true (.T.), the input is considered correct and the checkbox is exited. If it evaluates to false (.F.), the selection is rejected. The VALID <expN7> option may be used with a function that returns a numeric value to select a GET for input. GET numbers are assigned in the order that they appear on the screen. The GETNO() function returns the number of the currently active get. The numeric value returned can have three different effects. When <expN7> is 0, the current GET remains in focus. When <expN7> is positive, the value is used to advance the GET focus the number of GETS specified. When <expN7> is negative, the value is used to move back the GET focus the number of GETS specified.
WHEN <expL2>	The checkbox only allows selection when <expL2> evaluates to true (.T.).

NOTE: The @...GET...BUTTON syntax can also be used to create check box controls.

Example

```
mContinue = .F.
```

```
@20,0 get mContinue function “*CT Do you want to continue?”
```

```
read
```

```
// Another example
```

```
mContinue = .F.
```

```
@20,0 get mContinue picture “@*CT Do you want to continue?”;
```

```
    message “Press Return key or Spacebar to continue, cursor key to exit”;
```

```
    size 1,79
```

```
read
```

```
if mContinue
```

```
    // Continue
```

```
else
```

```
    // Exit
```

```
endif
```

Products

Recital Mirage Server, Recital Terminal Developer

@...GET - Lists

Class

Screen Forms

Purpose

Create a FoxPro style listbox

Syntax

```
@<expN1>,<expN2> GET <memvar>
FROM <array> [RANGE <expN3> [,<expN4>]] | POPUP <expC1>
FUNCTION <expC2> | PICTURE <expC3>
[COLOR SCHEME <expN5> | COLOR <expC4>]
[DEFAULT <exp>]
[DISABLE | ENABLE]
[FONT <expC5>[, <expN6>]]
[MESSAGE <expC6>]
[PROPERTIES <expC7>]
[SIZE <expN7>,<expN8>]
[STYLE <expC8>]
[VALID <expL1> | <expN9>]
[WHEN <expL2>]
```

See Also

@...GET, @...SAY, DECLARE, DEFINE POPUP, DIMENSION, PRIVATE, PUBLIC, SET COMPATIBLE, SET FILETYPE, READ

Description

The @...GET command can be used to create FoxPro style listboxes. Listboxes offer a scrolling list of choices for numeric or character values. The top left hand corner of the listbox is positioned at row <expN1>, column <expN2>. When an item is selected from the list, <memvar> is updated. If the initial value of <memvar> is numeric, the position of the selected item is stored, if it is character, the prompt is stored. The cursor keys and [PAGE UP], [PAGE DOWN] keys can be used to navigate within the list and the [SPACEBAR] or [RETURN] key or mouse click to select an item.

The list can be based either on the contents of a pre-declared one or two-dimensional array using the FROM <array> clause, or from a predefined popup using the POPUP <expC1> clause. Popups are defined using the DEFINE POPUP command.

The optional RANGE <expN3> [,<expN4>] clause can be used with array based listboxes to restrict the array elements that appear in the listbox. The <expN3> defines the number of the first array element to appear in the list and the optional <expN4> defines the number of elements to include. If <expN4> is not defined, all elements starting from <expN3> are included in the list.

The FUNCTION clause or the PICTURE clause is required. The ampersand character, '&' signals that the @...GET is a listbox:

```
@...GET mchoice FUNCTION "&" ...
or
@...GET mchoice PICTURE "@&"....
```

The following options may also be included, immediately after the “&”:

Picture/Function Option	Description
N	READ is not terminated when item is chosen.
T	READ is terminated when item is chosen.

The following clauses can optionally be used:

Keyword	Description
COLOR SCHEME <expN5> COLOR <expC4>	The color of @...GET output can be specified by including a set of color pairs in the COLOR clause or specifying a numeric COLOR SCHEME <expN5>. If you do not specify the COLOR COLOR SCHEME clause then the @...GET is displayed in the default Color of Fields. The color that is specified overrides the SET COLOR command, but only for the output of the current @...GET command.
DEFAULT <expr>	The DEFAULT clause specifies a default value for <memvar>.
DISABLE ENABLE	If the DISABLE clause is included, the GET is not active and cannot be selected or modified. GETs are enabled by default. READ will exit immediately if all GETs are marked DISABLE.
FONT <expC5>[, <expN6>]	The name of the font is specified in <expC5> and, optionally, the font size in <expN6>.
MESSAGE <expC6>	Defines a message to be displayed in the message line when the listbox is the active GET.
PROPERTIES <expC7>	The PROPERTIES clause can be used to specify properties for the listbox. For information on the available properties, please see The Mirage Object Model in the Recital Mirage documentation.
SIZE <expN7>,<expN8>	Defines the size of the listbox, <expN7> is the height and <expN8> is the width. By default, the width and height of the listbox are determined by the width of the widest item and the number of items in the list
STYLE <expC8>	The STYLE clause can include the following in <expC8>: B = Bold I = Italic U = Underline - = Strikeout
VALID <expL1> <expN9>	The VALID clause does not carry out validation in this case, since the <memvar> has already been updated, but can be used to call a UDF (User Defined Function). The VALID <expN9> option may be used with a function that returns a numeric value to select a GET for input. GET numbers are assigned in the order that they appear on the screen. The GETNO() function returns the number of the currently active get. The numeric value returned can have three different effects. When <expN9> is 0, the current GET remains in focus. When <expN9> is positive, the value is used to advance the GET focus the number of GETS specified. When <expN7> is negative, the value is used to move back the GET focus the number of GETS specified.
WHEN <expL2>	The listbox only allows selection when <expL2> evaluates to true (.T.).

Example

```
declare listarray[3]
listarray[1] = "Apple"
listarray[2] = "Banana"
listarray[3] = "Orange"
mchoice = "Banana"
@10,10 get mchoice from listarray function "&"
read
```

Products

Recital Mirage Server, Recital Terminal Developer

@...GET - Popups

Class

Screen Forms

Purpose

Create a FoxPro style popup

Syntax

```
@<expN1>,<expN2> GET <memvar>
[FUNCTION <expC1> | PICTURE <expC2>] | [FROM <array> [RANGE <expN3> [,<expN4>]]]
[COLOR SCHEME <expN5> | COLOR <expC3>]
[DEFAULT <exp>]
[DISABLE | ENABLE]
[FONT <expC4>[, <expN6>]]
[MESSAGE <expC5>]
[PROPERTIES <expC6>]
[SIZE <expN7>,<expN8>]
[STYLE <expC7>]
[VALID <expL1> | <expN9>]
[WHEN <expL2>]
```

See Also

@...GET, @...SAY, DECLARE, DEFINE POPUP, DIMENSION, PRIVATE, PUBLIC, SET COMPATIBLE, SET FILETYPE, READ

Description

The @...GET command can be used to create FoxPro style popups. Popups offer a popup list of options for numeric or character values. The top left hand corner of the popup is positioned at row <expN1>, column <expN2>. When an item is selected from the list, <memvar> is updated. If the initial value of <memvar> is numeric, the position of the selected item is stored, if it is character, the prompt is stored. The cursor keys and can be used to navigate within the list and the [SPACEBAR] or [RETURN] key or mouse click to select an option. In Recital Terminal Developer environments, the [SPACEBAR] or [RETURN] key is used to popup the list.

The FUNCTION clause or the PICTURE clause is required. The circumflex character, '^' signals that the @...GET is a popup. The following options may also be included, immediately after the '^':

Picture/Function Option	Description
N	READ is not terminated when option is chosen.
T	READ is terminated when option is chosen.

Unless the optional FROM <array> clause is being used, the FUNCTION or PICTURE clause must also contain the options for the popup. The options are preceded by a space, following the circumflex and any read termination option, and are separated with semi-colons.

```
@...GET mchoice FUNCTION "^ <option1>;<option2>;<option3> ..."...
or
@...GET mchoice PICTURE "@^ <option1>;<option2>;<option3> ..."...
```

FROM <array> [RANGE <expN3> [,<expN4>]]

As an alternative to defining the options in the FUNCTION or PICTURE clause, the popup can be based on the contents of a pre-declared one or two-dimensional array, specified in <array>. The optional RANGE clause can be used with array based popups to restrict the array elements that appear in the list. The

<expN3> defines the number of the first array element to appear in the list and the optional <expN4> defines the number of elements to include. If <expN4> is not defined, all elements starting from <expN3> are included in the list.

The following clauses can optionally be used:

Keyword	Description
COLOR SCHEME <expN5> COLOR <expC3>	The color of @...GET output can be specified by including a set of color pairs in the COLOR clause or specifying a numeric COLOR SCHEME <expN5>. If you do not specify the COLOR COLOR SCHEME clause then the @...GET is displayed in the default Color of Fields. The color that is specified overrides the SET COLOR command, but only for the output of the current @...GET command.
DEFAULT <expr>	The DEFAULT clause specifies a default value for <memvar>.
DISABLE ENABLE	If the DISABLE clause is included, the GET is not active and cannot be selected or modified. GETs are enabled by default. READ will exit immediately if all GETs are marked DISABLE.
FONT <expC4>[, <expN6>]	The name of the font is specified in <expC5> and, optionally, the font size in <expN6>.
MESSAGE <expC5>	Defines a message to be displayed in the message line when the popup is the active GET.
PROPERTIES <expC6>	The PROPERTIES clause can be used to specify properties for the popup. For information on the available properties, please see The Mirage Object Model in the Recital Mirage documentation.
SIZE <expN7>,<expN8>	By default, the width of the popup is determined by the width of the widest item and the height is determined by the number of options in the popup. The SIZE clause can optionally be used to override the width only. The height is specified in <expN7> (this is ignored but must be included) and the width in <expN8>.
STYLE <expC7>	The STYLE clause can include the following in <expC7>: B = Bold I = Italic U = Underline - = Strikeout
VALID <expL1> <expN9>	The VALID clause does not carry out validation in this case, since the <memvar> has already been updated, but can be used to call a UDF (User Defined Function). The VALID <expN9> option may be used with a function that returns a numeric value to select a GET for input. GET numbers are assigned in the order that they appear on the screen. The GETNO() function returns the number of the currently active get. The numeric value returned can have three different effects. When <expN9> is 0, the current GET remains in focus. When <expN9> is positive, the value is used to advance the GET focus the number of GETS specified. When <expN7> is negative, the value is used to move back the GET focus the number of GETS specified.
WHEN <expL2>	The popup only allows selection when <expL2> evaluates to true (.T.).

Example

```
clear  
store "Banana" to mchoice  
@10,10 get mchoice function "^T Apple;Orange;Banana"  
read  
dialog box "You chose" + mchoice
```

Products

Recital Mirage Server, Recital Terminal Developer

@...GET – Push Buttons

Class

Screen Forms

Purpose

Create a group of FoxPro style push buttons

Syntax

```
@<expN1>,<expN2> GET <memvar> FUNCTION <expC1> | PICTURE <expC2>
[COLOR SCHEME <expN3> | COLOR <expC3>]
[DEFAULT <exp>]
[DISABLE | ENABLE]
[FONT <expC4>[, <expN4>]]
[MESSAGE <expC5>]
[PROPERTIES <expC6>]
[SIZE <expN5>,<expN6>[, <expN7>]]
[STYLE <expC7>]
[VALID <expL1> | <expN8>]
[WHEN <expL2>]
```

See Also

@...GET, @...SAY, SET COMPATIBLE, SET FILETYPE, READ

Description

The @...GET command can be used to create FoxPro style push buttons. One of the group of push buttons may be selected and the number of the selected push button is returned to the <memvar>, which must be of numeric data type.

Push buttons are displayed on the screen starting at the row <expN1>, column <expN2> coordinates as a pair of angled brackets containing a character string label. A push button can be selected using the [SPACEBAR] or the [RETURN] key. To move off a push button, use the cursor keys.

The FUNCTION or PICTURE clauses are used to define the GET as a group of push buttons. The definition must include the push button code, “*”, followed by a space, followed by semi-colon separated text labels for the buttons. The push button code must be preceded with an “@” in the PICTURE clause. The following options may also be included, immediately after the “*”:

Picture/Function Option	Description
N	READ is not terminated when a button is selected.
T	READ is terminated when a button is selected. This is the default.
H	The buttons are displayed in a horizontal row.
V	The buttons are displayed in a vertical column. This is the default.

Accelerator keys may be defined for each button by preceding the accelerator letter in the button label with the characters “\<”. For example, ...function “* \<Save;Save \<As”, assigns the letter “S” as the accelerator key for “Save” and the letter “A” as the accelerator key for “Save As”.

The following clauses can optionally be used:

Keyword	Description
COLOR SCHEME <expN3> COLOR <expC3>	The color of @...GET output can be specified by including a set of color pairs in the COLOR clause or specifying a numeric COLOR SCHEME <expN3>. If you do not specify the COLOR COLOR SCHEME clause then the @...GET is displayed in the default Color of Fields. The color that is specified overrides the SET COLOR command, but only for the output of the current @...GET command.
DEFAULT <expr>	The DEFAULT clause specifies a default value for <memvar>.
DISABLE ENABLE	If the DISABLE clause is included, the GET is not active and cannot be selected or modified. GETs are enabled by default. READ will exit immediately if all GETs are marked DISABLE.
FONT <expC4>[, <expN4>]	The name of the font is specified in <expC4> and, optionally, the font size in <expN4>.
MESSAGE <expC5>	Defines a message to be displayed in the message line when the buttons are the active GET.
PROPERTIES <expC6>	The PROPERTIES clause can be used to specify properties for the buttons. For information on the available properties, please see The Mirage Object Model in the Recital Mirage documentation.
SIZE <expN5>,<expN6> [, <expN7>]	Defines the size of the buttons, <expN5> is the height and <expN6> is the width. The optional <expN7> defines the spaces between buttons.
STYLE <expC7>	The STYLE clause can include the following in <expC7>: B = Bold I = Italic U = Underline - = Strikeout
VALID <expL1> <expN8>	When a button is selected, the validation is called. If the <expL1> evaluates to true (.T.), the selection is considered valid and the buttons are exited. If it evaluates to false (.F.), the selection is rejected. The VALID <expN7> option may be used with a function that returns a numeric value to select a GET for input. GET numbers are assigned in the order that they appear on the screen. The GETNO() function returns the number of the currently active get. The numeric value returned can have three different effects. When <expN7> is 0, the current GET remains in focus. When <expN7> is positive, the value is used to advance the GET focus the number of GETS specified. When <expN7> is negative, the value is used to move back the GET focus the number of GETS specified.
WHEN <expL2>	The buttons can only be selected when <expL2> evaluates to true (.T.).

Example

```
mButton = 2
@10,10 get mButton function “*TH OK;Cancel”;
    message “Select Button to continue”
read

if mButton = 1
    // OK selected
else
    // Cancel or no selection made
endif
```

Products

Recital Mirage Server, Recital Terminal Developer

@...GET – Radio Buttons

Class

Screen Forms

Purpose

Create a group of FoxPro style radio buttons

Syntax

```
@<expN1>,<expN2> GET <memvar> FUNCTION <expC1> | PICTURE <expC2>
[COLOR SCHEME <expN3> | COLOR <expC3>]
[DEFAULT <exp>]
[DISABLE | ENABLE]
[FONT <expC4>[, <expN4>]]
[MESSAGE <expC5>]
[PROPERTIES <expC6>]
[SIZE <expN5>,<expN6>]
[STYLE <expC7>]
[VALID <expL1> | <expN7>]
[WHEN <expL2>]
```

See Also

@...GET, @...SAY, SET COMPATIBLE, SET FILETYPE, READ

Description

The @...GET command can be used to create FoxPro style radio buttons. One of the group of radio buttons may be selected and the number of the selected push button is returned to the <memvar>, which must be of numeric data type. Radio buttons are displayed on the screen starting at the row <expN1>, column <expN2> coordinates as a pair of brackets followed by a space then a character string label. If the radio button is selected, an asterisk is displayed inside the brackets. If the radio button is not selected, the brackets are empty. The radio buttons can be selected/deselected using the [SPACEBAR] or the [RETURN] key. To move off a radio button, use the cursor keys.

The FUNCTION or PICTURE clauses are used to define the GET as a group of radio buttons. The definition must include the radio button code, “*R”, followed by a space, followed by semi-colon separated text labels for the buttons. The radio button code must be preceded with an “@” in the PICTURE clause. The following options may also be included, immediately after the “*R”:

Picture/Function Option	Description
N	READ is not terminated when a button is selected. This is the default.
T	READ is terminated when a button is selected.

The following clauses can optionally be used:

Keyword	Description
COLOR SCHEME <expN3> COLOR <expC3>	The color of @...GET output can be specified by including a set of color pairs in the COLOR clause or specifying a numeric COLOR SCHEME <expN3>. If you do not specify the COLOR COLOR SCHEME clause then the @...GET is displayed in the default Color of Fields. The color that is specified overrides the SET COLOR command, but only for the output of the current @...GET command.
DEFAULT <expr>	The DEFAULT clause specifies a default value for <memvar>.
DISABLE ENABLE	If the DISABLE clause is included, the GET is not active and cannot be selected or modified. GETs are enabled by default. READ will exit immediately if all GETs are marked DISABLE.
FONT <expC4>[, <expN4>]	The name of the font is specified in <expC4> and, optionally, the font size in <expN4>.
MESSAGE <expC5>	Defines a message to be displayed in the message line when the any of the buttons is the active GET.
PROPERTIES <expC6>	The PROPERTIES clause can be used to specify properties for the specified button. For information on the available properties, please see The Mirage Object Model in the Recital Mirage documentation.
SIZE <expN5>,<expN6>	Defines the size of the control, <expN5> is the height and <expN6> is the width. Radio buttons always have a height of one, by default have the same width as their label, and are displayed on consecutive rows.
STYLE <expC7>	The STYLE clause defines the font style. The following style qualifiers are supported in <expC7>: B = Bold I = Italic U = Underline - = Strikeout
VALID <expL1> <expN7>	The VALID clause is used to validate data entry. When the selected button is changed, the validation is called. If the <expL1> evaluates to true (.T.), the input is considered correct and the button group is exited. If it evaluates to false (.F.), the data change is rejected. The VALID <expN7> option may be used with a function that returns a numeric value to select a GET for input. GET numbers are assigned in the order that they appear on the screen. The GETNO() function returns the number of the currently active get. The numeric value returned can have three different effects. When <expN7> is 0, the current GET remains in focus. When <expN7> is positive, the value is used to advance the GET focus the number of GETS specified. When <expN7> is negative, the value is used to move back the GET focus the number of GETS specified.
WHEN <expL2>	The buttons can only be selected when the <expL2> condition evaluates to true (.T.).

NOTE: The @...GET...BUTTON syntax can also be used to create radio button controls.

Example

```
mDest = 2
@10,10 get mDest function “*R Screen;Printer;File”
read
```

Products

Recital Mirage Server, Recital Terminal Developer

@...GET...SPINNER

Class

Screen Forms

Purpose

Create a FoxPro style numeric spinner

Syntax

```
@<expN1>,<expN2> GET <memvar> | <field> SPINNER <expN3> [, <expN4> [,<expN5>]]  
[COLOR SCHEME <expN6> | COLOR <expC1>]  
[DEFAULT <expN7>]  
[DISABLE | ENABLE]  
[FONT <expC2>[, <expN8>]]  
[FUNCTION <expC3>]  
[MESSAGE <expC4>]  
[PICTURE <expC5>]  
[PROPERTIES <expC6>]  
[RANGE [<expN9>][,<expN10>]]  
[SIZE <expN11>,<expN12>]  
[STYLE <expC7>]  
[VALID <expL1> | <expN13> [ERROR <expC8>]]  
[WHEN <expL2>]
```

See Also

@...GET, @...SAY, DECLARE, DEFINE POPUP, DIMENSION, PRIVATE, PUBLIC, SET COMPATIBLE, SET FILETYPE, READ

Description

The @...GET...SPINNER command can be used to create FoxPro style numeric spinners. Spinners allow you to 'spin' through a series of numeric values. The top left hand corner of the spinner is positioned at row <expN1>, column <expN2>.

Spinners are only available in Recital Mirage. In Recital Terminal Developer the @...GET...SPINNER command is treated as a standard numeric @...GET. In Recital Mirage, the mouse is used to click on the up or down arrows to change the spinner value.

The numeric expression, <expN3>, defines the increment/decrement value to be used when the up/down arrow is clicked. The range of allowed values can be specified in the optional <expN4>, used to specify the minimum value, and <expN5>, used to specify the maximum value.

The following clauses can optionally be used:

Keyword	Description
COLOR SCHEME <expN6> COLOR <expC1>	The color of @...GET output can be specified by including a set of color pairs in the COLOR clause or specifying a numeric COLOR SCHEME <expN6>. If you do not specify the COLOR COLOR SCHEME clause then the @...GET is displayed in the default Color of Fields. The color that is specified overrides the SET COLOR command, but only for the output of the current @...GET command.
DEFAULT <expN7>	The DEFAULT clause specifies a default value for <memvar>.
DISABLE ENABLE	If the DISABLE clause is included, the GET is not active and cannot be selected or modified. GETs are enabled by default. READ will exit immediately if all GETs are marked DISABLE.
FONT <expC2>[, <expN8>]	The name of the font is specified in <expC2> and, optionally, the font size in <expN8>.
FUNCTION <expC3>	The FUNCTION clause can include the following in <expC3>: B = The value is left justified. I = The value is centered. J = The value is right justified. K = The entire value is selected when the spinner is selected. L = The value is displayed with leading zeroes. Z = The value is displayed as blank if zero. ^ = The value is displayed using scientific notation. \$ = The value is displayed in currency format.
MESSAGE <expC4>	Defines a message to be displayed in the message line when the spinner is the active GET.
PICTURE <expC5>	The PICTURE clause can include the following in <expC5>: 9 = Allows digits and signs (+ -) to be entered. # = Allows digits, blanks and signs (+ -) to be entered. \$ = Displays the currency symbol. * = Displays asterisks to the left of the value. . = Specifies the position of the decimal point. , = Specifies the position of 'thousand' separators.
PROPERTIES <expC6>	The PROPERTIES clause can be used to specify properties for the spinner. For information on the available properties, please see The Mirage Object Model in the Recital Mirage documentation.
RANGE [<expN9>[,expN10>]	The RANGE clause can be used to check whether the spinner value falls between acceptable minimum, <expN9> and maximum, <expN10> limits. Either limit can be omitted.
SIZE <expN11>,<expN12>	Defines the size of the control, <expN11> is the height and <expN12> is the width.
STYLE <expC7>	The STYLE clause can include the following in <expC7>: B = Bold I = Italic U = Underline - = Strikeout

VALID <expL1> <expN13> [ERROR <expC8>]	The VALID clause is used to validate data entry. When the selected button is changed, the validation is called. If the <expL1> evaluates to true (.T.), the input is considered correct and the button group is exited. If it evaluates to false (.F.), the data change is rejected. The optional ERROR clause causes the character expression <expC6> to be displayed when <expL1> evaluates to .F. (false). The VALID <expN13> option may be used with a function that returns a numeric value to select a GET for input. GET numbers are assigned in the order that they appear on the screen. The GETNO() function returns the number of the currently active get. The numeric value returned can have three different effects. When <expN13> is 0, the current GET remains in focus. When <expN13> is positive, the value is used to advance the GET focus the number of GETS specified. When <expN13> is negative, the value is used to move back the GET focus the number of GETS specified.
WHEN <expL2>	The spinner can only be used when the <expL2> condition evaluates to true (.T.).

Example

store 1 to mspinner

@20,0 say "Spinner: " get mspinner spinner 1, -5 , 20

Products

Recital Mirage

@...MENU

Class

Menus

Purpose

Define a menu option

Syntax

```
@<expN1>,<expN2> MENU <expC1>
[COMMAND <commands>]
[HELP <expC2>]
[HELPPFILE <.hlp filename> | (<expC3>)]
[NOREFRESH]
[PULLDOWN <expC4> | @<program | procedure> WITH [<parameter-list>]
```

See Also

MENU, MENU FIELDS, MENU QUERY, MENU SCOPE, MENU FRAME, MENU AT, MENU FORMAT, MENU(), MENUITEM() SAVE MENU, SET MCONFIRM, SET PREMENU, SET POSTMENU, RESTORE MENU

Description

The @...MENU command is used to create menus. This command defines the menu options that are activated with the MENU command. @...MENU menus can also be created in the Forms Designer, see the CREATE SCREEN command for full details.

Menus can be navigated using the cursor keys. Pulldown menus will be activated automatically as they are highlighted, unless SET MCONFIRM is ON. Menu items can be selected using the first character of the item, the item accelerator key or the [RETURN] key. If SET MCONFIRM is ON, then a menu must be highlighted using the cursor keys or accelerator, then selected using the [RETURN] key.

The MENU <expC1> clause defines the menu option that will be displayed at the specified row <expN1> and column <expN2> when the MENU command is executed. You may also use the character expression <expC1> to define horizontal lines, non-selectable items, and accelerator keys in menus. Horizontal lines are defined by specifying a backslash and a dash (\-). Non-selectable items are defined by preceding the menu option with a backslash. Accelerator keys are highlighted letters in menu option text that select the option when pressed. To designate a letter as an accelerator key, precede the letter with a backslash and a "less than" sign (<). Recital menu options can always be selected by pressing the first letter of the option. Accelerator keys are an excellent way to illustrate that selection method, and to provide alternate keys when there are menu options beginning with the same first letter.

NOTE: When a menu is placed in user-defined window, the coordinates are relative to the window not to the screen.

COMMAND <commands>

The COMMAND <commands> clause specifies the commands to be executed if the particular menu option is selected when the menu is activated. Multiple commands can be used by separating each command with the ";" character. If no COMMAND option is specified, you will exit from the menu when a menu option is selected. If you want to exit from the menu and use an @...MENU COMMAND option, the EXIT keyword must be specified on the MENU command. Selection of any item from the menu will then exit the menu after execution of the specified COMMAND line, unless the NOEXIT keyword is included in the COMMAND line.

It should be noted that the COMMAND option in the MENU command can call another menu, which can in turn call another menu. By nesting MENU commands the Recital/4GL code can be written in a non-procedural way, since once the menu is exited, control will return automatically to the calling menu.

HELP <expC2>

A HELP message, associated with each menu option, can be displayed in the message line when the menu option is highlighted. The message <expC2>, is a text line (maximum 80 characters) which can include macro substitution of memory variables and function key substitution of legal control characters.

HELPPFILE <.hlp filename> | (<expC3>)

The HELPPFILE clause allows context sensitive help information to be displayed when the [HELP] key is pressed. The <.hlp filename> will be displayed in a read-only window for viewing when the [HELP] key is pressed. The file name can be substituted with a <expC3>, enclosed in round brackets, which returns a valid filename. The command, SET INSTRUCT, must be ON when using this option.

NOREFRESH

By default, when returning from executing the command(s) specified for the menu option, the current menu is refreshed on the screen. The NOREFRESH keyword disables this.

PULLDOWN <expC4> | @<program | procedure> WITH [<parameter-list>]

Pulldown and pullright menus can be invoked in Recital/4GL programs by using the PULLDOWN option of the @...MENU command and the PULLRIGHT option of the MENU command respectively. The PULLDOWN <expC4> refers to a list of items that will appear in the pulldown menu upon placing the cursor on the associated menu item. The items are defined in a comma-separated character string.

If a text item is selected (i.e. by pressing [RETURN] while the item is highlighted) the MENUITEM() and MENU() functions can be used in the command line to allow processing to continue based on the item that was selected.

The PULLDOWN <program | procedure> WITH <parameter-list> option will cause execution of a command procedure or function and pass parameters to it when the cursor is placed on the menu item. In this option, <program | procedure> is any procedure or function name and <parameter-list> is a list of parameters separated by commas that will be passed to the <program | procedure>. The <program | parameter> can include expressions, so that if you pass MENUITEM() or MENU() as a parameter to a procedure, processing can proceed selectively based on the menu item chosen.

The @...MENU command also supports FoxBASE+ style menus, in which case the full syntax is as shown below:

```
@ <row>, <col> MENU <array name>,<nitems> [,<scroll size>];  
[TITLE <expC>]
```

This FoxBASE+ command cannot be used in conjunction with any of Recital's @...MENU commands. The READ MENU TO command is used to activate the FoxBASE+ @...MENU command.

Example

```
@0,0 menu "Exit";  
    help "Exit from the system."  
@0,5 menu "Browse";  
    help "Browse the table.";  
    command "Browse; clear"  
@0,12 menu "Append";  
    help "Append a new record.";  
    command "append"  
menu
```

Products

Recital Mirage Server, Recital Terminal Developer

@...PROMPT

Class

Menus

Purpose

Display a menu option

Syntax

```
@ <expN1>,<expN2> PROMPT <expC1>
[MESSAGE <expC2>]
[SIZE <expN3>,<expN4>]
[IMAGE <expC3>]
```

See Also

@...BOX, @...CLEAR, @...GET, @...FILL, @...MENU, @...SAY, @...SCROLL, @...TO, CREATE SCREEN, MENU TO, MODIFY SCREEN

Description

The @...PROMPT command is used in conjunction with the MENU TO command. @..PROMPT displays a menu with the prompt <expC1> at the row and column position specified by <expN1> and <expN2>. The menu is activated using the MENU TO <variable> command and the number of the selected menu is stored to <variable>. Menu numbers start from 1.

MESSAGE <expC2>

If the optional MESSAGE clause is specified, <expC2> is displayed in the message line when that menu is highlighted.

SIZE <expN3>,<expN4>

The SIZE clause can be used under Recital Mirage to define the size of the prompt <expC1> or image <expC3>. The <expN3> refers to the number of rows occupied by an image, <expN4> to the number of columns occupied by an image or prompt.

IMAGE <expC3>

The IMAGE clause can be used under Recital Mirage to define an image to be displayed in place of the textual prompt. The GIF or JPEG image file to be displayed is specified using <expC3>.

Example

```
@10,10 prompt "edit";
    message "Edit a record"
@11,10 prompt "append";
    message "Add a record"
@12,10 prompt "delete";
    message "Delete a record"
menu to choice
```

```
do case
    case choice = 1
        do editproc
    case choice = 2
        do addproc
    case choice = 3
        do deleproc
    otherwise
```



```
? "No choice made"  
endcase
```

```
//Recital Mirage
```

```
@5,30 prompt "Edit" message "Edit menu" size 2,2 image "lockbtn.gif"
```

```
@7,30 prompt "Append" message "Append menu" size 2,2 image "newbtn.gif"
```

```
@9,30 prompt "Delete" message "Delete menu" size 2,2 image "deletebtn.gif"
```

```
menu to m_var
```

Products

Recital Mirage Server, Recital Terminal Developer

@...SAY

Class

Input/Output

Purpose

Display the result of an expression at a specified screen position

Syntax

@<expN1>,<expN2> SAY <expr>
[COLOR <standard | enhanced> | COLOR SCHEME <color-scheme>]
[FUNCTION <expC1> | PICTURE<expC2>]
[NAME <expC3>]
[PROPERTIES <expC4>]
[GET <field>|<memvar>]

See Also

?, ??, ???, @...BOX, @...CLEAR, @...GET, @...FILL, @...MENU, @...PROMPT, @...SCROLL, @...TO, @...TOOLBUTTON, SET DEVICE, SET PRINT

Description

The @...SAY command displays the result of an expression, <expr> at the specified row, <expN1> and column, <expN2> on the screen. @...SAY may also be used with an optional GET clause. This causes the specified field or memory variable to be displayed on the screen at the current cursor position. The <expr> can be any valid Recital/4GL expression, including user-defined functions. For screens and windows, the first addressable row, column coordinate pair is 0,0. Rows are numbered from top to bottom, columns from left to right. If SET DEVICE TO PRINT is in effect, then the @...SAY command directs output to the printer. The specified row and column denote the position of the print head at which the resulting expression will be placed. For printers the first addressable row, column coordinate pair is 1,0.

COLOR <standard> | COLOR SCHEME <color-scheme>

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. If you do not specify the COLOR clause then the @...SAY is displayed in the standard color. The color that is specified overrides the SET COLOR command, but only for the output of the current @ command. Only the first color pair in the color scheme or color pair list affects the color of the @...SAY output. A color scheme is a set of 10 predefined color pairs. The color pairs in a color scheme can be changed with the SET COLOR OF SCHEME. 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.

FUNCTION <expC1>

The optional FUNCTION clause can be used to control how the <expr> is displayed or printed. FUNCTION codes can also be specified in a PICTURE clause if they are prefixed with an @ character. Since a FUNCTION code effects the entire expression only the following FUNCTION codes can be used with a FUNCTION.

Function Codes	Description
B	Left-justifies numeric data within the display.
C	CR is displayed after a positive number to indicate credit.
D	Uses the current SET DATE format.
E	Displays as a BRITISH date.
F	Displays negative data blinking.
Q	DB is displayed after positive numbers.

Function Codes	Description
T	Trims leading and trailing blanks.
X	DB is displayed after negative numbers to indicate a debit.
Y	CR is displayed after negative numbers.
Z	<expr> is displayed as all blanks if its numeric value is 0.
(Encloses negative numbers in parentheses.
!	Converts characters to upper case.

PICTURE <expC2>

The optional PICTURE clause allows for specialized formatting to be performed on the expression result before it is output. Picture codes apply to individual characters in the output. The following picture codes can be used.

Picture Code	Description
X	Allows any character at the specified position.
Y	Displays 'Y' for .T. and 'N' for .F. if there is a logical value at the specified position.
!	Displays the character at the specified position in upper case.
\$	Displays a dollar sign at the specified position or prefixes numeric values with the currency symbol specified by the SET CURRENCY command.
*	Displays an asterisk at the specified position or prefixes numeric values with an asterisk.
.	Specifies the decimal point position for numeric values or displays a '.' at the specified position.
,	Specifies the separation character for numeric values over one thousand or displays a comma at the specified position.

NAME <expC3> (Recital Mirage only)

The NAME clause is used to create a 'wrapper' object for the SAY. The <expC3> must be a unique name for the current READ. Once defined, the NAME clause allows properties to be set and methods to be called on the client for that SAY. For additional information on object properties and methods, please see the Recital Mirage documentation.

PROPERTIES <expC4> (Recital Mirage only)

The PROPERTIES clause can be used to specify properties for the specified text. For information on the available properties, please see the Recital Mirage documentation.

GET

The @...SAY and @...GET commands can be combined using the @...SAY...GET syntax. The @...GET edit region will be positioned after the @...SAY display, with a single space in between. Please see the @...GET command itself for the full @...GET syntax.

Example

```
@07,03 say "Recital" color r/w
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

@...SAY...BITMAP

Class

Screen Forms

Purpose

Display a bitmap image

Syntax

@<expN1>,<expN2> SAY <expC> BITMAP
[CENTER]
[ISOMETRIC | STRETCH]
[SIZE <expN3>,<expN4>]

See Also

@...GET, @...SAY, SET COMPATIBLE, SET FILETYPE

Description

The FoxPro compatible @...SAY...BITMAP command is used to display images in Recital Mirage applications. The top left hand corner of the image is displayed at row <expN1>, column <expN2>. The name of the file to be displayed is specified in the character expression <expC>. The supported file formats are GIF and JPG.

CENTER

If the CENTER keyword is specified, the image is centered in the limits specified in the SIZE clause.

ISOMETRIC | STRETCH

The ISOMETRIC and STRETCH keywords can be used to determine the display behavior when the image dimensions and the SIZE dimensions do not match. If ISOMETRIC is specified, the image is scaled using its original proportions. If STRETCH is specified, the image is stretched to fit and the original proportions are not maintained. If the SIZE dimensions are smaller than the image size and neither ISOMETRIC nor STRETCH is specified, the image is clipped to fit.

SIZE <expN3>,<expN4>

The size clause can be included to specify the display size of the image. If the SIZE clause is omitted, the image is displayed at its default size. The height of the image is specified in <expN3> and the width in <expN4>. Both the height and the width are specified as multiples of the current font size.

Example

```
@5,0 say "Image:      "  
@6,0 say "recital.jpg" bitmap;  
    size 6,12
```

Products

Recital Mirage

@...SCROLL

Class

Screen Forms

Purpose

Scroll designated screen area

Syntax

```
@ <expN>,<expN2> TO <expN3>,<expN4> SCROLL  
[UP | DOWN | LEFT | RIGHT [BY <expN>]]  
[WRAP]
```

See Also

@...BOX, @...CLEAR, @...GET, @...FILL, @...MENU, @...PROMPT, @...SAY, @...TO,
@...TOOLBUTTON, HSCROLL(), SCROLL, SCROLL(), MAXCOL(), MAXROW()

Description

The @...SCROLL command shifts the contents of a specified region of the screen up, or down, or to the left or right. If a window is active, the coordinates are used relative to the origin of that window.

<expN>,<expN2> TO <expN3>,<expN4>

The top left corner is specified at <expN>,<expN2> and the bottom right corner at <expN3>,<expN4>.

UP | DOWN | LEFT | RIGHT [BY <expN>]

You may optionally specify a direction, UP, DOWN, LEFT or RIGHT in which to move contents in the specified area. If no direction is specified, the @...SCROLL command moves characters UP. You may also specify the number of rows or columns by which to scroll the characters with the optional BY <expN> clause. If you have specified LEFT or RIGHT as a direction, the BY <expN> clause refers to the number of columns. If you have specified UP or DOWN, the BY <expN> clause refers to the number of rows. If the BY <expN> clause is not specified, the @...SCROLL command moves characters one row or column. Negative numbers reverse the direction of scrolling.

WRAP

The WRAP keyword causes scrolled characters to move onto the opposite row or column if they are scrolled beyond the designated area of the screen.

Example

Define window scroll from 03,05 to 15,75

activate window scroll

```
@ 0,0 say replicate ("0",50)
```

```
@ 1,0 say replicate ("1",50)
```

```
@ 2,0 say replicate ("@",50)
```

```
@ 3,0 say replicate ("#",50)
```

```
@ 4,0 say replicate ("$",50)
```

```
for i = 1 to 5
```

```
    @ 0,0 to 4,60 scroll
```

```
    sleep 1
```

```
next
```

```
release windows
```

Products

Recital Terminal Developer

@...TO

Class

Screen Forms

Purpose

Draw a box or line on the screen

Syntax

@ <expN1>,<expN2> TO <expN3>,<expN4>
[BOLD]
[COLOR <color code> | COLOR SCHEME <color scheme>]
[DOUBLE | SINGLE | PANEL | BOX <expC>]
[FILL <expC>]
[LABEL <expC>]
[REVERSE]
[SHADOW]

See Also

@...BOX, @...CLEAR, @...GET, @...FILL, @...MENU, @...PROMPT, @...SAY, @...SCROLL, MENU FRAME, MENU AT, ROW(), COL()

Description

This command is used to draw a box or a line. If you omit the optional clauses, the box is drawn with single lines.

<expN1>,<expN2> TO <expN3>,<expN4>

The top left corner is specified at row <expN1>, column <expN2> and a bottom right corner at row <expN3>, column <expN4>. If the two row coordinates are the same, a horizontal line is drawn. If the two column coordinates are the same, a vertical line is drawn.

BOLD

If the BOLD keyword is specified, then the contents of the box are displayed in highlighted reverse video.

COLOR <color> | COLOR SCHEME <color-scheme>

The optional COLOR <color> clause will set the foreground, background or both colors when the color code is specified. These are the same codes used by SET COLOR. If you do not provide color codes, this command uses the <standard> colors as defined in the SET COLOR command. The COLOR SCHEME <color scheme> takes a color scheme from a set of 10 predefined color pairs. The color pairs in a color scheme can be changed with the SET COLOR OF SCHEME command. The color scheme number should be specified in <color scheme>.

DOUBLE | SINGLE | PANEL | BOX <expC>

If SINGLE is specified, then the box is drawn with a single line border. If DOUBLE is specified, then the box is drawn with a double-line border. If PANEL is included, the box is drawn with a solid border. The BOX <expC> specifies characters to use to draw the box. The <expC> is a string of up to nine characters. The characters define the top-left corner, top row, top-right corner, right side, bottom right corner, bottom row, bottom left corner, left side and the inside of the box, in that order.

LABEL <expC>

The LABEL clause draws a box with the specified title, <expC>.

REVERSE

If the REVERSE keyword is specified then the contents of the box are displayed in reverse video.

SHADOW

The SHADOW keyword shades the right hand side and bottom edge of the box, producing a 3D effect.

Example

@05,00 to 08,79 box "123456789"

Products

Recital Mirage Server, Recital Terminal Developer

ACCEPT

Class

Screen Forms

Purpose

Prompt for input to a memory variable

Syntax

ACCEPT [<expC>] TO <memvar>

See Also

INPUT, WAIT, @...GET, READ, MENU

Description

The ACCEPT command prompts for input to the memory variable <memvar>. If the optional <expC> is specified, this will be displayed as the prompt, otherwise a colon ':' will be displayed. The user entry is stored in the <memvar> as a character string. If the [RETURN] key is entered, the <memvar> will contain a null string. If the <memvar> does not exist prior to the ACCEPT then it will be created.

Example

accept "Enter customer name? " to name

Enter customer name? BILL

? name

BILL

? type("name")

C

Products

Recital Mirage Server, Recital Terminal Developer

ACTIVATE MENU

Class

Menus

Purpose

Activate an Xbase style bar menu

Syntax

```
ACTIVATE MENU <expC1>  
[NOCLEAR]  
[NOWAIT]  
[PAD <expC2>]
```

See Also

DEFINE MENU, DEFINE PAD, ON PAD, PAD(), DEACTIVATE MENU, SHOW MENU, MENU(), SET COMPATIBLE ON

Description

The ACTIVATE MENU command activates and displays an Xbase style menu. The menu name <expC1> must have been previously defined. The command SET COMPATIBLE must be set ON when using Xbase style menus.

NOCLEAR

The NOCLEAR keyword prevents the popup display being cleared after the menu is exited.

NOWAIT

The NOWAIT clause will not wait for user input after the ACTIVATE POPUP command is issued. The popup is displayed and activated, but program execution continues immediately.

PAD <expC2>

The optional PAD keyword will position the highlight bar on the pad name <expC2> when the menu is displayed. The pad name must have been previously defined. If no pad name is used, then the highlight is positioned on the first defined pad.

Example

```
activate menu sort_men
```

Products

Recital Mirage Server, Recital Terminal Developer

ACTIVATE POPUP

Class

Menus

Purpose

Activate an Xbase style pop-up menu

Syntax

ACTIVATE POPUP <expC>

[AT <expN1>,<expN2>]

[BAR <expN3>]

[NOCLEAR]

[NOWAIT]

See Also

BAR(), DEACTIVATE POPUP, DEFINE POPUP, POPUP(), PROMPT(), SHOW POPUP, SET COMPATIBLE ON

Description

The ACTIVATE POPUP command activates an Xbase style pop-up menu name <expC> and displays it when the user moves the highlight to the corresponding pad. Each pop-up is DEACTIVATED as the highlight is moved to the next pad, or when the DEACTIVATE POPUP command is used.

AT <expN1>,<expN2>

You can specify where the popup is displayed by including AT <expN1>,<expN2>. The upper left corner of the popup is positioned at the coordinates specified by <expN1>,<expN2>. The position specified by this clause take precedence over a position specified by the FROM clause in DEFINE POPUP.

BAR <expN>

The BAR <expN3> allows you to specify the selected popup option number when the popup is displayed.

NOCLEAR

The NOCLEAR keyword prevents the popup display being cleared after the menu is exited.

NOWAIT

The NOWAIT clause will not wait for user input after the ACTIVATE POPUP command is issued. The popup is displayed and activated, but program execution continues immediately.

Example

on pad file_nm of sort_men activate popup popup_1

on pad srt_type of sort_men activate popup popup_2

on pad key_nm of sort_men activate popup popup_3

on pad prfrm of sort_men activate popup popup_4

Products

Recital Mirage Server, Recital Terminal Developer

ACTIVATE SCREEN

Class

Screen Forms

Purpose

Activate full screen display

Syntax

ACTIVATE SCREEN

See Also

ACTIVATE WINDOW, CLEAR WINDOWS, DEACTIVATE WINDOW, DEFINE WINDOW, HIDE WINDOW, MOVE WINDOW, MODIFY MEMO, RELEASE WINDOWS, RESIZE WINDOW, RESTORE WINDOW, SAVE WINDOW, SHOW WINDOW, SET COMMANDWINDOW, SET ERRORWINDOW, SET KEY...TO, SET PROCEDURE TO, SET TRACEWINDOW, SET WINDOW OF EDIT, SET WINDOW OF MEMO, WROWS(), WCOLS(), WEXIST(), WVISIBLE(), WONTOP(), WOUTPUT()

Description

The ACTIVATE SCREEN command disables the active window and redirects output to the entire screen. Displayed windows remain on the screen, and subsequent output is displayed behind the windows.

A window is an area of the screen designated for output and input. Windows are defined with the DEFINE WINDOW command, and are displayed to the screen with the ACTIVATE WINDOW or SHOW WINDOW commands. There is no limit to the number of defined windows.

The HIDE WINDOW command may be used in a hot-key procedure to switch the screen display from windows to full screen. Hot-keys allow a procedure to be called when the user presses a particular key.

Example

```
define window temp_output;  
  from 16,45 to 22,79;  
  title "Output Window";  
  float;  
  grow
```

```
activate window temp_output  
activate screen
```

Products

Recital Mirage Server, Recital Terminal Developer

ACTIVATE WINDOW

Class

Screen Windows

Purpose

Activate a defined window, or list of windows.

Syntax

ACTIVATE WINDOW <window-name> |<window-name list> | ALL
[NOSHOW]

See Also

ACTIVATE SCREEN, CLEAR WINDOWS, DEACTIVATE WINDOW, DEFINE WINDOW, HIDE WINDOW, MOVE WINDOW, RELEASE WINDOWS, RESIZE WINDOW, RESTORE WINDOW, SAVE WINDOW, SHOW WINDOW, WROWS(), WCOLS(), WEXIST(), WVISIBLE(), WONTOP(), WOUTPUT()

Description

The ACTIVATE WINDOW command displays and activates windows that have been defined with the DEFINE WINDOW command. When a window is activated, all subsequent output is displayed in that window. Only one window may be activated at a time. Activating additional windows does not clear the display of previously activated windows. The DEACTIVATE WINDOW command clears the display of activated windows, but leaves the window definition in memory. The RELEASE WINDOW command clears both the display and the definition of windows from memory. The SAVE and RESTORE WINDOW commands may be used to keep window definitions in a file that can be reused at any time.

The ACTIVATE WINDOW command can be used to activate a single window, a list of windows, or all currently defined windows. To activate a single window, specify the name of the window that you wish to activate. The <window-name> is the name in the window definition created with the DEFINE WINDOW command. The <window-name list> is a list of window names, each separated by a comma. When activating a list of windows, the ACTIVATE WINDOW command displays the windows in the order that they are listed, and the last window in the list is activated. To display all currently defined windows, use the keyword ALL. The windows display in the order that they are defined, and the last window defined is activated.

NOSHOW

If the NOSHOW keyword is included, the specified window will be activated, but will not be displayed on the screen until the SHOW WINDOW <window-name> command is issued.

Example

```
define window win1;  
    from 2,2 to 12,43  
activate window win1 noshow  
dir  
wait  
show window win1
```

Products

Recital Terminal Developer

ALIAS

Class

Applications

Purpose

Define a User Defined Command (UDC)

Syntax

ALIAS [<expC> [<command>]]

See Also

RUN, !, SPAWN, FUNCTION, KEYWORD

Description

The ALIAS command allows for the definition of User Defined Commands (UDCs). The ALIAS command, on its own, causes all active UDCs to be listed.

<expC>

The <expC> is the name to call the user-defined command. The ALIAS <expC> without a subsequent <command> statement removes that UDC.

<command>

The <command> character expression can contain multiple Recital/4GL commands, each separated with a ';'. Up to nine parameters may be substituted by preceding the parameter number with a '%' character, e.g. %1 %2 etc. Parameter %0 matches the complete line following the command name.

Example

alias lm "list memory" && defines lm

alias lo "list off"

alias l "list off for ord_id = [%1]"

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

APPEND

Class

Screen Forms

Purpose

Full screen append of records into the active table

Syntax

APPEND
[NOCLEAR]

See Also

@...GET, APPEND BLANK, APPEND FROM, CREATE, CREATE SCREEN, CHANGE, INSERT, SET CARRY, FMT()

Description

The APPEND command is a full screen command used to append records to the end of the active table. A default form with blank fields will be activated on the screen.

The APPEND can be terminated using the [EXIT/SAVE] key to save the current record, or the [ABANDON] key to abandon the current record. If SET VERIFY is ON, the save or abandon must be confirmed. If SET MENU is ON and the default APPEND form is being used, a menu with details of the active keys is displayed at the top of the screen. Pressing the [TAB] key allows the menu to be toggled on or off.

You can design your own forms for appending records, with the Forms Designer (CREATE SCREEN). Once the form has been activated with SET FORMAT TO, it will be used instead of the default form when the APPEND command is issued.

Normally, all of the fields are initialized to blank. This behavior can be overridden with the SET CARRY ON command or the [CARRY MODE] key from within the append form. If SET CARRY is ON, data from the previous record will be carried over as the default for the next APPEND operation. Default information may also be automatically inserted into the form via the Applications Data Dictionary.

When memo fields are displayed, the memo field label is in lower case if the field is empty and upper case if the memo field contains a value. Memos can be edited in a popup notepad window by pressing the [INSERT] or [HELP] keys. Once in the notepad window, pressing the [HELP] key displays a menu of memo editing keys. These keys include facilities for reading from and writing to external files and printing on the system printer.

Forms may be overlaid, one on top of the other, if APPEND is used in conjunction with the FMT() function and the SAVE SCREEN and RESTORE SCREEN commands. If the topmost form is 'boxed' using the @...TO...LABEL command, the area of the screen it occupies will be cleared.

All open indexes are updated. If any of the indexes is unique, then duplicate keys will not be appended. An appropriate error message will be displayed. If the active table is shared, other users may edit existing records during an APPEND operation, as no records are locked.

Pressing the [EXIT/SAVE] or [NEXT SCREEN] keys causes the record to be written to the table and made to other users. The [MENUBAR] key does not write the record to the table.

NOCLEAR

The NOCLEAR keyword disables the erasing of the screen on entry and exit from APPEND.

The following keys are active within an APPEND form.

Key	Action
ABANDON	Discard current record then exit from the form
CARRY MODE	Toggle CARRY on and off
CURSOR DOWN	Skip to next field
CURSOR LEFT	Skip to previous field
CURSOR RIGHT	Skip to next field
CURSOR UP	Skip to previous field
DELETE FIELD	Initialize field
EDIT FIELD	Enter field edit mode
EXIT/SAVE	Write current record then exit from the form
HELP	Activate pop-up help
MENUBAR	Activate the APPEND menu bar
NEXT RECORD	Write record then skip to next record
REFRESH	Redraw the form
TAB	Toggle function key menu on and off

If SET MOUSE is ON, cursor keys will move the cursor anywhere on the screen rather than just from field to field. If SET NAVIGATE is ON, the cursor moves to fields following the direction specified by the key being pressed, rather than following the order of the GETS on the form. When the RETURN key is pressed, the cursor moves to the nearest field when SET NAVIGATE is ON.

The following keys are active in field edit mode:

Key	Action
BACKSPACE	Delete character before cursor
CURSOR LEFT	Skip to previous character
CURSOR RIGHT	Skip to next character
DELETE CHAR	Delete character under cursor
DELETE FIELD	Delete from cursor to end of field
DELETE WORD	Delete current word
INSERT MODE	Toggle insert / overwrite mode
WORD LEFT	Skip left a word
WORD RIGHT	Skip right a word

The following menu options are available from the APPEND menu bar in the default form:

Menu Item	Action
Descriptions	Toggle the field descriptions on and off
Help	Activate on-line help system

Example

use patrons index names
set format to patrons
append

Products

Recital Mirage Server, Recital Terminal Developer

APPEND AUTOMEM

Class

Memory Variables

Purpose

Appends a blank record into the active table, then updates the fields with memory variable values

Syntax

APPEND AUTOMEM

See Also

APPEND BLANK, APPEND FROM, CLEAR AUTOMEM, REPLACE AUTOMEM, STORE AUTOMEM, USE...AUTOMEM

Description

The APPEND AUTOMEM command appends a blank record into the active table and then updates the fields with the values from memory variables of the same name. Such memory variables can be generated automatically using the STORE AUTOMEM or USE...AUTOMEM commands.

Example

```
use cust
store automem
m.account_no = strzero(seqno(),5)
append automem
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

APPEND BLANK

Class

Fields and Records

Purpose

Append a record into the active table

Syntax

APPEND BLANK [<expN>]

See Also

@...GET, APPEND, APPEND FROM, CREATE, CREATE SCREEN, CHANGE, INSERT, SET CARRY, FMT()

Description

The APPEND BLANK command adds a blank record to the end of the active table. Default information can automatically be inserted into the record via the Applications Data Dictionary (ADD). Using this method can greatly increase performance of batch processes which use the APPEND BLANK command followed by the REPLACE command. If the numeric expression <expN> is specified, then <expN> blank records will be added.

Example

use patrons index names
append blank 10

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

APPEND FROM

Class

Table Basics

Purpose

Append records to the active table from another table or external file

Syntax

```
APPEND FROM <filename> | (<expC1>) [ENCRYPTION <expC2>]  
[FOR <condition>]  
[WHILE <condition>]  
[[TYPE] SDF | FIXED | DELIMITED | DELIMITED WITH BLANK | DELIMITED WITH <delimiter>]
```

See Also

COPY, APPEND, APPEND FROM ARRAY, CREATE BRIDGE, COPY TO, DECRYPT, ENCRYPT, SET ENCRYPTION

Description

The APPEND FROM command adds records from another file to the end of the active table. The FROM file can be in any one of the formats shown following the TYPE option. The <filename> can be substituted with a <expC1>, enclosed in round brackets, which returns a valid filename. The <filename> can include the encryption key for encrypted database tables. The three part comma-separated key should be enclosed in angled brackets and appended to the filename, e.g. mytable<key_1,key_2,key_3>.

If the FROM file is another Recital table, it cannot be open and active at the time of the APPEND. When processing the FROM file, the Recital/4GL only copies fields which exist in both of the tables. If the field in the FROM file is longer than the field in the active table it will be truncated. If a field in the FROM file is shorter than that in the active table, then the field in the active table is padded with blanks if it is a character field, otherwise it is converted to the new width. If no file extension is specified, then a '.dbf' extension is assumed. Records marked for deletion are not appended.

If the active table is shared, then the table and any associated index files will be locked as each record is appended. The locks are only applied as each record is added to the table, and not enforced for the full duration of the append operation, to provide optimum concurrent access to the table. Any indexes currently associated with the table will be automatically updated as the new records are added.

If a FILTER <condition> is currently active, then only those records in the FROM file which satisfy the specified <condition> will be appended.

ENCRYPTION <expC2>

If the FROM table is encrypted, its DES3 encryption key must be entered correctly before the data can be accessed. The key can be specified using ENCRYPTION <expC2>, where <expC2> is the 3 part encryption key, e.g. "key_1,key_2,key_3". The SET ENCRYPTION command allows a default encryption key to be defined. If the ENCRYPTION <expC2> clause is not specified and the key is not included in the <filename>, this default key will be used. If the default key is not the correct key for the FROM table, an error will be given. If no default key is active, a dialog box will be displayed in Recital Terminal Developer to allow the user to enter the key.

FOR <condition>

If the FOR <condition> clause is specified, then only those records in the FROM file which satisfy the specified <condition> will be appended.

WHILE <condition>

The WHILE <condition> clause can be used to restrict the range of records which are appended. When the <condition> becomes false, the APPEND operation will stop.

TYPE FIXED

If the FIXED keyword is specified, then the FROM file must contain fixed length records, where each field is exactly the same width as that in the active table. A FIXED file does not contain a deletion marker as the first character of each record. If no file extension is specified, then '.txt' is assumed. FIXED files can be created with the Recital/4GL COPY...FIXED command, or they can be created by an external program written in another programming language (e.g. C, PASCAL, FORTRAN).

TYPE DELIMITED

If the DELIMITED option is used, each record ends with a carriage return/line feed. If no WITH has been specified in the DELIMITED option, then a ',' will separate each field, and character fields will be surrounded by "" double quotes. If no file extension is specified, then a '.txt' extension is assumed. DELIMITED files can be created using the Recital/4GL COPY...DELIMITED command or they too can be created by an external program.

SDF

If the SDF keyword is specified then records from a text file which end with a carriage return/line feed can be appended. The maximum length of the text line used with APPEND FROM...SDF is 8192 characters. On UNIX, the carriage return is not present. On OpenVMS the records are variable length text records. If there are any binary fields in the FROM file, then the SDF file is treated as being in FIXED format.

Example

```
use patrons
append from system type sdf
append from textfile type delimited
append from transactions for code <> "D"
```

// Another Example

```
use payroll
append from (iif(dow(date())>5, "weekend.dbf",;
    "weekday.dbf") for amount > 100
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

APPEND FROM ARRAY

Class

Fields and Records

Purpose

Append records to current table from an array

Syntax

```
APPEND FROM ARRAY <array>
[FOR <condition>]
[WHILE <condition>]
[REINDEX]
```

See Also

COPY TO ARRAY, APPEND FROM, DECLARE, DIMENSION, GATHER, PRIVATE, PUBLIC, RELEASE, RESTORE FROM, SAVE TO, SCATTER, AAVERAGE(), ACHOICE(), ACOPY(), ADEL(), ADIR(), AFIELDS(), AFILL(), AINS(), ALLEN(), AMAX(), AMIN(), ASCAN(), ASORT(), ASUM(), AVERAGE()

Description

The APPEND FROM ARRAY command allows you to append records to the current table from the contents of a previously declared two-dimensional array of the specified name, <array>. The data types and sizes of elements in the rows of the arrays must correspond to the fields in the table.

FOR <condition>

If the FOR <condition> clause is specified, only those elements in the rows which satisfy the specified <condition> will be appended.

WHILE <condition>

The WHILE <condition> clause can be used to restrict the number of records appended. When the condition becomes false, the APPEND FROM ARRAY operation will stop.

REINDEX

The REINDEX keyword can be used to automatically call the REINDEX command after the APPEND FROM ARRAY has completed.

Example

```
use suppliers
declare cust[reccount(), fcount()]
copy to array cust
copy structure to customer
use customer
append from array cust for also_cust = .T.
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

APPEND MEMO

Class

Memos

Purpose

Add a text file to a memo field

Syntax

APPEND MEMO <memo fieldname> FROM <filename> [OVERWRITE]

See Also

COPY MEMO, MEMOREAD(), MEMOWRITE(), SET MEMOFORMAT

Description

The APPEND MEMO command adds the specified text file to the memo field of the current record. The <memo fieldname> of the current record in the active table is appended with the text in the file specified by <filename>. If no file extension is specified with <filename>, a '.txt' extension is assumed.

OVERWRITE

The optional OVERWRITE keyword causes the APPEND MEMO command to write over pre-existing text in the memo. Without the OVERWRITE keyword text will be appended to the memo field.

Example

append memo minutes from meeting

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

ASSERT

Class

Error Handling and Debugging

Purpose

Display a message dialog with options when a condition evaluates to False

Syntax

ASSERT <expL> [MESSAGE <expC>]

See Also

DEBUG, SET ASSERTS, SET COMPILE

Description

The ASSERT command is used for program debugging purposes: to display a message dialog with options when a condition evaluates to False (.F.). The condition is specified in <expL>. An optional message can be specified using the MESSAGE <expC> clause. If no message is specified; the default message is:

Assertion failed on line <#> of
procedure <prg>.

The dialog has four buttons, offering the following options:

Button	Option
Debug	Suspends program execution and starts the Debugger. The Debug option is only available if the program is being run uncompiled.
Cancel	Stops program execution.
Ignore	Continues program execution.
Ignore All	Continues program execution and issue SET ASSERTS OFF, causing subsequent asserts to be ignored.

If SET ASSERTS is OFF, asserts are ignored. SET ASSERTS is OFF by default.

Example

```
set asserts on
parameters para1, para2, para3
assert pcount() = 3 message [3 parameters required]
// code continues
return
```

Products

Recital Mirage Server, Recital Terminal Developer

AVERAGE

Class

Fields and Records

Purpose

Calculate the average of specified numeric expressions

Syntax

```
AVERAGE [<scope>] <exp> [,<exp>...]  
[FOR <condition>]  
[WHILE <condition>]  
[TO <memvar-list> | TO ARRAY <array name>]
```

See Also

SUM, COUNT, TOTAL, AVERAGE()

Description

The AVERAGE command calculates the arithmetic mean of all the specified numeric expressions. All records in the currently selected table are averaged unless the <scope> is specified.

FOR <condition>

If the FOR clause is specified, then only those records matching the specified <condition> are averaged.

WHILE <condition>

The WHILE is used to restrict the range of records processed while the specified <condition> is true. When used in conjunction with the SEEK or LOCATE commands, it gives a quick way of averaging selected records. When the WHILE clause is used, the <scope> will default to REST unless otherwise specified.

TO <memvar>

If TO <memvar> is specified then the result of AVERAGE will be stored in the specified memory variable. If the variable does not exist it will be created.

TO ARRAY <array>

The TO ARRAY clause is used to store results in a pre-defined one-dimensional array. The result of the first numeric expression is placed in the first array element, the second result is placed in the second element, and so on. If there are fewer elements than expressions, the AVERAGE command will only store results for which there are elements. If there are more elements than expressions, the remaining elements are left empty.

Example

```
use patrons  
average seats, cost;  
  to avg_seats, avg_cost;  
  for event = "PHANTOM"
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

BEGIN SEQUENCE

Class

Error Handling and Debugging

Purpose

Start error-handling block

Syntax

```
BEGIN SEQUENCE
<command>
[BREAK]
<command>
END SEQUENCE
```

See Also

ON ERROR

Description

The main purpose of the BEGIN SEQUENCE ... BREAK ... END SEQUENCE construct is to allow for programmable error handling. The commands that follow the END SEQUENCE command will normally be devoted to error handling and will be executed immediately.

BREAK

The BREAK keyword can be used whenever the program commands detect an error. For this purpose, BREAK may appear at any depth within nested procedures. When BREAK is encountered, all stacked procedures will unwind automatically.

Example

```
error = .F.
begin sequence
  do while not eof()
    if empty(name)
      rec = recno()
      error = .T.
      break
    else
      display name
    endif
    skip
  enddo
end sequence
if error
  set message to "Error: No name in record; &rec."
else
  set message to "End of file reached."
endif
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

BEGIN TRANSACTION

Class

Transaction Processing

Purpose

Begin transaction Before Image Journaling

Syntax

```
BEGIN TRANSACTION [<path name>]
<commands>
END TRANSACTION
```

See Also

SET ROLLBACK, ISMARKED(), RESET IN, ROLLBACK, ROLLBACK(), COMPLETED()

Description

The BEGIN TRANSACTION command is used to flag the beginning of a transaction for Before Image Journaling (BIJ). A 'transaction' consists of all file modifications that occur within the commands BEGIN TRANSACTION and END TRANSACTION.

When BEGIN TRANSACTION is issued all currently open files and all files opened between BEGIN and END TRANSACTION will have BIJ invoked automatically. If BIJ is not required on a particular table then the RESET IN command should be issued for the relevant workarea so that journaling will no longer occur in that workarea. The journals are stored in a log file (with a file extension of '.log') which the Recital/4GL generates automatically. You can optionally specify the disk and directory <path name> where the log file will be stored when the BEGIN TRANSACTION command is issued.

A "rollback" causes record contents to be restored to their value before modification (i.e. at the time BEGIN TRANSACTION was issued). This is particularly useful if an error occurs during a program that modifies files.

If SET ROLLBACK is ON, the Recital/4GL will automatically execute the ROLLBACK command if an error occurs during the transaction process. Otherwise, error trapping should be handled manually using the ON ERROR command.

The COMPLETED() function can be used after the END TRANSACTION command to determine if any errors occurred during processing of the commands between BEGIN and END TRANSACTION.

Please note that the following commands are not allowed during a transaction:

```
CLEAR ALL
CLOSE ALL
CLOSE DATABASE
CLOSE INDEX
MODIFY STRUCTURE
PACK
ZAP
```

Example

```
procedure recovery
rollback
if rollback()
    dialog box "Rollback was ok."
else
```

```
        dialog box "Rollback not completed."
    endif
return

use accounts
on error do recovery
begin transaction
    delete first 15
    insert
    replace all t1 with (t2*t3)/100
    list
end transaction
if completed()
    dialog box "Transaction completed"
else
    dialog box "Errors occurred during transaction"
endif
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

BLANK

Class

Fields and Records

Purpose

Fill fields with blanks

Syntax

BLANK [<scope>]
[FIELDS <field list> | LIKE | EXCEPT <skeleton>]
[REINDEX]
[FOR <condition>]
[WHILE <condition>]

See Also

APPEND BLANK, EMPTY(), ISBLANK(), REPLACE

Description

The BLANK command is used to fill a field or fields with empty values. This command is often used to empty out records that have been marked for deletion. This allows the record to be re-used without the need to perform an APPEND BLANK. Fields in the current record of the active table will be blanked, unless a <scope> is specified.

The way that the BLANK command fills each field depends on its data type. The following table describes how each data type is filled with blanks.

Data Type	Description
Character	All spaces
Data	Empty date
Logical	.F.
Memo	Length of 0
Numeric	0

FIELDS <expC>

You may specify which fields to fill with blanks with a field list of comma-separated fieldnames, or a pattern matching skeleton to blank those fields whose contents either match (LIKE), or do not match (EXCEPT) the specified pattern. If no fields or patterns are specified, the BLANK command will fill all fields in the current record with empty values.

FOR <condition>

Use a FOR <condition> to blank the specified fields in records where a certain condition is true.

WHILE <condition>

To blank fields until a condition is false, use the WHILE <condition>. Blanking a field that is involved in a WHILE <condition> is not recommended as the order of the records will be changed for each key field blanked.

Example

blank all for deleted()

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

BROWSE

Class

Screen Forms

Purpose

User-Interface tool browse

Syntax

BROWSE

[COLOR <color code> | COLOR SCHEME <color-scheme>]

[COMPRESS]

[FIELDS <field list>]

[FOR <expL1>]

[FORMAT]

[FREEZE <fieldname>]

[KEY <key expression>]

[LAST | NOINIT]

[LOCK<expN1>]

[NOAPPEND]

[NOCLEAR]

[NODELETE]

[NOEDIT | NOMODIFY]

[NOFOLLOW]

[NOMENU]

[NOWAIT]

[OVERLAY]

[STYLE <expC1>]

[TIMEOUT <expN2>]

[TITLE <expC2>]

[VALID [:F] <expL2> [ERROR <expC3>]]

[WHEN <expL3>]

[WIDTH<expN3>]

[WINDOW <window name>]

See Also

@...GET, CHANGE, SET EDITFIELD, SET FORMAT, APPEND, EDIT

Description

The BROWSE user-interface tool displays the records from the currently active table in a grid window. Browse can also have an edit window, and/or a memo window active at the same time. If the table being browsed is related to other tables, then these related tables can also be edited from within the edit widow. Full automatic record locking in the browse window is performed on shared tables.

All the dictionary attributes defined in the Application Data Dictionary (ADD) are active in BROWSE. From the ADD, field validation, editing pictures, static and dynamic choice lists, calculated and default attributes can be defined. Security and field protection can also be defined. Enabling or disabling of update, append, delete and hidden fields by user and group ID is available.

Keyword	Description
COLOR <color code> COLOR SCHEME <color-scheme>	Specifies a set of color pairs or the number of an existing color scheme in the COLOR SCHEME to change the default color of BROWSE. A color scheme is a set of 10 predefined color pairs. The color pairs in a color scheme can be changed with the SET COLOR OF SCHEME. 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.
COMPRESS	Places column headings on the top line of the BROWSE work surface, with no separating line above the records, to allow more records to be displayed.
FIELDS <field list>	The comma-separated <field list> enables you to: specify the order in which the specified fields display and incorporate special handling options for each field. Please see below for these options.

<field> [:R] [:column width]
 [:B = <expr>, <expr> [:F]]
 [:C = <expC> | <@alias>, <expC>]
 [:V = <expL> [:F:] [:E = <expC>]
 [:P = <expC>]
 [:H = <expC>]
 [:W = <expL>]

Each field in the <field list> may be followed by special handling options. A delimiter precedes each option, either : or \.

Option	Description
:R	Fields are read-only, so may be viewed, but not edited.
:<column width>	The <column width> allows you to specify the numeric display width of a field.
:B = <expr>, <expr> [:F]	Restricts valid data input to a range of values. The expressions specified must match the data type of the field and represent low <expr1> and high <expr1> values. The specified range is only enforced when the field value changes unless the :F option is used.
:C = <expC> <@alias>, <expC>	The Choicelist option associates a pop-up choicelist or a User Defined Function (UDF) with a field. The choicelist may be either 'static', a string of comma-separated choices, 'dynamic', @alias,string or a UDF, a function name prefixed with a question mark.
:V = <expL> [:E = <expC>] [:F]	Validates the field value based on the evaluation of <expL>. If <expL> returns true (.T.), the field value is considered valid. If <expL> returns false (.F.), the value is rejected and an error message displayed. If the :E option is specified, <expC> replaces the default error message. The validation is only enforced when the field value changes unless the :F option is used. This option does not work on memo fields.
:H = <expC>	Replaces the default field headings, field name or description, with the character expression <expC>.
:M = <expC>	Displays <expC> in the message line when the field is selected.
:P = <expC>	Sets the picture format to <expC>. For more information on picture template characters and picture formatting, see @...GET.
:W = <expL>	Only allows data entry when <expL> evaluates to true (.T.), otherwise the field is read-only.

The <field list> can also include memory variables calculated by specified expressions. These take the format <memvar> = <expression>. The memory variables may not be manually updated in the BROWSE.

Keyword	Description
FOR <condition>	Restricts the BROWSE to those records matching the <condition>.
FORMAT	Causes BROWSE to assume the triggers and @...GET attributes as defined in the currently active format file.
FREEZE <fieldname>	Restricts cursor movement to the column containing the field <fieldname>.
KEY <key expression>	Restricts the BROWSE to those records whose index key matches the specified <key expression>.
LAST NOINIT	Causes BROWSE to inherit the settings and clauses from the previous BROWSE.
LOCK <expN1>	Locks the specified number of columns <expN1> from the left. As you pan right and left, the locked columns remain on the screen and the columns to the right are panned.
NOAPPEND	Disables the appending of new records into the table.
NOCLEAR	Leaves the BROWSE window displayed on the screen when you exit.
NODELETE	Disables record deletion.
NOEDIT NOMODIFY	Disables record editing, providing read-only access to the table.
NOFOLLOW	Prevents repositioning in the workspace when an index key field value is edited.
NOMENU	The NOMENU clause disables the BROWSE menu system.
NOWAIT	Continues program execution immediately after the BROWSE command has been issued, rather than when the BROWSE is exited. The BROWSE is displayed, but not active.
OVERLAY	Allows format files to be overlaid on the BROWSE workspace rather than clearing the screen.
STYLE <expC1>	Specify the font style, <expC1>, to be used. Recital Mirage only.
TIMEOUT <expN2>	Specify how many seconds, <expN2>, BROWSE should wait for input before closing automatically.
TITLE <expC2>	Specify a title, <expC2> to be displayed. Recital Mirage only.
VALID [:F] <expL2> [ERROR <expC3>]	Allows record-level validation in a Browse window. VALID is executed only if a change is made to the record and an attempt is made to move the cursor to another record. The VALID clause is not executed if the only change is to a memo field. If <expL2> evaluates to true (.T.), the cursor can be moved to another record. If <expL2> evaluates to false (.F.), the cursor remains in the current field, and the system message "Invalid input" appears. An alternative error message can be specified, using the optional ERROR clause and the character expression <expC3> will be displayed. To force the VALID clause to execute before moving to another record even if no changes have been made, include the optional :F.
WHEN <expL3>	FoxPro syntax compatibility only.
WIDTH	Specifies a display size for all fields. This option will override the width specified at the field level.
WINDOW <window name>	Causes the BROWSE output window to take on the characteristics of the named <window name> pre-defined window. The specified window need not be visible or active.

Example

use demo

browse field title:c="MR.Mrs,Miss,Mr",;

last_name:w=10,;

first_name

Products

Recital Mirage Server, Recital Terminal Developer

BUILD

Class

Table Basics

Purpose

Exports bridge files, tables and their associated files in ASCII format to allow them to be transferred to a binary incompatible platform

Syntax

BUILD <filename> [INTO <directory>]

See Also

INSTALL, SET FILETYPE

Description

The BUILD command exports tables and their associated memo, dictionary and multiple index files into ASCII format to allow them to be transferred to a binary incompatible format. The import on the target machine requires the use of the INSTALL command.

<filename>

The <filename> is the name of a '.xaf' file. This is a text file in the following format, assuming the files data1.dbf and data2.dbf are the tables to be converted:

```
dbf,data1.dbf
dbf,data2.dbf
```

The dictionary, memo and multiple index file information is picked up automatically.

The BUILD command creates a single '.xat' file with information about all the files being exported. The '.xat' file has the same name as <filename> and is specified in the INSTALL command on the target machine. A '.xds' file is created for each table specified, containing the table structure information. A '.xmd' file is created for each table that has a memo file and a '.xdd' file for each table's data. These files contain the information required to rebuild the files on the target machine.

Recital Bridges can also be transferred. The name of the bridge file should be specified in the '.xaf' file, e.g.

```
brg,cisamdemo.dbf
```

Note: although the default extension for bridge files is '.brg', many bridge files are given a '.dbf' extension to allow them to be opened with the USE command by specifying the basename only. This is the case for the cisamdemo.dbf C-ISAM bridge file included in the Linux and UNIX distributions.

The matching '.str' file and the target data files are automatically picked up. For the cisamdemo.dbf bridge, this means that the following files are included:

```
cisamdemo.str
cisamdemo.dat
cisamdemo.idx
```

INTO <directory>

If the optional INTO <directory> clause is used, the export files will be created in the specified directory. If the directory does not exist, it will be created.

Example

On Source machine demo.xaf contains the following lines:

dbf,customer.rdb
dbf,accounts.rdb
dbf,state.rdb
dbf,product.rdb
brg,cisamdemo.dbf

- build demo into ./transfer

On Target Machine, once files have been transferred

- install demo from ./transfer

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

CALCULATE

Class

Fields and Records

Purpose

Computes certain financial and statistical calculations for a range of records

Syntax

CALCULATE [<scope>] <options list>

[FOR <condition>]

[WHILE <condition>]

[TO <memvar-list>]

[TO ARRAY <array name>]

See Also

AVERAGE, COUNT, TOTAL, SUM, REPORT

Description

The CALCULATE command computes certain financial and statistical functions against the records of the current and related tables. If no <scope> is specified, all records will be processed. If a WHILE <condition> is specified, the <scope> will default to REST.

The options list consists of one or more of the following CALCULATE functions:

Function	Description
AVG(<expN>)	Returns the arithmetic mean of <expN>.
CNT()	Returns the number of records selected during a calculate command.
MAX(<expN>)	Returns the maximum number of <expN>.
MIN(<expN>)	Returns the minimum number of <expN>.
NPV(<expN1>,<expN2>,<expN3>)	Calculates net present. The value <expN1> is the interest percentage represented by a decimal number. (For example, 0.15 is 15%). The value <expN2> represents the periodic payment amounts. <expN3> is a numeric value that represents an initial investment. The initial value should be a negative number since it represents cash outflow
STD(<expN>)	Calculates the standard deviation of <expN>. Standard deviation measures the degree to which each individual score in a sample varies from the mean of the scores in the sample
SUM(<expN>)	Returns the sum of <expN>.
VAR(<expN>)	Calculates the population variance of <expN>

FOR <condition>

If a FOR <condition> is specified, only those records matching the <condition> are included in the calculations.

WHILE <condition>

The WHILE <condition> can be used in conjunction with the FIND or SEEK commands, and the REST <scope>, to restrict the number of records which are processed and therefore optimize the performance of the CALCULATE command.

TO <memvar>

The results of the CALCULATE operation will be displayed on the screen if the TO <memvar-list> list clause is not specified and SET TALK is ON. If SET TALK is OFF, the results will not be displayed, so the <memvar list> must be specified to make the results accessible. The memory variables do not need to be predefined. The memory variables are specified in a comma-separated list and their number must match the number of functions used

TO <array>

The TO ARRAY clause is used to store results in a pre-defined one-dimensional array. The result of the first numeric expression is placed in the first array element, the second result is placed in the second element, and so on. If there are fewer elements than expressions, the CALCULATE command will only store results for which there are elements. If there are more elements than expressions, the remaining elements are left empty.

Example

```
// Calculate the average and variance.
use accounts
seek "a1456"
calculate;
    cnt(),avg(ord_value),sum (ord_value),;
    var(ord_value) to cnt,avg,sum,var;
    while acc_no="A1456"
// Calculate the net present value.
use payments
m_rate = 0.12
m_initial = -40000
calculate npv(m_rate, payment, m_initial) to m_npv
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

CALCULATOR

Class

Screen Dialogs

Purpose

Display a popup calculator

Syntax

CALCULATOR

[AT <expN1>,<expN2>]

[BOLD]

See Also

CALENDAR, MENUITEM(), VAL()

Description

The CALCULATOR command displays a calculator that can be used to perform addition, subtraction, multiplication, division and percentage operations entered using the keyboard.

The calculator is operated by pressing the appropriate mathematical operators and digits on the keyboard. Percentage calculations are performed by loading the value to be calculated before entering the percentage. If the value is already displayed in the calculator, press the '=' operator, then press, for example: '50 % =' to calculate fifty percent of that value. If the value is not already displayed in the calculator, type for example '2000 =' to indicate that you are going to calculate a percentage of two thousand, then enter the percentage amount followed by the '=' operator. If the [EXIT/SAVE] key is pressed the MENUITEM() function will return the result from the CALCULATOR as a character string. The VAL() function can be used to convert a character string to a numeric. If the [ABANDON] key is pressed, MENUITEM() will return a null string.

When editing a numeric field, pressing the [HELP] key displays the CALCULATOR. If you save the result of the currently displayed calculation, using the [EXIT/SAVE] key, then the value is returned to the field.

Keyword	Description
AT <expN1>,<expN2>	Specify the calculator top left row and column coordinates. If no coordinates are specified, the calculator is centered on the screen.
BOLD	Display the calculator in bold, if supported by the terminal display.

Example

calculator at 2,30

Products

Recital Terminal Developer

CALENDAR

Class

Screen Dialogs

Purpose

Display a pop-up monthly calendar relating to today or to a specified date

Syntax

CALENDAR

[AT <expN1>,<expN2>]

[BOLD]

[CLEAR]

[DATE <expD>]

[LABEL <expC>]

[MENU [SCHEDULE]]

See Also

@...GET, CALCULATOR, DIARY, SET SCHEDULE

Description

The CALENDAR command displays a pop-up calendar page for the current month, with today's date highlighted. Alternatively, it can be used to display the monthly calendar relating to a specified date.

When editing a date field, pressing the [HELP] key invokes the CALENDAR. Selecting a date from the CALENDAR causes the date value to be returned to the field.

Keyword	Description
AT <expN1>,<expN2>	Specify the calendar top left row and column coordinates. If no coordinates are specified, the calendar is centered on the screen.
BOLD	Display the calendar in bold, if supported by the terminal display.
CLEAR	Clears the screen behind the calendar display area.
DATE <expD>	Specify a date to be displayed on the calendar. The dates from that month will be shown and the date <expD> itself will be highlighted.
LABEL <expC>	Specify a character expression <expC> to replace the default title "Calendar".
MENU	Allows the calendar to be 'paged' between months. You can locate a specific date using the [FIND] key and select a date by pressing [EXIT/SAVE] when the required date is highlighted. The MENUITEM() function will return the selected date from the CALENDAR, or a null string if no date is selected.
MENU SCHEDULE	Also displays the time scheduler, allowing appointment and other details to be associated with a specific date and time.

Example

calendar menu at 10,10

Products

Recital Terminal Developer

CANCEL

Class

Applications

Purpose

Cancel processing of commands in a program file

Syntax

CANCEL

See Also

RETURN, RETRY, QUIT

Description

The CANCEL command cancels the execution of the current program, closes all open command files and returns the system to the interactive command mode. If the current program was called from the Operating System prompt, control will be returned to the Operating System.

Example

cancel

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

CHANGE

Class

Screen Forms

Purpose

Full screen editing of records through a form

Syntax

CHANGE

[<scope>]

[FIELDS <field list>]

[FOR <condition>]

[KEY <exp>]

[NOAPPEND]

[NOCLEAR]

[NODELETE]

[NOEDIT]

[NOFOLLOW]

[NOINIT]

[NOMENU]

[NOORGANIZE]

[NOWAIT]

[WHILE <condition>]

See Also

APPEND, BROWSE, EDIT, READ, SET FORMAT, SET UPDATE

Description

The CHANGE command provides the facility to edit records in the active table that match specified record selection criteria. The CHANGE command uses a default form to display and allow updating of the specified records.

When memo fields are displayed, the memo field label is in lower case if the field is empty, and upper case if the field contains text. If a memo window (SET WINDOW OF MEMO) is active, then the memo can be displayed and edited in the specified window. Otherwise, pressing the [INSERT] key or the [HELP] key on the memo field will popup a notepad editor. Pressing the [HELP] key within the notepad editor displays the memo editing keys available. These keys include facilities for reading from and writing to external files and printing on the system printer.

You can design your own forms using the Forms Designer (CREATE SCREEN). The Form Designer will automatically generate a format file that can contain @...GET, @...SAY, @...MENU and other display objects. This form will be used by the CHANGE command if it has been activated using the SET FORMAT command.

If the active table is shared, then automatic record locking will take place for each of the tables referenced on the form. The records are automatically unlocked when you skip to another record, or exit from the CHANGE command. If UPDATE is ON then 'Upd' appears in the 3rd box of the status bar at the bottom of the screen. When UPDATE is ON records are automatically locked before they are read from the database. When UPDATE is OFF, no record locking is performed. It is recommended that you toggle UPDATE mode ON when you have a record displayed on the form which you want to update, and leave UPDATE OFF while you are browsing through your records. If SET LOCKWAIT is OFF, then whenever an attempt is made to lock a record that is already locked by another user, you are given the choice of waiting for the lock or continuing in query mode.

When UPDATE is toggled ON, the form is refreshed with the most current information from the table. As only one user at a time can be in UPDATE mode on a particular record, the information displayed on the form is always current at the time of editing. The changed data on the form is written to the table and the lock released when the [EXIT/SAVE], [NEXT SCREEN], [PREVIOUS SCREEN] or [MENUBAR] keys are pressed. If the [ABANDON] key is pressed and changes have been made to the data, SET VERIFY ON causes a dialog box to be displayed asking for confirmation.

Keyword	Description
<scope>	If no <scope> is specified, CHANGE is activated on the current record and all records are accessible using the [NEXT RECORD] and [PREVIOUS RECORD] keys.
FIELDS <field list>	The active fields can be restricted to those specified in the comma separated <field list>.
FOR <condition>	Record navigation is restricted to those records that match the <condition>.
KEY <exp>	The active records can be restricted to those that match the specified <exp>. The <exp> must be based on the index key of the current master index.
NOAPPEND	The ability to append records is disabled.
NOCLEAR	Erasing of the screen on entry and exit from CHANGE is disabled.
NODELETE	The ability to delete records is disabled.
NOEDIT	The ability to edit the record is disabled. Access is read only.
NOFOLLOW	The NOFOLLOW key word determines whether the record pointer follows a record whose position has changed. A record's position may change when fields that are part of the master index expression are modified. Normally, upon update, the record pointer will be moved to the new position, NOFOLLOW disables this.
NOINIT	The work surface retains the keywords and specifications from the last CHANGE session.
NOMENU	The menu bar is disabled in the default CHANGE form.
NOORGANIZE	Language compatibility only
NOWAIT	Control is returned to the executing program without waiting for the user to exit the CHANGE worksurface.
WHILE <condition>	Record navigation is restricted to those records that match the specified <condition>. Navigation cannot continue beyond a record that does not match the <condition>.

The following keys are active in CHANGE:

Key	Action
ABANDON	Discard current changes then exit from the form
CURSOR DOWN	Skip to next field
CURSOR LEFT	Skip to previous field
CURSOR RIGHT	Skip to next field
CURSOR UP	Skip to previous field
DELETE FIELD	Initialize field
DELETE RECORD	Delete / Recall the record
EDIT FIELD	Enter field edit mode
EXIT/SAVE	Write current changes then exit from the form
FIND	Find record by key or condition
FIND NEXT	Find next record matching specified key or condition
HELP	Activate pop-up help
MENUBAR	Activate the CHANGE menu bar
NEXT RECORD	Write current changes then skip to next record
PREVIOUS RECORD	Write current changes then skip to next record

REFRESH	Redraw the form
TAB	Toggle function key menu on and off
UPDATE MODE	Toggle update mode on and off

If SET MOUSE is ON, cursor keys will move the cursor anywhere on the screen rather than just from field to field. If SET NAVIGATE is ON, the cursor moves to fields following the direction specified by the key being pressed, rather than following the order of the GETS on the form. When the RETURN key is pressed, the cursor moves to the nearest field when SET NAVIGATE is ON.

The following keys are active in field edit mode:

Key	Action
BACKSPACE	Delete character before cursor
CURSOR LEFT	Skip to previous character
CURSOR RIGHT	Skip to next character
DELETE CHAR	Delete character under cursor
DELETE FIELD	Delete from cursor to end of field
DELETE WORD	Delete current word
INSERT MODE	Toggle insert / overwrite mode
WORD LEFT	Skip left a word
WORD RIGHT	Skip right a word

The following menu options are available from the CHANGE menu bar in the default form:

Menu Item	Action
Descriptions	Toggle the field descriptions on and off
Top	Position to the top of the table or selected records
Bottom	Position to the bottom of the table or selected records
Order	Select index order
Query	Query the table for selected records
Help	Activate on-line help system

Example

use demo
change

Products

Recital Mirage Server, Recital Terminal Developer

CLASS

Class

Objects

Purpose

Create a user-defined class

Syntax

```
CLASS <class name>
[OF <base class> [, ...]]
[PUBLIC | PRIVATE]
[DYNAMIC]
[NOTIFY]
[PROPAGATE]
ENDCLASS
```

See Also

CLASS - METHODS, CLASS - PARAMETERS, CLASS - PROPERTIES, CLASS - SCOPING, DEFINE CLASS, METHOD, ADDPROPERTY(), CREATEOBJECT(), DODEFAULT(), NEWOBJECT(), REMOVEPROPERTY()

Description

The Class construct is a fundamental part of the object-oriented programming language (OOPS) which is part of the Recital/4GL. The OOPS language supports encapsulation, inheritance (single and multiple), polymorphism, complex member scoping, property notification, and method propagation.

Fundamental to object-oriented programming is the concept of Objects and classes. An object is a self-contained unit of data and functions that manipulate that data. A class is a specification of an object. A class contains memory variable specifications known as properties and functions that perform actions on the object known as methods. An object is an instance of a class.

The NEW operator is used to define a new object based on a class. The class name must be postfixed with parentheses when the new operator is used. The syntax is therefore as follows:

```
<object> = NEW <class>()
```

e.g.

```
myobject = new myclass()
```

The CLASS...ENDCLASS construct is used to create a user-defined class. The beginning of a class is specified with CLASS <class name>, where <class name> can be any valid name up to 32 characters. The ENDCLASS command is used to complete the class construct. The CLASS...ENDCLASS construct is built using the commands describe in this section. Any Recital/4GL command can be used in building methods in the class, however these commands cannot be used outside the method definition inside a CLASS...ENDCLASS construct.

OF <base-name> [, ...]

A key feature of the object-oriented code is reusability through a mechanism called inheritance, that is, one class can inherit the members and their implementation from another class. Building new classes out of existing classes allows for the reusing of proven classes and the incorporation of system classes into user defined classes. Inheritance enables developers to build a hierarchy of descending objects. The inheriting class is called a derived class, and the class from which the derived class inherits is called a base class. The

OF clause is used to inherit the <base name>. You can inherit multiple classes by specifying a class name comma separated list.

PUBLIC | PRIVATE

A user-defined class by default is defined as PRIVATE, which means that it will be visible only at the level where the class is instantiated. If the PUBLIC clause is specified, then when the class is instantiated, it will be visible at all levels until it is RELEASED from memory.

DYNAMIC

The DYNAMIC keyword provides the ability to add property names to objects at runtime.

NOTIFY

The Recital/4GL supports what is known as Property notification. If you specify the NOTIFY clause all properties defined will have notify set. Property notification is an essential element in the real-time interaction with system objects. When a property that has been defined with the NOTIFY clause is read, and the class contains a method called GETPROPERTY, that method is called. The property name is passed to the GETPROPERTY method as a parameter.

PROPAGATE

Any methods which are called, which have the PROPAGATE attribute set, cause a cascading execution effect down through the class hierarchy. After the method is executed, a search is made in all of the sub-objects (if any) that are defined as properties within the current object. If a method with the same name is found and that method has the NOTIFY attribute specified, then that method is called.

The CONSTRUCTOR and the DESTRUCTOR methods always act as if sub-objects have the PROPAGATE and the NOTIFY attributes set. This allows any properties that have been specified as sub-objects to operate correctly when an object is created with the NEW operator.

All classes have an inbuilt ADDPROPERTY 'factory method'. This can be used as an alternative to the ADDPROPERTY() function to add properties to an object at runtime.

Example

```
class Null dynamic
  property CHARACTER
  property NUMERIC
endclass

oNULLDATA = new Null()
oNULLDATA.CHARACTER = ""
oNULLDATA.NUMERIC = 0
oNULLDATA.DATE = { / / }
oNULLDATA.LOGICAL = .f.
display memory
```

```

// Use of the Notify Keyword
class CharWin
    public notify:
        property IVISIBLE as logical
        property nROW, nCOL, nENDROW, nENDCOL
    public:
        property cWINDOW
        property cCOLOR

    method Constructor
    parameters cWINDOW, cCOLOR, nROW, nCOL, nENDROW, nENDCOL
    define window &cWINDOW ;
        from nROW, nCOL to nENDROW,nENDCOL ;
        color &cCOLOR panel float grow
        this.cWINDOW = cWINDOW
        this.cCOLOR = cCOLOR
        this.nROW = nROW
        this.nCOL = nCOL
        this.nENDROW = nENDROW
        this.nENDCOL = nENDCOL
    return && Constructor

    method SetProperty
    parameter cNAME
    if lower(cNAME) = [lvisible]
        if this.IVISIBLE
            activate window &(this.cWINDOW)
        else
            hide window &(this.cWINDOW)
        endif
    endif
    return && SetProperty
endclass

clear screen
oMYWIN = new CharWin("mywin", "w/r",2,20,15,60)
dialog box [Calling a NOTIFY property]
oMYWIN.IVISIBLE = .t.
?
? [ Welcome to the world of Objects]
?

```

//Example of Constructor & Destructor

```
class OpenTable
  property cALIAS
  property nRECNUM

  method Constructor
    parameters cTABLENAME, cTAGNAME
    local cTMPALIAS
    cTMPALIAS = basename(cTABLENAME)
    cTMPALIAS = iif(at('.',cTMPALIAS) = 0, ;
      cTMPALIAS, ;
      left(cTMPALIAS,at('.',cTMPALIAS) - 1))
    if select(cTMPALIAS) = 0
      use &(cTABLENAME + iif(empty(cTAGNAME), ", " ;
        " order " + cTAGNAME)) in workarea()
    else
      select select(cTABLENAME)
      set order tag &cTAGNAME
    endif
    this.cALIAS = alias()
    this.nRECNUM = recno()
    return && Constructor

  method Destructor
    close &(this.cALIAS)
    return && Destructor

endclass

set exclusive off
oCOMPANY = new OpenTable("/usr/recital/unixdeveloper/demo/state.rdb", "state")
? oCOMPANY.cALIAS
? oCOMPANY.nRECNUM
```

//Example of ADDPROPERTY method

```
class Box
endclass

oDIALOG = new Box()
oDIALOG.AddProperty("myprop", "hello world")
dialog box oDIALOG.myprop
release oDIALOG
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

CLASS - Methods

Class

Objects

Purpose

Define a method in a user-defined class

Syntax

METHOD <method name> [EXTERNAL]

RETURN

See Also

CLASS, CLASS - PARAMETERS, CLASS - PROPERTIES, CLASS - SCOPING, DEFINE CLASS, METHOD, ADDPROPERTY(), CREATEOBJECT(), DODEFAULT(), NEWOBJECT(), REMOVEPROPERTY()

Description

An object encapsulates properties and all of the methods that perform operations on the object. Encapsulation hides data within an object, and makes an object into a full self-contained operational unit. A class method is a self-contained function defined in the class, accessible only through an instantiation of the class.

You define the methods of a class using the METHOD command in the CLASS...ENDCLASS construct. The 'This.' operator is used to reference properties of the active object from within its methods. Parameters can be passed to a method, the PARAMETER statement can be used to define the parameters in the method.

<method name>

The <method name> must be unique name of up to 32 characters. The method is called using the "object.method()" syntax. You can create special methods in the class. When a new object is created, and the class contains a method called CONSTRUCTOR, that method is called to complete the process of creating the object. You can pass parameters to the CONSTRUCTOR method when you create the new object. The PARAMETER statement must be specified in the method for accepting parameters. When the object is released, and it contains a method called DESTRUCTOR, that method is called prior to the object storage being released. The Visual FoxPro equivalents are also supported: INIT for CONSTRUCTOR and DESTROY for DESTRUCTOR. When a property that has been defined with the NOTIFY clause is read, and the class contains a method called GETPROPERTY, that method is called. The property name is passed as an upper case character string to the GETPROPERTY method as a parameter. When a property that has been defined with the NOTIFY clause is updated, and the class contains a method called SETPROPERTY, that method is called. The property name, in upper case, and the new value are passed to the SETPROPERTY method as parameters.

EXTERNAL

Methods can be defined outside the CLASS...ENDCLASS construct with the METHOD command. The EXTERNAL clause is used to make an external method known to the CLASS...ENDCLASS construct. When a method is defined externally, its name should be preceded by the keyword METHOD, followed by the Class name, followed by two colon characters (e.g. Method MyClass::ExtMethod).

RETURN

The RETURN clause is used to specify the end of the method definition.

Example

```
// Example of External Method
class Box
    method DrawFrame external
endclass
```

```
Method Box::DrawFrame
parameters nX1, nY1, nX2, nY2, cFGCOL, cBGCOL
@nX1,nY1 clear to nX2,nY2
@nX1,nY1 fill to nX2,nY2 ;
    color &(cFGCOL + "/" + cBGCOL)
@nX1,nY1 to nX2,nY2 ;
    color &(cFGCOL + "/" + cBGCOL)
return && DrawFrame
```

```
oDIALOGOK = new Box()
oDIALOGOK.DrawFrame(5,25,12,54, "R", "Gr")
```

```
// Example of Visual FoxPro Method names
```

```
class Box
procedure Draw
    messagebox("This is the parent Draw Method")
endproc && Draw
endclass
```

```
class Dialog1 of Box
procedure Init
    messagebox("This is the object Init Method")
endproc
procedure Destroy
    messagebox("This is the object Destroy Method")
endproc
procedure Draw
    messagebox("This is the object Draw Method")
    dodefault()
endproc && Draw
endclass
```

```
oDIALOG = createobject("Dialog1")
oDIALOG.Draw()
oDIALOG.AddProperty("myprop", "hello world")
messagebox(oDIALOG.myprop)
release oDIALOG
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

CLASS - Parameters

Class

Objects

Purpose

Define a parameter list for a method inside a user-defined class

Syntax

```
PARAMETERS <name>  
[AS CHARACTER | NUMERIC | LOGICAL | DATE | <class name>]  
[, ...]
```

See Also

CLASS, CLASS - METHODS, CLASS - PROPERTIES, CLASS - SCOPING, DEFINE CLASS, METHOD, ADDPROPERTY(), CREATEOBJECT(), DODEFAULT(), NEWOBJECT(), REMOVEPROPERTY()

Description

The PARAMETER command is used inside the CLASS...ENDCLASS construct to define a parameter list for a method. You can pass parameters to methods just as you would to regular functions. If the method name is called CONSTRUCTOR then the parameters are passed to this method when you create the object.

<name>

The parameter <name> is the name of a memory variable or array.

AS CHARACTER | NUMERIC | LOGICAL | DATE

The data type of values assigned to a property within an object can be checked at run-time by associating a data type with the property in the class definition. The AS clause is used to perform data scoping. Once a variable has been defined to a specific data type, you cannot change its data type by assigning it a value from a different data type. Doing this will result in the run-time error "Data type mismatch".

AS <class name>

A property data type can inherit a class <class name>, and its members and their implementation from another class. Inheritance enables developers to build a hierarchy of descendant objects. The specification of a property data type as an existing class name provides for construction of a class hierarchy. The inheriting class is called a derived class, and the class that the derived class inherits is called a base class.

Example

```
class Box  
  method Draw  
    parameters nX1, nY1, nX2, nY2, cFGCOL, cBGCOL  
    @nX1,nY1 clear to nX2,nY2  
    @nX1,nY1 fill to nX2,nY2 ;  
      color &(cFGCOL + "/" + cBGCOL)  
    @nX1,nY1 to nX2,nY2 ;  
      color &(cFGCOL + "/" + cBGCOL)  
  return && Draw  
endclass
```



```

class Dialog1 of Box
  property IOK

  method Show
    parameters nX1, nY1, nX2, nY2, cMESSAGE

    this.Draw(nX1,nY1,nX2,nY2, "n", "bg")
    @nX1 + 2,nY1 + int((nY2 - nY1 - ;
      len(cMESSAGE))/2) say cMESSAGE color w+/bg
    @nX2 - 2,nY1 + int((nY2 - nY1 - 5)/2) ;
      menu [ \<Ok ]
    menu quit
    this.IOK = (not empty(menuitem()))
    @0,0 clear to 0,79
    return && Show
endclass

oDIALOGOK = new Dialog1()
oDIALOGOK.Show(5,25,12,54, "Completed")
? oDIALOGOK.IOK
?

```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

CLASS - Properties

Class

Objects

Purpose

Create properties in a user-defined class

Syntax

```
PROPERTY <memvar> | <array> | <dynamic array> [, ...]  
[AS CHARACTER | NUMERIC | LOGICAL | DATE | <class>] [, ...]
```

See Also

CLASS, CLASS - METHODS, CLASS - PARAMETERS, CLASS - PROPERTIES, CLASS - SCOPING, DEFINE CLASS, LOCAL, METHOD, PRIVATE, PUBLIC, STATIC, ADDPROPERTY(), CREATEOBJECT(), DODEFAULT(), NEWOBJECT(), REMOVEPROPERTY()

Description

The PROPERTY command is used inside the CLASS...ENDCLASS construct to define properties of a class. The PROPERTY command can create memory variables, arrays, and dynamic arrays. You reference properties in an object using the dot (.) member access operator. The full syntax is <object>.<property>.

The visibility of the properties of an object is governed by their scope. Unless otherwise specified, all properties defined within a class are public.

<memvar> | <array> | <dynamic array>

The property name is either a memory variable name <memvar>, or an array name <array>. Arrays are defined with the “arrayname[<expN>]” construct. The <expN> represents the number of array elements, if the number is not specified then the array is defined dynamic.

AS CHARACTER | NUMERIC | LOGICAL | DATE

The data type of values assigned to a property within an object can be checked at run-time by associating a data type with the property in the class definition. The AS clause is used to perform data scoping. Once a variable has been defined as a specific data type, you cannot change its data type by assigning it a value from a different data type: doing this will result in the run-time error “Data type mismatch”.

AS <class name>

A property data type can inherit a class <class name>, and its members and their implementation from another class. Inheritance enables developers to build a hierarchy of descendant objects. The specification of a property data type as an existing class name provides for construction of a class hierarchy. The inheriting class is called a derived class, and the class that the derived class inherits is called a base class.

Example

// Example of Properties

```
class Company  
    property cCOMPANY_NAME as character  
    property cCOMPANY_CODE as character  
    property aADDRESS[]  
    property cCOUNTRY as character  
    property cWWW  
    property cTEL  
    property cFAX  
endclass
```

```
oCOMPANY = new Company()
oCOMPANY.cCOMPANY_NAME = [Recital Corporation Inc]
oCOMPANY.aADDRESS.line1 = [85 Constitution Lane]
oCOMPANY.aADDRESS.line2 = [Danvers]
oCOMPANY.aADDRESS.line3 = [MA 01923]
oCOMPANY.cWWW = [http://www.recital.com]
display memory
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

CLASS - Scoping

Class

Objects

Purpose

Specify scoping in user-defined class

Syntax

LOCAL | PRIVATE | PUBLIC | STATIC

[NOTIFY] [PROPAGATE] :

See Also

CLASS, CLASS - METHODS, CLASS - PARAMETERS, CLASS - PROPERTIES, DEFINE CLASS, METHOD, ADDPROPERTY(), CREATEOBJECT(), DODEFAULT(), NEWOBJECT(), REMOVEPROPERTY()

Description

The visibility of the properties of an object is governed by their scope. The default visibility is public, but this can be changed by using one of the scoping commands. Once the scope is changed, it remains in effect on all property and method definitions that follow, until it is changed again or the ENDCLASS command is reached.

Scope	Description
LOCAL	Local properties and methods are visible only to the object where they are defined.
PRIVATE	Private properties and methods are visible only to the object where they are defined and to all methods and properties of objects called by the object.
PUBLIC	Public properties and methods are visible everywhere.
STATIC	Static property values are shared by all objects of the same class.

NOTIFY

The Recital/4GL supports what is known as Property notification. If you specify the NOTIFY clause all properties defined will have 'notify' set. Property notification is an essential element in the real-time interaction with system objects. When a property that has been defined with the NOTIFY clause is read, and the class contains a method called GETPROPERTY, that method is called. The property name is passed to the GETPROPERTY method as a parameter. When a property that has been defined with the NOTIFY clause is updated, and the class contains a method called SETPROPERTY, that method is called. The property name and the new value are passed to the SETPROPERTY method as parameters.

PROPAGATE

Any methods called, which have the PROPAGATE attribute set, cause a cascading execution effect down through the class hierarchy. After the method is executed, a search is made in all of the sub-objects (if any) that are defined as properties within the current object. If a method with the same name is found and that method has the NOTIFY attribute specified, then that method is called.

The CONSTRUCTOR and the DESTRUCTOR methods always act as if sub-objects have the PROPAGATE and the NOTIFY attributes set. This means that any properties that have been specified as sub-objects operate correctly when an object is created with the NEW operator.

:

The colon (:) is used to terminate the property scoping command. All properties following will now have the visibility of the specified scoping command, until a new scoping command is specified or the ENDCLASS is reached.

Example

```
// Use of Scope Options
class myclass
  public:
    property cPUBLIC_PROPERTY

    method Show && Publicly available method
    //...
    return && Show

  private:
    property cPRIVATE_PROPERTY
  local:
    property cLOCAL_PROPERTY
  static:
    property cSTATIC_PROPERTY
endclass

oMyObj = New myclass()
display memory
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

CLEAR

Class

Screen Forms

Purpose

Clear the screen

Syntax

CLEAR

See Also

@...CLEAR, CLEAR ALL, CLEAR GETS, CLEAR MEMORY, CLEAR LOCKS, CLEAR SCREEN

Description

The CLEAR command clears the terminal screen, positions the cursor at the top left corner of the screen, and releases all pending GETS issued with the @...GET command.

Example

clear

Products

Recital Mirage Server, Recital Terminal Developer

CLEAR ALL

Class

Environment

Purpose

Reinitialize the Recital environment

Syntax

CLEAR ALL

See Also

CLEAR, CLEAR GETS, CLEAR MEMORY, CLEAR LOCKS, CLOSE ALL, UNLOCK

Description

The CLEAR ALL command closes all tables which are currently open, all associated index files and format files, releases all memory variables, releases all record and file locks, releases all pending GETS, and selects workarea 1. In effect, it does a software reset of the system.

Example

clear all

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

CLEAR AUTOMEM

Class

Memory Variables

Purpose

Initializes memory variables corresponding to the current record of the active table

Syntax

CLEAR AUTOMEM

See Also

REPLACE AUTOMEM, STORE AUTOMEM, USE...AUTOMEM

Description

The CLEAR AUTOMEM command re-initializes memory variables corresponding to the current record of the active table, setting them to empty. Such memory variables can be generated automatically using the STORE AUTOMEM and USE...AUTOMEM commands.

Data Type	Empty
Character	""
Numeric	0
Logical	.F.
Date	{ }
Memo	""

Example

```
set locktype to optimistic
use customer
store automem
@1,1 get m.name
@2,1 get m.address
@3,1 get m.state
read
if not change()
    replace customer.name with m.name,;
    customer.address with m.address,;
    customer.state with m.state
endif
clear automem
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

CLEAR FCACHE

Class

Performance and Optimization

Purpose

Close files that are logically closed but physically open

Syntax

CLEAR FCACHE

See Also

SET FCACHE, CLEAR PROGRAM

Description

The CLEAR FCACHE command physically closes any files that are logically closed but physically open due to the use of SET FCACHE ON. SET FCACHE ON speeds up program execution by leaving files physically open, thereby eliminating the involvement of the operating system when re-opening a file.

Normally this command should not be needed as files should be removed from the open file cache when an open request conflicts with the file's current status, for example, shared versus exclusive or update versus read-only.

Example

```
// Program to tidy up on exit of application  
clear fcache
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

CLEAR GETS

Class

Screen Forms

Purpose

Clear pending @...GET commands

Syntax

CLEAR GETS

See Also

@...GET, CLEAR, CLEAR MEMORY, CLEAR ALL, CLEAR LOCKS, READ

Description

The CLEAR GETS command releases all pending GETS issued with the @...GET command. This command is often used with SAVE GETS and RESTORE GETS to save a batch of GETS, clear them to free the screen for other input and then restore them.

Example

clear gets

Products

Recital Mirage Server, Recital Terminal Developer

CLEAR IOSTATS

Class

Performance and Optimization

Purpose

Reset all I/O statistics to zero

Syntax

CLEAR IOSTATS

See Also

SET ICACHE, SET DCACHE, SET FCACHE, IOSTATS()

Description

The CLEAR IOSTATS command is used to clear statistics that have been generated as a result of the IOSTATS() Input/Output monitoring function. The CLEAR IOSTATS command resets all of the I/O statistics to zero. Note that whenever a table is opened, the I/O statistics for that particular workarea are reset to zero.

Example

```
clear iostats
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

CLEAR KEYS

Class

Keyboard Events

Purpose

Reset function key assignments to default

Syntax

CLEAR KEYS

See Also

SET KEY...TO, SAVE KEYS, RESTORE KEYS, ON KEY

Description

The CLEAR KEYS command clears hot key assignments that have been established with the SET KEY...TO, ON KEY, or RESTORE KEYS commands. All hot keys are disabled and all keys will perform their normal settings. Hot key assignments that have been saved with the SAVE KEYS command may be reinstated with the RESTORE KEYS command.

Example

```
set key -1 to helpfile
save keys to m_hotkeys
clear keys
//....
restore keys from m->m_hotkeys
```

Products

Recital Mirage Server, Recital Terminal Developer

CLEAR LOCKS

Class

Manual Locking

Purpose

Release all file and record locks

Syntax

CLEAR LOCKS

See Also

CLEAR, CLEAR GETS, CLEAR ALL, CLEAR MEMORY, UNLOCK ALL, CLOSE ALL

Description

The CLEAR LOCKS command releases all file and record locks. The UNLOCK ALL command is synonymous with CLEAR LOCKS. The CLEAR LOCKS command is only necessary if you are using manual locking techniques because the Recital/4GL automatically performs record locking and unlocking.

Example

clear locks

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

CLEAR MEMORY

Class

Memory Variables

Purpose

Release all memory variables

Syntax

CLEAR MEMORY

See Also

CLEAR, CLEAR ALL, CLEAR LOCKS, CLEAR GETS, RELEASE

Description

The CLEAR MEMORY command releases all memory variables and frees the storage that they had occupied. This includes user-defined classes, arrays and the system parameters, `_para1` to `_para9`.

Example

clear memory

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

CLEAR MENUS

Class
Menus

Purpose
Release all pending MENUS

Syntax
CLEAR MENUS

See Also
@...CLEAR, CLEAR ALL, CLEAR GETS, CLEAR MEMORY, CLEAR LOCKS, CLEAR PROMPT, CLEAR SCREEN

Description
The CLEAR MENUS command clears all pending MENUS issued with the @...MENU command.

Example
clear menus

Products
Recital Mirage Server, Recital Terminal Developer

CLEAR POPUPS

Class

Menus

Purpose

Clears the screen of Xbase style pop-ups and releases them from memory

Syntax

CLEAR POPUPS

See Also

DEACTIVATE POPUP, RELEASE POPUPS, SHOW POPUP, SET COMPATIBLE

Description

The CLEAR POPUPS command clears the screen of Xbase style pop-up menus and releases them from memory. The active pop-up is deactivated in the process, and all ON SELECTION commands related to that pop-up are cleared. The command SET COMPATIBLE should be set ON when using Xbase style menus.

Example

```
// Tidy up
clear popups
close tables
return
```

Products

Recital Mirage Server, Recital Terminal Developer

CLEAR PROGRAM

Class

Applications

Purpose

Closes files that are logically closed but physically open

Syntax

CLEAR PROGRAM

See Also

SET FCACHE, CLEAR FCACHE

Description

The CLEAR PROGRAM closes files that are logically closed but physically open because of the SET FCACHE ON command. SET FCACHE ON speeds up program execution by leaving files physically open, thereby eliminating the involvement of the operating system when re-opening a file.

Normally this command should not be needed as files should be removed from the open file cache when an open request conflicts with the file's current status, for example shared versus exclusive or update versus read-only.

Example

```
// Program to tidy up on exit of an application  
clear program
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

CLEAR PROMPT

Class

Menus

Purpose

Release pending @...PROMPT menus

Syntax

CLEAR MENUS

See Also

@...CLEAR, CLEAR ALL, CLEAR GETS, CLEAR MEMORY, CLEAR LOCKS, CLEAR MENUS, CLEAR SCREEN

Description

The CLEAR PROMPT command clears all pending menus issued with the @...PROMPT command.

Example

clear prompt

Products

Recital Mirage Server, Recital Terminal Developer

CLEAR READ

Class

Screen Forms

Purpose

Exit the active READ

Syntax

CLEAR READ [ALL]

See Also

@...CLEAR, CLEAR, CLEAR ALL, CLEAR GETS, CLEAR LOCKS, CLEAR MEMORY, CLEAR MENUS, CLEAR SCREEN, READ

Description

The CLEAR READ command exits the active READ. If the active READ is nested, control is returned to the parent READ. If the ALL keyword is specified all READs are terminated.

Example

set compatible to foxpro

```
@2,2 say "First Name:"  
@4,2 say "Last Name:"  
@2,16 get firstname size 1,24 default ""  
@4,16 get lastname size 1,24 default ""  
on key label f3 clearexit()  
read
```

```
function clearexit  
clear read  
return
```

Products

Recital Mirage Server, Recital Terminal Developer

CLEAR SCREEN

Class

Screen Forms

Purpose

Clear the screen

Syntax

CLEAR SCREEN

See Also

@...CLEAR, CLEAR, CLEAR ALL, CLEAR GETS, CLEAR MEMORY, CLEAR LOCKS

Description

The CLEAR SCREEN command resembles the CLEAR command in that it clears the terminal screen and positions the cursor at the top left corner of the screen (0,0). CLEAR SCREEN differs from the CLEAR command in that it does not release all pending GETS issued with the @...GET command.

Example

clear screen

Products

Recital Mirage Server, Recital Terminal Developer

CLEAR TYPEAHEAD

Class

Keyboard Events

Purpose

Clear terminal typeahead buffer

Syntax

CLEAR TYPEAHEAD

See Also

SET TBUFSIZE

Description

The CLEAR TYPEAHEAD command clears the typeahead buffer of all keystrokes entered. The CLEAR TYPEAHEAD commands also resets the LASTKEY() and READKEY() functions to zero, unless SET COMPATIBLE is set to FOXBASE | FOXPRO | FOXPLUS.

Example

```
clear typeahead
dialog box "Press ^W to continue, ^G to abandon."
if lastkey()=ctrl("g")
    return
endif
```

Products

Recital Mirage Server, Recital Terminal Developer

CLEAR WINDOWS

Class

Screen Windows

Purpose

Clear windows from screen and memory

Syntax

CLEAR WINDOWS

See Also

ACTIVATE WINDOW, DEFINE WINDOW, DEACTIVATE WINDOW, HIDE WINDOW

Description

The CLEAR WINDOWS command erases all displayed windows from the screen and releases all window definitions from memory. A window is an area of the screen designated for output and input. Windows are defined with the DEFINE WINDOW command, and displayed to the screen with the ACTIVATE WINDOW or SHOW WINDOW commands. There is no limit to the number of defined windows.

The CLEAR WINDOWS command is a quick way to clear the screen and reclaim memory space for more windows. Once the CLEAR WINDOWS command is issued, the DEFINE WINDOW command must be used to establish new window definitions, and the ACTIVATE WINDOW or SHOW WINDOW commands must be used to display them.

The CLEAR WINDOWS command is synonymous with the RELEASE WINDOWS ALL command. If you wish to clear a window from the screen, but retain its definition in memory, use the DEACTIVATE WINDOW command. If you wish to clear a window from the screen, but keep it active, use the HIDE WINDOW command. If you wish to clear windows from the screen, and save the window definition and the current window contents to a file, use the SAVE WINDOW and RESTORE WINDOW commands.

Example

clear windows

Products

Recital Mirage Server, Recital Terminal Developer

CLOSE

Class

Table Basics

Purpose

Close a table and associated files in a specified workarea

Syntax

CLOSE <workarea | alias>

See Also

CLOSE DATABASES, CLOSE FORMAT, CLOSE INDEX, CLOSE PROCEDURE, CLOSE ALL

Description

The CLOSE command closes the table and its associated index files and format files in the specified <workarea | alias> workarea.

Example

```
select a
use patrons index names
select b
close patrons
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

CLOSE ALL

Class

Table Basics

Purpose

Close all open files

Syntax

CLOSE ALL

See Also

CLOSE, CLOSE ALL, CLOSE ALTERNATE, CLOSE INDEX, CLOSE FORMAT, CLOSE PROCEDURE

Description

The CLOSE ALL command closes all open files. If FCACHE is OFF, the CLOSE ALL command will physically close files in the operating system. If FCACHE is ON, the CLEAR FCACHE must be issued as well as the CLOSE ALL command in order to physically close files in the operating system. To close procedure libraries, use the CLOSE PROCEDURE command.

Example

```
select a
use patrons
select g
use diary
close all
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

CLOSE ALTERNATE

Class

Input/Output

Purpose

Close the alternate output file

Syntax

CLOSE ALTERNATE [TO PRINT]

See Also

SET ALTERNATE, SET CONSOLE, PRINT

Description

The CLOSE ALTERNATE command closes the active alternate output file then optionally prints the file on the system printer. If the TO PRINT option is specified, then the system specific print command contained in the symbol DB_PRINT is executed. Under OpenVMS, DB_PRINT is defined in the Recital/4GL file, login.com, and by default will queue the print job on SYS\$PRINT. Under UNIX and Linux, DB_PRINT is defined in the file profile.db, and by default will queue a print job using the 'lp' command.

Example

```
use patrons index names
set alternate to namelist
set console off
list for event = "BALLET"
close alternate to print
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

CLOSE DATABASES

Class

Databases

Purpose

Closes the currently open database

Syntax

CLOSE DATABASES [ALL]

See Also

ALTER INDEX, ALTER TABLE, BACKUP DATABASE, CLOSE TABLES, COMPILE DATABASE, CREATE DATABASE, CREATE TABLE, CREATE INDEX, CREATE VIEW, DISPLAY DATABASE, DISPLAY INDEXES, DISPLAY TABLES, DROP DATABASE, DROP INDEX, DROP TABLE, LIST DATABASE, LIST INDEXES, LIST TABLES, OPEN DATABASE, RESTORE DATABASE, USE, SET EXCLUSIVE, ADATABASES(), DBUSED(), GETENV()

Description

The CLOSE DATABASES command closes the currently open database and its tables. If no database is currently open, all tables and their associated files are closed.

Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directories are sub-directories of the Recital data directory. The environment variable / symbol DB_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality.

CLOSE DATABASES does not close gateway sessions. The SET GATEWAY TO or CLOSE ALL commands can be used for this purpose.

Example

```
VFP/SQL> OPEN DATABASE hr EXCLUSIVE  
VFP/SQL> SELECT staff_no, lastname from staff  
VFP/SQL> CLOSE DATABASES
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

CLOSE FORMAT

Class

Screen Forms

Purpose

Close screen format file in current workarea

Syntax

CLOSE FORMAT

See Also

CLOSE, CLOSE ALL, CLOSE ALTERNATE, CLOSE INDEX, CLOSE DATABASES, CLOSE PROCEDURE, SET FORMAT TO

Description

The CLOSE FORMAT command closes the format file in the currently selected workarea. Format files are opened using the SET FORMAT TO <format file> command.

Example

use patrons index names
set format to addform
append
close format

Products

Recital Mirage Server, Recital Terminal Developer

CLOSE INDEX

Class

Indexing

Purpose

Close index files in the current workarea

Syntax

CLOSE INDEX

See Also

CLOSE, CLOSE ALL, CLOSE ALTERNATE, CLOSE DATABASES, CLOSE FORMAT, CLOSE PROCEDURE, SET INDEX TO

Description

The CLOSE INDEX command closes index files in the currently selected workarea. All single indexes and non-production multiple indexes will be closed. If the active table has a production index, that is a multiple index with the same basename as the table, the production index will remain open. Production indexes can be detached from their associated tables using the USE <table> NODBX command.

Example

```
use patrons index events, dates, names
seek "BALLET"
list while event = "BALLET"
close index
list for event = "BALLET"
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

CLOSE PROCEDURE

Class

Applications

Purpose

Close the procedure library file

Syntax

CLOSE PROCEDURE [<filename> | (<expC>)]

See Also

CLOSE, CLOSE ALL, CLOSE ALTERNATE, CLOSE FORMAT, CLOSE DATABASES, SET PROCEDURE TO, DO

Description

The CLOSE PROCEDURE command closes the procedure library named <filename>, or if no library is specified, all currently open procedure libraries. The <filename> can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. The <filename> is assumed to have a '.prg' extension unless otherwise stated.

Only procedures and functions declared with the SET PROCEDURE TO command are released with the CLOSE PROCEDURE command. Procedures and functions declared in the main program are left active. Up to a maximum of ten procedure files can be opened at one time when the ADDITIVE keyword is used on the SET PROCEDURE command.

Example

```
set procedure to yourlib
do yourproc
do yourproc2
close procedure
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

CLOSE TABLES

Class

Databases

Purpose

Closes the currently open tables and their associated files in the active database

Syntax

CLOSE TABLES [ALL]

See Also

ALTER INDEX, ALTER TABLE, BACKUP DATABASE, CLOSE DATABASES, COMPILE DATABASE, CREATE DATABASE, CREATE TABLE, CREATE INDEX, CREATE VIEW, DISPLAY DATABASE, DISPLAY INDEXES, DISPLAY TABLES, DROP DATABASE, DROP INDEX, DROP TABLE, LIST DATABASE, LIST INDEXES, LIST TABLES, OPEN DATABASE, RESTORE DATABASE, USE, SET EXCLUSIVE, ADATABASES(), DBUSED(), GETENV()

Description

The CLOSE TABLES command closes the currently open tables and their associated files in the active database. The database itself remains open. If no database is currently open, all tables and their associated files are closed.

If the ALL keyword is included, all tables and their associated files will be closed.

Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directories are sub-directories of the Recital data directory. The environment variable / symbol DB_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality.

Example

```
Recital/SQL> set sql to vfp
VFP/SQL> OPEN DATABASE hr EXCLUSIVE
VFP/SQL> use staff
VFP/SQL> CLOSE TABLES
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

COMPILE

Class

Applications

Purpose

Compile one or more program files

Syntax

COMPILE <filename>| (<expC>)

See Also

DO, SET COMPILE, SET DEVELOPMENT, MODIFY COMMAND, SET PSHARE

Description

The COMPILE command translates the source code of the specified program files into object code, and creates a file containing that object code. The object files created by the COMPILE program have the same basename as the program file, but a file extension that ends with an “o”. In most cases, the execute-only object code in these files runs much faster than the program files. The gain in speed is dependent on the number and size of the DO WHILE, DO CASE and IF constructs within the program.

Unless the full filename is specified in the <filename> argument, the COMPILE program looks for a file in the current directory and path (see SET PATH) with a .prg extension. COMPILE accepts any file or combination of files that contain source code. You may specify an expression that returns a file name or group of file names, as COMPILE will use all files matching the specified file pattern. The <filename> can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename.

You cannot modify or de-compile an object file, you can only modify the source file. The MODIFY COMMAND, ED and VI commands can be used to modify the source file. If SET DEVELOPMENT is ON, and SET COMPILE is ON, the DO command compares the time and date stamp of a source file with the time and date stamp of its associated object file. If the object file is older than the source file, then DO will recompile the source file before executing it.

In Recital Terminal Developer environments, program files must be compiled before they can be accessed from a server or runtime license. Compiled programs can be run from the operating system prompt as follows:

```
dbrt <compiled program name>
```

Example

```
compile *.prg
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

COMPILE DATABASE

Class

Databases

Purpose

Compile stored procedure files in the specified database or databases

Syntax

COMPILE DATABASE <database name> | <skeleton>

See Also

ALTER INDEX, ALTER TABLE, BACKUP DATABASE, CLOSE DATABASES, CLOSE TABLES, CREATE DATABASE, CREATE TABLE, CREATE INDEX, CREATE VIEW, DISPLAY DATABASE, DISPLAY INDEXES, DISPLAY TABLES, DROP DATABASE, DROP INDEX, DROP TABLE, LIST DATABASE, LIST INDEXES, LIST TABLES, OPEN DATABASE, RESTORE DATABASE, USE, SET EXCLUSIVE, ADATABASES(), DBUSED(), GETENV()

Description

The COMPILE DATABASE command compiles all the stored procedure files or program source files in the specified database or databases. The name of the target database is specified in <database name>. Multiple databases can be specified using the skeleton and wild card characters. The database or databases need not be open when the COMPILE DATABASE command is issued.

Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directories are sub-directories of the Recital data directory. The environment variable / symbol DB_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality.

Example

```
> compile database southwind
```

```
Recital/SQL> set sql to vfp  
VFP/SQL> COMPILE DATABASE hr
```

```
Recital/SQL> COMPILE DATABASE Mirage_*;
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

CONTINUE

Class

Applications

Purpose

Resume locate search

Syntax

CONTINUE

See Also

LOCATE, FIND, SEEK, SET FILTER, FOUND(), SCAN

Description

The CONTINUE command searches for the next record which meets the criteria as specified in the LOCATE command. If the search was successful, the FOUND() function will return .T., otherwise .F.. For large searches, indexing the table and using SEEK or FIND is recommended.

Example

```
use patrons
locate for event = "OPERA"
do while found()
    display name, event
    continue
enddo
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

CONVERT

Class

Table Basics

Purpose

Convert tables to Recital 9 format

Syntax

CONVERT [VER90 | INDEX | DBF | FMT | FRX | MEM | PRG | TXT | FRM] <filename>| (<expC>)

See Also

Optimizing Indexes using SYNCNUM, Upgrading from Pre-9.0 Versions, USE

Description

The CONVERT command, like the DBCONVERT utility in Recital Terminal Developer, can be used to convert Recital version 8.x tables to the Recital 9.0 format, to populate tables' SYNCNUM values or to convert files from other Xbase formats. If no conversion type is specified, the default is VER90, which converts Recital version 8.x tables to the Recital 9.0 format.

VER90

Database tables (.dbf files) used by the Recital 9.0 and later product lines use a different file structure to previous Recital versions. Therefore, before you can use your existing Recital data tables, they must be converted to the new file structure. The CONVERT command can be used to convert a single specified table or multiple tables. Wildcard characters can be used in the file specification.

Recital Corporation strongly recommends that you perform a full backup of your Recital applications upgrading and converting your tables. Production tag index files are recreated by the DBCONVERT/CONVERT process, but single index files (.ndx) will need to be rebuilt manually.

INDEX

The INDEX convert option processes the specified tables or all Recital 9 tables in the current directory if no filenames are specified. It updates all the SYNCNUM values starting with 1 in the first row and adding one to the value for each subsequent row. Any previous values stored in the rows are discarded.

This option will also locate any .dbx files associated with the table or tables and convert them to use SYNCNUM at the end of each index expression to optimize the index by making all the keys unique. Character indexes have SYNCNUM added to the end of the expression. Date indexes are converted to DTOS() and have the SYNCNUM added to the end. Numeric indexes are not affected.

The SYS(14) and INDEXKEY() functions will not return the SYNCNUM if it is on the index, however DISPLAY/LIST INDEX and DISPLAY/LIST STATUS will.

Please see DB_INDEXSEQNO for optimizing .ndx files.

DBF | FMT | FRX | MEM | PRG | TXT | FRM

The other file formats are as follows:

File Type	Description
DBF	Tables and Memos
FMT	Screen Format files
FRX	FoxPro Report Format files
MEM	Memory files
PRG	Program files
TXT	Text files
FRM	Report Format files

Example

```
convert ver90 *.dbf
```

```
convert ver90 customer.rdb
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

COPY DICTIONARY TO

Class

Table basics

Purpose

Copy dictionary attributes to a table file

Syntax

COPY DICTIONARY TO <.xad file>

[ADDITIVE]

See Also

COPY STRUCTURE EXTENDED, COPY, CREATE FROM, CREATE DICTIONARY

Description

The copy dictionary command is used to import attributes in an Application Data Dictionary (ADD) into a table file. This is useful for moving dictionary files between non binary-compatible hardware platforms. A new dictionary may be created by using the CREATE DICTIONARY command.

The .xad table is created with the following structure:

Field	Type	Length	Description
XAD_TYPE	Character	3	File Type
XAD_FILE	Character	255	File Name
XAD_ID	Character	32	Object ID
XAD_OBJECT	Character	10	Object Type
XAD_SEQNO	Character	3	Object Value Sequence #
XAD_VALUE	Character	255	Object Value

The records of the table define the ADD attributes from the source table. These may be modified using any of the standard 4GL record editing commands (EDIT, REPLACE etc) if required.

ADDITIVE

If the <.xad file> already exists, the ADDITIVE keyword can be used to append records. If the ADDITIVE keyword is not specified, any records in the existing <.xad file> will be overwritten.

Example

use accounts

copy dictionary to accounts

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

COPY FILE

Class

Disk and File Utilities

Purpose

Copy any type of file

Syntax

COPY FILE <filename1> | (<expC1>) | &<memvar> TO <filename2> | (<expC2>) | &<memvar>

See Also

COPY, COPY STRUCTURE, COPY STRUCTURE EXTENDED, DECRYPT, ENCRYPT, RUN

Description

The COPY FILE command copies a file of any type <filename1> to <filename2>. The file extensions for both files must be specified. The file names can be substituted with any character expression, enclosed in round brackets or memory variable preceded by an ampersand (&), which returns a valid filename. If <filename2> already exists, it will be overwritten.

The COPY FILE command copies only the specified file, it does not copy associated files. For example, if COPY FILE is used to copy a database table (.dbf), then only the .dbf file will be copied, not the associated multiple index file (.dbx), memo file (.dbt), dictionary file (.dbd), DES3 encryption key file (.dkf). When using COPY FILE to copy an encrypted database table, the .dkf file must also be copied or the target table will not be accessible.

Example

```
close tables
copy file patrons.dbf to backup.dbf
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

COPY INDEXES

Class

Indexing

Purpose

Copy single indexes to a multiple index file

Syntax

COPY INDEXES <.ndx filelist>

[TO <.dbx filename>]

See Also

INDEX ON, CREATE STRUCTURE, DELETE TAG, COPY STRUCTURE TO, DELETE TAG, MODIFY STRUCTURE, SET INDEX TO, USE, TAG(), TAGCOUNT(), TAGNO()

Description

The COPY INDEXES command creates a multiple index by copying single index files to one file. This command is useful for converting .ndx files to .dbx files. This command requires the indexes to be active before the copy can be performed. Use SET INDEX TO <.ndx file> to open a single index file. By default the index tag or tags are created in the production index. If no production index exists, then a new one is created.

TO <.dbx filename>

If the TO <.dbx filename> clause is used, the tag or tags are created in the specified .dbx file. If the .dbx file does not exist, it will be created.

Example

use accounts

set index to ordno, invno

copy indexes ordno, invno

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

COPY MEMO

Class

Memos

Purpose

Copy a memo field into a file

Syntax

COPY MEMO <memo fieldname> TO <filename>
[ADDITIVE]

See Also

APPEND MEMO, MEMOREAD(), MEMOWRITE(), SET MEMOFORMAT

Description

The COPY MEMO command copies the contents of a single memo field into a file. The <memo fieldname> of the current record in the active table is copied to the file specified by <filename>. If no file extension is specified, '.txt' is assumed.

ADDITIVE

The optional ADDITIVE keyword causes the memo to be appended to the end of the text file. Without the ADDITIVE keyword, any existing text will be overwritten.

Example

```
seek "JimL"  
do while emp_code = "JimL"  
    copy memo notes to comments additive  
    skip  
enddo
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

COPY STRUCTURE

Class

Table basics

Purpose

Copy the structure of the active table to another table

Syntax

```
COPY STRUCTURE TO <filename> | (<expC>)  
[FIELDS <field list>]  
[[WITH] CDX | [WITH] PRODUCTION]]
```

See Also

COPY, COPY FILE, COPY STRUCTURE EXTENDED, MODIFY STRUCTURE, CREATE FROM

Description

The COPY STRUCTURE command creates a new empty table file with the same structure as the active table. The dictionary is also copied if one exists. The TO <filename> can be substituted with any <expC>, enclosed in round brackets, which returns a valid filename. This file will be created or, if it already exists, will be overwritten. If no file extension is specified, then the target file will have an extension of '.dbf'.

If the active table is encrypted, the encryption information will be copied to the target table. A .dkf encryption key file will be created for the target table and the correct three-part DES3 encryption key must be specified before the table can be accessed. A different key can be specified for the new table by including the new encryption key in the <filename>. The three part comma-separated key should be enclosed in angled brackets and appended to the filename, e.g. mytable<key_1,key_2,key_3>. This syntax can also be used to encrypt the new table when the source table itself is not encrypted.

FIELDS <field list>

If the optional FIELDS clause is specified, then the operation is restricted to those fields named in the comma separated <field list>. Dictionary entries are only copied for the specified fields.

WITH CDX | WITH PRODUCTION

The [[WITH] CDX | [WITH] PRODUCTION]] clause causes tag expressions in the currently active multiple index file to be copied to an empty .cdx/.dbx file with the same basename as the TO <filename>.

Example

```
use patrons  
copy structure to namelist;  
    fields name, street, city, state
```

```
// make an encrypted copy of the structure  
use patrons  
copy structure to enc_patrons<key_1,key_2,key_3>
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

COPY STRUCTURE EXTENDED

Class

Table basics

Purpose

Create a table with records containing field definitions

Syntax

COPY STRUCTURE EXTENDED TO <.dbf filename> | (<expC>)
[FIELDS <field list>]

See Also

COPY, COPY FILE, COPY STRUCTURE, CREATE FROM, MODIFY STRUCTURE, REPLACE

Description

The COPY STRUCTURE EXTENDED command operates on the currently active table and creates a new table with records containing the active table's structure. The new table has five fields:

Field	Type	Length	Description
FIELD_NAME	Character	32	Field name
FIELD_TYPE	Character	1	Data type
FIELD_LEN	Numeric	3	Width of field
FIELD_DEC	Numeric	3	Number of decimal places
FIELD_DES	Character	25	Field description

A record is created for each field from the active table, so that each record contains a complete field definition. Once a table has been created in this manner, it can be used to build a new table structure. The records containing the field definitions can be modified in the same way as any standard records (EDIT, REPLACE etc). The CREATE FROM command can then use this structure table as the source for a new table.

TO <.dbf filename>

The TO <.dbf filename> can be substituted with any <expC>, enclosed in round brackets, which returns a valid filename.

FIELDS <field list>

If the optional FIELDS clause is specified, then the operation is restricted to those fields named in the comma separated <field list>.

Example

```
use patrons
copy structure extended to patstru
select workarea()
create newpatrons from patstru
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

COPY TAG

Class

Indexing

Purpose

Create a single index file from an index tag

Syntax

COPY TAG <tagname> [OF <.dbx filename>] TO <.ndx filename>

See Also

INDEX ON, CREATE STRUCTURE, COPY INDEXES, COPY STRUCTURE TO, DELETE TAG, MODIFY STRUCTURE, SET INDEX TO, USE, MDX(), TAG(), TAGCOUNT(), TAGNO()

Description

The COPY TAG command creates a single index (.ndx) file from a tag in a multiple index (.dbx) file, copying the specified <tagname> to a single, stand-alone index file specified with <.ndx filename>. If no file extension is specified, then '.ndx' is used.

OF <.dbx filename>

If the tag does not exist in the currently open multiple index file the OF <.dbx filename> clause must be used to specify the <tagname> source.

Example

*Open up dbf with a dbx file
use customer
copy tag zip to zip.ndx

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

COPY TO

Class

Disk and File Utilities

Purpose

Copy all or part of the active table to another table or file

Syntax

```
COPY TO <filename> | (<expC1>)  
[<scope>]  
[DECRYPT | ENCRYPT <expC2>]  
[FIELDS <field list>]  
[FOR <condition>]  
[WHILE <condition>]  
[[WITH] CDX | [WITH] PRODUCTION]]  
[[TYPE] FIXED | SDF | XML | DELIMITED | DELIMITED WITH BLANK | DELIMITED WITH  
<delimiter> | DELIMITED WITH TAB]
```

See Also

APPEND FROM, COPY FILE, COPY STRUCTURE, COPY STRUCTURE EXTENDED, CREATE BRIDGE, DECRYPT, ENCRYPT, JOIN, SET FILTER

Description

The COPY command copies records from the active table to another table or file. The file name can be substituted with a <expC1> enclosed in round brackets that returns a valid filename. If there is a table dictionary or a memo file on the current table, these files will also be copied to the new Recital table. If the current table is indexed, records will be copied in the indexed order, but the index file itself is not copied to the new table. Format files associated with the current table will not be copied to the new table. If SET FILTER TO is in effect, then only records that satisfy the filter condition are copied. If SET DELETED is ON, then records marked for deletion will not be copied.

To <filename>

The TO file will be created or, if it already exists, will be overwritten. If no file extension is specified for the TO <filename>, tables will default to '.dbf', XML files will default to '.xml', and all other files will default to '.txt'. The <filename> can include an encryption key for encrypted database tables. The three part comma-separated key should be enclosed in angled brackets and appended to the filename, e.g. mytable<key_1,key_2,key_3>. This allows the creation of an encrypted copy of a non-encrypted table or an encrypted copy with a different key to the encrypted source table.

<scope>

If no <scope> is specified, then the default is ALL.

DECRYPT | ENCRYPT <expC2>

The DECRYPT and ENCRYPT clauses can be used to specify whether the target table of a COPY TO operation is encrypted or not. Specifying DECRYPT allows the creation of a non-encrypted copy of an encrypted table.

The ENCRYPT <expC2> clause encrypts the target table using the three part key specified in <expC2>. The <expC2> must contain a three part comma-separated key, each part a maximum of 8 characters, e.g. "key_1,key_2,key_3". Angled brackets may optionally enclose the key, e.g. "<key_1,key_2,key_3>". A .dkf file is created with the same basename as the target table. This allows the creation of an encrypted copy of a non-encrypted table or an encrypted copy with a different key to the encrypted source table. By default, when copying an encrypted table, if the key is not included in the <filename> and neither clause is specified, the target table is encrypted and has the same encryption key as the source table. A .dkf file is created with the same basename as the target table. If the source table is not encrypted and neither clause is specified, the target table will not be encrypted.

FIELDS <field list>

If the FIELDS clause is specified, then only those fields specified will be copied, otherwise all fields will be copied. The <field list> is a comma-separated list of field names. The fields specified can contain alias pointers, allowing copy to retrieve fields from multiple data files to be copied to a single data file. The number of records copied when using alias pointers is determined by the number of records in the table in the workarea from which the copy was initiated.

FOR <condition>

If the FOR clause is specified, then only those records which satisfy the specified <condition> are copied. The record pointer will always be positioned at EOF at the end of the operation if SET COMPATIBLE is in effect.

WHILE <condition>

If the WHILE option is used, the <scope> defaults to REST. The WHILE clause will copy records so long as the <condition> is true (.T.), and is used to restrict the range of records processed. When used in conjunction with the SEEK or LOCATE commands, it allows a quick way of copying selected records. The record pointer will always be positioned at EOF at the end of the operation if SET COMPATIBLE is in effect.

WITH CDX | WITH PRODUCTION

The [[WITH] CDX | [WITH] PRODUCTION]] clause causes the currently active multiple index file to be copied along with the table to a .cdx/.dbx file with the same basename as the TO <filename> when the target is a database table.

TYPE FIXED

If the target file type specified is FIXED, then the file will be created containing fixed length records without any record terminating character. This file type is useful for exporting records into a file that can be read by PASCAL, C, FORTRAN, etc.

TYPE SDF

If the target file type specified is SDF then the file will be created containing records as lines of text terminated with a carriage return/linefeed sequence. On UNIX the carriage return is not present. On OpenVMS the records are stored as variable length text records. If any of the fields being copied are binary fields, then the records are created in FIXED format. The maximum length of the text line used with COPY...SDF is 8192 characters.

TYPE XML

The XML clause copies the records to an Extensible Markup Language (XML) file. It also creates a matching Document Type Definition file with a '.dtd' file extension if the XML format is set to RECITAL. The default XML file format is Microsoft® ActiveX® Data Objects (ADO). This default can be set with the command SET XMLFORMAT TO <RECITAL | ADO>.

TYPE DELIMITED

If the DELIMITED clause is specified, then the target file will be created as a text file. Each field will be separated by a comma (,), and character fields will be enclosed in double quotes. If DELIMITED WITH BLANK is specified, then fields will be separated by a single space character instead of a comma. Character fields will not be enclosed. If DELIMITED WITH <delimiter> is specified, then the double quotes used to enclose character fields will be substituted with the <delimiter> and fields will be comma separated. If DELIMITED WITH TAB is specified, then fields will be separated by a tab character instead of a comma. Character fields will not be enclosed. Files created with the COPY TO...DELIMITED command can be appended into other Recital tables using the APPEND FROM...DELIMITED command.

Example

```
use patrons index names
copy to ballet for event = "BALLET"
seek "OPERA"
copy to opera rest;
    while event = "OPERA";
        for date = date()
// Another example
use payroll
// Copy to a file with today's name
copy to (cdow(date())) for amount > 100
// Make an encrypted copy
copy to encver<key_1,key_2,key_3>
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

COPY TO ARRAY

Class

Fields and Records

Purpose

Copy current table to an array

Syntax

```
COPY TO ARRAY <array>
[FIELDS <field list>]
[FOR <condition>]
[WHILE <condition>]
```

See Also

APPEND FROM ARRAY, DECLARE, DIMENSION, GATHER, PRIVATE, PUBLIC, RELEASE, RESTORE FROM, SAVE TO, SCATTER, AVERAGE(), ACHOICE(), ACOPY(), ADEL(), ADIR(), AFIELDS(), AFILL(), AINS(), ALLEN(), AMAX(), AMIN(), ASCAN(), ASORT(), ASUM(), AVERAGE()

Description

The COPY TO ARRAY command allows all or part of the active table to be copied to the previously declared two-dimensional specified array, <array>. The target array must have elements defined that match the field list from the table.

NOTE: If SET COMPATIBLE is set to FOXPRO | FOXPLUS | FOXBASE the array need not be pre-defined, it will be created automatically.

FIELDS <field list>

The optional FIELDS clause restricts the operation to those fields listed in <field list>. The <field list> contains a comma-separated list of field names.

FOR <condition>

The optional FOR clause restricts the operation to those records that match the specified <condition>.

WHILE <condition>

The WHILE clause will copy records so long as the <condition> is true (.T.), and is used to restrict the range of records processed. When used in conjunction with the SEEK or LOCATE commands, it allows a quick way of copying selected records. When the WHILE clause is used, the <scope> will default to REST.

Example

```
declare orders[10000,10]
use suppliers
copy to array orders for ord_date < date()
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

COUNT

Class

Fields and Records

Purpose

Count records in the active table that satisfy a specified condition

Syntax

```
COUNT [<scope>]  
[FOR <condition>]  
[TO <memvar>]  
[WHILE <condition>]
```

See Also

SUM, AVERAGE, TOTAL, ACC(), CALC()

Description

The COUNT command counts the number of records in the active table. If no <scope> is specified, ALL records will be counted. If SET FILTER TO is in effect, then only those records that satisfy the filter <condition> are counted. If SET DELETED is ON, then records marked for deletion will not be included in the COUNT.

FOR <condition>

If the for <condition> is used then only the records that satisfy the specified <condition> are counted.

TO <memvar>

Using the TO <memvar> clause causes the result of the COUNT operation to be stored in the specified memory variable.

WHILE <condition>

The WHILE clause can be used in conjunction with the SEEK or LOCATE commands to restrict the range of records counted. When the WHILE option is used the <scope> defaults to REST.

Example

```
use patrons  
count to nResult for event = "CHOPIN" and;  
    date = date()
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

CREATE

Class

Terminal Developer Development Tools

Purpose

Define the structure of a new database table through a form

Syntax

CREATE <.dbf file> | (<expC>)

See Also

ALTER INDEX, ALTER TABLE, APPEND FROM, BACKUP DATABASE, CLOSE DATABASES, CLOSE TABLES, COMPILE DATABASE, COPY TO, COPY STRUCTURE EXTENDED, CREATE DATABASE, CREATE FROM, CREATE TABLE, CREATE INDEX, CREATE VIEW, DISPLAY DATABASE, DISPLAY INDEXES, DISPLAY TABLES, DROP DATABASE, DROP INDEX, DROP TABLE, INDEX, LIST DATABASE, LIST INDEXES, LIST TABLES, MODIFY STRUCTURE, OPEN DATABASE, PACK, REBUILD DATABASE, RESTORE DATABASE, USE, SET AUTOCATALOG, SET EXCLUSIVE, ADATABASES(), DBUSED(), GETENV(), DB_MAXWKA

Description

The CREATE command provides a full-screen forms-based facility used to define a new database table structure. The filename can be substituted with an <expC> enclosed in round brackets that returns a valid filename. If no file extension is specified, '.dbf' will be used. Database table filenames should not begin with numeric characters.

The cursor keys can be used to move around the form in order to fill in the fields. Each row of the form corresponds to a field in the table structure being created. A maximum of 256 fields may be created.

Each column in the CREATE form represents information needed to define a field. Required information includes the field name, the field data type and the field width. Field descriptions are optional, but recommended.

The field name can be a maximum of 32 characters long, beginning with a letter or underscore, followed by any combination of letters (A-Z, case insensitive), digits (0-9) and underscores (_).

As the cursor moves on to the data type column, the value "Character" is filled in automatically. Typing the first letter of any of the data types listed below will complete the full value. Pressing the [HELP] key displays a pop-up choice list of available data types. The spacebar can also be used to cycle through the supported data types.

The field data type can be any one of the following:

Initial	Type	Description
B	Byte	1 byte integer
C	Character	Character
D	Date	Date
F	Float	8 byte floating point
I	Integer	4 byte integer
L	Logical	Logical
M	Memo	Memo
N	Numeric	Numeric
O	Object	Binary large object
P	Packed	Packed Decimal (OpenVMS only)
Q	Quad	Quadword (OpenVMS only)
R	Real	4 byte floating point
S	Short	2 byte integer
T	DateTime	Date and Time
V	VAXdate	VMS date (OpenVMS only)
Y	Currency	Currency
Z	Zoned	Dibol zoned numeric

The maximum field width for each data type is:

Initial	Display Width	Physical Storage
B	3 (fixed, cannot be input)	1 byte
C	255	255 bytes
D	8 (fixed, cannot be input)	4 bytes
F	25*	8 bytes
I	25*	4 bytes
L	1 (fixed, cannot be input)	1 byte
M	Unlimited (disk & OS permitting)	Variable length
N	25*	25 bytes
O	Unlimited (disk & OS permitting)	Variable length
P	25*	13 bytes
Q	25*	8 bytes
R	25*	4 bytes
S	25*	2 bytes
T	8 (fixed, cannot be input)	8 bytes
V	8 (fixed, cannot be input)	8 bytes
Y	8,4 (fixed, cannot be input)	8 bytes
Z	20*	20 bytes

*Including decimal point and up to nine decimal places.

Field types 'F', 'I', 'P', 'Q', 'R' and 'S' are stored as binary, and the field width specified is the output display width. Fields with fixed length data types have the width filled in automatically. The number of decimal places can only be added when the data type requires it.

The field description can be a maximum of 25 characters long. These field descriptions can be toggled on and off from within the default forms for EDIT, CHANGE, APPEND, QUERY and INSERT. The MENU FIELDS and MENU QUERY commands also display the field descriptions in the FIELDS MENU window. The use of meaningful descriptions is highly recommended.

To create an index tag from a field, place a “Y” in the Index column. Index tags may also be created with the INDEX ON ... TAG command. Index tags created in this way form what is called the ‘production’ index file. The production index file has the same base name as the table and a ‘.dbx’ file extension. The production index is opened automatically whenever the table is opened. No master index tag order is set, but all tags will be updated and can be set to be the master index order. To remove an index tag, replace the “Y” in the Index column with an “N” or use the DELETE TAG command.

It is good practice not to overload one table with redundant fields. The SET RELATION command can be used to join tables together. If the After Image Journal facility is being used, then seven of the maximum number of fields must be left unused for the journal file.

The CREATE work surface provides a menu bar which is activated by pressing the [MENUBAR] key. When the command SET MCONFIRM is OFF, the CREATE menu bar operates as pulldown menus.

The menu options <DICTIONARY>, <TRIGGERS>, <SECURITY> and <PROTECTION> all store their information in the Applications Data Dictionary (.dbd file). The Applications Data Dictionary is called by any commands that need to read, update, view or modify table data. This is very powerful feature allowing the maintenance of referential integrity, help prompts, pop-up choice lists and security in one central repository for each table.

DICTIONARY

You can specify any of the following field ‘attributes’ by selecting the <DICTIONARY> menu bar item.

Attribute	Description
Picture	An editing picture for data input and output. See @...GET...PICTURE for more information.
Validation	A logical expression for field validation. You can perform cross-table lookups with RLOOKUP(). See @ GET...VALID for more information. This can also be used to bypass default field objects, such as the calculator on numeric fields or the calendar on date fields. See @...GET...VALIDATE WITH for more information.
Error	An error message to display in an ALERT box if validation fails. See @...GET...ERROR for more information.
Choices	<p>A pop-up choice list to be displayed when the [HELP] key is pressed on a character field.</p> <p>Examples Miss,Mr,Ms,Mrs @suppliers, name+code</p> <p>The Choices attribute also allows you to customize a pop-up choice list by specifying a User Defined Function (UDF). As an alternative to the dynamic @<alias>,<expC> choice list, a UDF could use record selection commands, such as MENU BROWSE, to display the data from records that match particular selection criteria rather than from the entire table. A UDF can also display the choice list anywhere on the screen, whereas the static and dynamic options will always display in the center. UDFs may be entered in the <Choices> attribute by prefixing the UDF name with a question mark. See the FUNCTION command for more information about User Defined Functions.</p> <p>Example ?helpproc(“NAME”,”names”,10,10,20,70)</p>
Range	A validity range for numerics or dates. The format is minimum, maximum. Specify dates with the { } notation. See @...GET...RANGE for more information

Attribute	Description
	Examples 10,100 (01/01/2000},{01/01/2001}
Help	A help message displayed in the message line when the field is selected. See @...GET...HELP for more information.
Calculated	The field is calculated by this expression, which can contain references to other tables. See @...GET...CALCULATED BY for more information. Example ord_value - paid_value
Recalculate	A 'trigger', causing all calculated fields to be recalculated and redisplayed when this field is modified in a form. See @...GET...CALCULATE for more information.
Must enter	Check that data has been entered in this field when user moves off the field in a form. See @...GET...MUST_ENTER for more information.
Default	An expression specifying the 'default' value when APPEND [BLANK] is executed. Example Date()

You can move around the database structure in the <DICTIONARY> menu using the [NEXT PAGE] and [PREV PAGE] keys. If you exit the <DICTIONARY> menu with the [EXIT/SAVE] key, changes are saved to the Dictionary (.dbd file). You can override this by exiting the CREATE | MODIFY STRUCTURE work surface with the [ABANDON] key. Pressing the [ABANDON] key in the <DICTIONARY> menu discards any changes made to the dictionary.

TRIGGERS

The <TRIGGERS> menu bar option allows you to specify table level triggers. Triggers are event-driven procedures called before an I/O operation. You may edit a trigger procedure from within the <TRIGGERS> menu by placing the cursor next to the procedure name and pressing the [HELP] key. A text window pops up for editing. If the table triggers are stored in separate <.prg> files, rather than in a procedure library, procedures need not be pre-defined (SET PROCEDURE) before using the table. The following triggers can be selected and associated with a specified procedure name in the <TRIGGERS> menu.

Trigger	Description
UPDATE	The specified procedure is called prior to an update operation on the table. If the procedure returns .F., then the UPDATE is canceled.
DELETE	The specified procedure is called prior to a delete operation on the table. If the procedure returns .F., then the DELETE is canceled.
APPEND	The specified procedure is called prior to an append operation on the table. If the procedure returns .F., then the APPEND is canceled.
OPEN	The specified procedure is called after an open operation on the table.
CLOSE	The specified procedure is called prior to a close operation on the table.
ROLLBACK	The specified procedure is called when a user presses the [ABANDON] key in a forms based operation.

SECURITY

The <SECURITY> menu bar option pulls down a menu of table operations for which Access Control Strings can be specified. An Access Control String (ACS) is a range of valid user identification codes, and is used to restrict table operations to certain individuals or groups. Each user on the system is allocated a group number and a user number. The user identification code is the combination of group and user numbers. When constructing an Access Control String of linked user identification codes, wildcard characters may be used.

Example ACS	Description
[1,2]	In group 1, user 2
[100,*]	In group 100, all users
[2-7,*]	In groups 2-7, all users
[*],100-200]	In all groups, users 100-200
[1,*]&[2-7,1-7]	In group 1, all users, in groups 2-7, users 1-7

Please note that the maximum ACS length is 254 characters. OpenVMS group and user numbers are stored and specified in octal. On other Operating Systems, group and user numbers are stored and specified in decimal.

Access Control Strings may be associated with the following operations:

Operation	Description
READONLY	Users specified in the ACS have read-only access to the table. All other users have update access.
UPDATE	Users specified in the ACS have update access to the table. All other users are restricted to read-only access.
APPEND	Users specified in the ACS can append records into the table. No users can append.
DELETE	Users specified in the ACS can delete records from the table. No users can delete.
COPY	Users specified in the ACS can copy records from the table. No users can copy.
ADMIN	Users specified in the ACS can use the following commands: SET DICTIONARY TO MODIFY STRUCTURE PACK ZAP REINDEX All other users cannot, except the creator of the table, who is always granted ADMIN access.

PROTECTION

Field level security can be defined through the <PROTECTION> menu bar option. The menu will show protection relating to the currently selected field. The [NEXT PAGE] and [PREV PAGE] keys can be used to move between the fields in the table without exiting the menu. The format of the ACS is the same as in <SECURITY> above. The following protection can be defined:

Operation	Description
READONLY	Users specified in the ACS have read-only access to the field. All other users have update access.
UPDATE	Users specified in the ACS have update access to the field. All other users are restricted to read-only access.
HIDDEN	Users specified in the ACS see the 'hiddenfield' character rather than the data in the field. All other users see the data.

Hidden fields can be accessed and viewed on a work surface, but the field contains the hiddenfield character, '?'. If the field is referenced in an expression, it will contain the following: blanks for character fields, 'F' for logical fields, 00/00/0000 for date fields and blank for memo fields.

LOAD

The <LOAD> menu bar option displays a popup dialog box prompting for an existing table name. The structure from the existing table can be loaded into the current CREATE worksurface. If fields have already been defined in the worksurface, these will be overwritten when the structure is loaded.

If SET CLIPPER is ON, the CREATE command can be used in programs to create empty structure tables. Structure tables can hold information about the structure of another table and can be used to recreate the table using the CREATE FROM command. The COPY STRUCTURE EXTENDED command can also be used to create a complete structure table including the records that define a table structure.

If a database is open when a new table is saved via the CREATE work surface, a dialog will pop up prompting for a file description. This file description information is stored in the table's record in the database catalog. Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directories are sub-directories of the Recital data directory. The environment variable / symbol DB_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality. Databases are opened using the OPEN DATABASE command.

Example

create patrons

Products

Recital Terminal Developer

CREATE BRIDGE

Class

Data Connectivity

Purpose

Create a bridge definition for connection to RMS or C-ISAM data files

Syntax

```
CREATE BRIDGE <.brg filename> | (<expC>)  
[FROM <.ini filename>]
```

See Also

CREATE, CREATE BRIDGE (SQL)

Description

The CREATE BRIDGE command is used to create a bridge file to an external file. Recital clients can access Informix compatible C-ISAM files and the following fixed length RMS File types: RMS Indexed Sequential, RMS Relative, RMS Sequential. Data access is achieved through a bridge. This requires the creation of a bridge file and an empty Recital table that has a structure matching that of the external file. The empty Recital table can be created using the SQL CREATE command or the Recital Terminal Developer CREATE Development Tool. By convention, the empty structure file is given the file extension '.str' rather than the default '.dbf'.

The CREATE BRIDGE worksurface provides a full screen facility for bridge creation in Recital Terminal Developer. The <.brg filename> can be substituted with any character expression, enclosed in round brackets, that returns a valid filename. If no file extension is specified, then .brg is used. The following elements can be defined for the bridge:

Element	Description	Maximum Width (Characters)
Bridge Type	External data file type: CISAM, RMSIDX, RMSREL or RMSSEQ.	10
External Name	Name of the external data file.	80
Database Name	Name of the Recital structure table.	80
Alias	The name to use to access the file.	10

For RMS files you can also specify the indexes to use.

Element	Description	Maximum Width (Characters)
Index keys 1-7	Indexes to use with the bridge (RMS only).	50

For RMS Indexed Sequential files specify the existing RMS index keys. For RMS Relative and Sequential file structures you can build Recital single indexes and associate them with the bridge file. In these cases, specify the full index filenames.

In other Recital products, the FROM <.ini filename> clause can be used to create the bridge file without the need to access the CREATE BRIDGE worksurface. Firstly, an 'ini' file should be created on the server in the data directory where the external data file is held. The ini file has the following contents:

```
[bridge]
bridgetype=<bridgetype>
externalname=<name of the external data file>
databasename=<name of the Recital structure table>
alias=<the name to use to access your file>
indexkey1=<optional RMS index key or index filename>
indexkey2=<optional RMS index key or index filename>
indexkey3=<optional RMS index key or index filename>
indexkey4=<optional RMS index key or index filename>
indexkey5=<optional RMS index key or index filename>
indexkey6=<optional RMS index key or index filename>
indexkey7=<optional RMS index key or index filename>
```

e.g. cisamdemo.ini

```
[bridge]
bridgetype=CISAM
externalname=cisam.dat
databasename=cisamstru.str
alias=cisamdemo
```

e.g. rmsreldemo.ini

```
[bridge]
bridgetype=RMSREL
externalname=rmsrel.dat
databasename=rmsreldemo.str
alias= rmsreldemo
indexkey1=ind1.ndx
indexkey2=ind2.ndx
```

e.g. rmsidxdemo.ini

```
[bridge]
bridgetype=RMSIDX
externalname=rmsidx.dat
databasename=rmsdemo.str
alias=rmsdemo
indexkey1=acc_prefix+acc_no
indexkey2=acc_prefix
```

NOTE: There should be no white space either side of the '=' signs.

The SQL CREATE BRIDGE command can also be used to create bridge files from all Recital products. Please see the Recital/SQL documentation for full details.

Bridge files are often given a '.dbf' file extension, instead of the default '.brg' to allow them to appear in table listings with standard Recital tables.

Bridges can also be used to call program files or view files:

Bridge Type	External Name
APP	Program file to run when the bridge is used
VIEW	View file to open when bridge is used

Example

create bridge cisamdemo.dbf from cisamdemo

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

CREATE DICTIONARY FROM

Class

Table basics

Purpose

Create a new applications data dictionary from an xad file

Syntax

CREATE DICTIONARY FROM <.xad file>

See Also

COPY DICTIONARY, COPY TO, COPY STRUCTURE EXTENDED, CREATE FROM, APPEND FROM

Description

The CREATE DICTIONARY command is used to create a new dictionary file from an xad file. The <.xad file> is created with the COPY DICTIONARY command. This command is useful for moving dictionary files between non binary-compatible hardware platforms.

Example

use accounts
create dictionary from accounts.xad

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

CREATE FROM

Class

Table basics

Purpose

Create a table from the contents of another table

Syntax

```
CREATE <filename1> | (<expC1>) FROM <filename2> | (<expC2>)
```

See Also

CREATE, COPY STRUCTURE EXTENDED

Description

The CREATE FROM command creates a new table, determined by the contents of a table created with the COPY STRUCTURE EXTENDED command. The filenames can be substituted with a <expC1>, enclosed in round brackets, which returns a valid filename. The FROM table consists of five fields, these are:

Field	Type	Width	Description
FIELD_NAME	Character	32	Field name
FIELD_TYPE	Character	1	Data type of field
FIELD_LEN	Numeric	3	Width of field
FIELD_DEC	Numeric	3	Number of decimal places
FIELD_DES	Character	25	Field Description

Each record of the FROM table defines a field. The structure of the new table, <filename1>, will be created from these field definitions.

Example

```
use patrons
copy structure extended to patstru
close patrons
create newpatron from patstru
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

CREATE GATEWAY

Class

Data Connectivity

Purpose

Create a gateway file

Syntax

CREATE GATEWAY <.gtw filename> | (<expC>)

See Also

SET GATEWAY, GATEWAY()

Description

The CREATE GATEWAY command is used to create a gateway file to an external SQL database table. See Supported Data Sources for the SQL Databases that can be accessed on each individual platform.

The GATEWAY method allows SQL database tables to be accessed via the Recital forms interface. This requires the creation of a gateway file and an empty Recital table that has a structure matching that of the external file. By convention, the empty structure file is given the file extension '.str' rather than the default '.dbf'. SQL Databases can also be accessed directly using Passthrough SQL. For more information on Recital/SQL capabilities, please see the Recital/SQL documentation.

The CREATE GATEWAY worksurface provides a full screen facility for gateway creation in Recital Terminal Developer. The <.gtw filename> can be substituted with any character expression, enclosed in round brackets, that returns a valid filename. If no file extension is specified, then .gtw is used. The following elements can be defined for the gateway:

Server Element	Description
Name	The remote server name, e.g. Oracle
Network Node Name	The node name or IP address of the server
Protocol Type	The connection protocol, DECNET or TCP/IP
Login Username	The login for the server
Login Password	The login password for the server
Database Name	The full name (including path if applicable) of the database
Table Name	The table name from the database
Table Primary Key	The primary index key for the table
Table Restriction	An optional SQL SELECT...WHERE clause

Client Element	Description
Structure Name	The name of the matching Recital structure table
Alias Name	The alias name for the client table
Default Form Name	The default screen form to be used

Example

create gateway employees

Products

Recital Terminal Developer

CREATE LABEL

Class

Terminal Developer Development Tools

Purpose

Create a label definition file through a full screen form

Syntax

CREATE LABEL <filename> | (<expC>)

See Also

CREATE REPORT, REPORT, TREPORT, SET DEVICE TO PRINT, SET PRINT, SET PRINTER, @...SAY

Description

The CREATE LABEL command is a full screen command used to create a label format file, which can subsequently be processed using the LABEL command. The LABEL command is used to generate mailing or other labels. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then '.lbl' is used.

The CREATE LABEL command displays two screens to prompt for the label format details. The first screen is used to define the following information:

Setting	Minimum	Maximum	Default
Width of label	1	120	35
Height of label	1	12	5
Left margin	0	256	0
Lines between labels	0	16	1
Spaces between labels	0	120	0
Number of labels across	1	15	1
Remark	N/A	N/A	3 1/2" x 15/16" x 1

The second screen allows you to enter the Recital/4GL expressions for each line of the label. Any valid expression may be entered. Entering multiple expressions, separated with a comma ',', causes each expression to be evaluated, trimmed and output with a single space between each result. This is the equivalent of trim(<expression>) + " " + trim(<expression>)

Fields in currently active tables can be selected from a popup choicelist when the [HELP] key is pressed or the 'Fields' menu item selected from the menu bar.

Example

use patrons index events
create label operalabels
label form operalabels for event = "OPERA"

Products

Recital Terminal Developer

CREATE REPORT

Class

Terminal Developer Development Tools

Purpose

Report Designer

Syntax

CREATE REPORT <filename> | (<expC>)

See Also

REPORT, TREPORT, SET DEVICE TO PRINT, SET PRINT, SET PRINTER, @...SAY

Description

The CREATE REPORT command activates the full screen Report Designer which is used to create report format files. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then '.frm' is used.

The menu bar for the Report Designer can be activated by pressing the [MENUBAR] key. There are nine options available in the menu bar: <OPTIONS>, <GROUPS>, <COLUMNS>, <FIELDS>, <TRIGGERS>, <RESET>, <LOAD>, <SAVE> and <HELP>.

OPTIONS

The <OPTIONS> menu option is used to go to the Options Screen. This screen is used to define the report layout. You can provide the following information:

Item	Description
Report heading	Specify a heading of up to four lines of text that will appear centered at the top of each page. The optional HEADING clause of the REPORT command can be used to provide an additional heading above this.
Page width	Specify the overall width of the report in characters.
Left margin	Specify the left-hand indent.
Right margin	Specify the right-hand indent.
# lines/page	Specify the number of lines per page.
Double space report (Y/N)?	Specify whether the report should be double-spaced.

GROUPS

The < GROUPS > menu option is used to go to the first Groups screen. This screen is used to define the report groups and subtotals. A maximum of ten group/subtotal expressions may be defined. The table must be indexed or sorted on the group/subtotal expression. You can provide the following information:

Item	Description
Group/Subtotal on	Specify the expression to group by. The [HELP] key will provide a popup choicelist of fields in the active table and open related tables.
Summary report only	Specify whether the report should be a summary. The optional SUMMARY clause of the REPORT command can also be used.
Eject after each group/subtotal	Specify whether the page should be ejected after completion of each group/subtotal.
Group/subtotal heading	Specify a group/subtotal heading of up to four lines of text. This must be specified in order for the subtotal to be displayed. The value returned from the group/subtotal expression is displayed next to this heading.

COLUMNS

The <COLUMNS> menu option is used to go to the first Columns screen. The Columns screen allows you to specify Recital/4GL expressions that will be evaluated and output as the columns of the report. A maximum of 24 columns may be defined. The 'Columns Remaining' field is update automatically based on the columns defined and the specified page width. The semi-colon character is treated as a request for a new line. If expressions contain memory variables or alias pointers, these must be active when the report is run. The following information may be specified in the Columns screen.

Item	Description
Column contents	Specify the expression to be output. The [HELP] key will provide a popup choicelist of fields in the active table and open related tables.
Decimal places	Specify the number of decimal places for numeric columns.
Total	Specify whether the column, if numeric, should be totaled..
Field header	Specify a column heading of up to four lines of text.
Picture	Specify a picture template for the column. See the @...GET...PICTURE command for more information.
Width	Specify the column width. If the specified width is less than the actual width of the evaluated expression, the result will be word-wrapped.

FIELDS

The <FIELDS> menu option is used to display a popup choicelist of fields from the active table. The [PAGE UP], [PAGE DOWN], [CURSOR UP] and [CURSOR DOWN] keys can be used to move to the required field. If related tables are open in other workareas, the [CURSOR LEFT] and [CURSOR RIGHT] keys can be used to move between the workareas. The [RETURN] key is used to select fields, the [EXIT/SAVE] key to save the selection and the [ABANDON] key to cancel the selection. If the command SET DESCRIPTIONS is ON, then the field descriptions are shown in the menu rather than field names. If fields are selected from tables other than the active table, they are prefixed with the alias pointer. No alias pointer is added to fields from the active table, so this must also be the active table when the report is run. All fields selected are automatically converted to character data types.

TRIGGERS

The <TRIGGERS> menu option is used to display a pulldown menu to allow the definition of report level triggers. Triggers are procedures executed when particular events take place. The name of the procedure to be associated with the particular trigger should be entered. Once the procedure name has been entered, the procedure itself can be created/modified by pressing the [HELP] key while the cursor is on the procedure name. This will display a popup editor to edit the <procedure>.prg file. Procedures can be associated with the following triggers:

Item	Description
Prereport	Called at the start of the report.
Postreport	Called at the end of the report.
Pregroup	Called at the start of a group/subtotal.
Postgroup	Called at the end of a group/subtotal.
Prerecord	Called after a record is read, but before it is processed.
Postrecord	Called after a record has been processed.
Precolumn	Called before a column is output.
Postcolumn	Called after a column is output.

RESET

The <RESET> menu option reinitializes the current report format file to blank. This option deletes all existing definitions in the current workspace.

LOAD

The <LOAD> option displays a popup choicelist with the names of existing report format files. Selection of a file causes that file's definition to be loaded into the current report format. This option deleted all existing definitions in the current workspace.

SAVE

The <SAVE> menu option displays a dialog prompting for a filename. The format file will then be saved to this file.

HELP

The <HELP> menu option accesses the on-line help system.

Example

use patrons index names, events
create report concert
report form concert for event = "CONCERT"

Products

Recital Terminal Developer

CREATE SCREEN

Class

Terminal Developer Development Tools

Purpose

Activate the Forms Designer

Syntax

CREATE SCREEN <filename> | (<expC>)

See Also

@...GET...MENU, MODIFY SCREEN, EDIT, CHANGE, APPEND, SET FORMAT

Description

The WYSIWYG Forms Designer is used to design the layout of forms, allowing you to create custom forms which can be used instead of the default forms with the following Recital/4GL screen form commands:

- APPEND
- EDIT
- CHANGE
- INSERT
- QUERY

The file name can be substituted with any character expression, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then '.scr' is used.

The Forms Designer work surface can be navigated with the cursor keys. It also has a full menu driven interface, allowing the following objects to be added to the form and configured. Each object has a corresponding Recital/4GL command that will be generated automatically by the Forms Designer when the screen form is saved.

Object	Description
Text	Simply type the text onto the work surface where you would like the text to appear. Text can be inserted before other text or overwrite existing text. Cut and paste facilities are also available. The corresponding command is @...SAY.
Lines and boxes	Lines (horizontal and vertical) and boxes can be added to the form using the <OPTIONS> menu item. A plus character marks the start and end of the object coordinates. Lines and boxes may be expanded and moved. The corresponding command is @...TO.
Fields	The <SELECT> and <MODIFY> menu items allow fields from multiple tables to be loaded onto the work surface. Each field inherits the attributes from the Application Data Dictionary. The corresponding command is @...GET.
Memory Variables	Memory variables can be added using the <MODIFY> menu item. The memory variables must be declared and initialized before the form is activated. The corresponding command is @...GET.
Menus	The <MODIFY> menu item provides facilities for placing menu options on the work surface and defining the commands to execute when they are selected. Pull down menus can be attached to a menu option. The corresponding command is @...MENU.

Table fields	The <TABLES> menu item is used to define multiple record child table fields. Before using this option, relevant tables, indexes, and relations must be created. The <MODIFY> menu is used to specify the source of fields contained in the Table Field. The corresponding commands are DEFINE TABLE and @...GET.
Check boxes	Check boxes are used for logical fields or memory variables. Each check box can be checked (true) or not (false). There is no interdependency between individual check boxes. In the Forms Designer, Check Boxes are defined using the <MODIFY> menu item. The corresponding command is @...GET...BUTTON
Radio buttons	Radio buttons form groups. Only one radio button can be selected from the group at a time and this selected button contains the value of a character field or memory variable. In the Forms Designer, Radio Buttons are defined using the <MODIFY> menu item. The corresponding command is @...GET...BUTTON...GROUP

The menu bar for the Forms Designer workspace can be activated by pressing the [MENUBAR] key. When the command MCONFIRM is set ON, the Forms Designer menu bar operates as pulldown menus. There are eight options available in the menu bar: <SETUP>, <MODIFY>, <OPTIONS>, <TRIGGERS>, <COLORS>, <WINDOW>, <DICTIONARY> and <HELP>.

SETUP

The <SETUP> menu bar option displays a pop-up menu containing the following options:

Option	Description
Select database	Select a table for further use. Tables can be selected from the dialog menu box displayed to the right of the < Select database...> menu.
Load fields	Select fields to load onto the form. Fields can be selected from the dialog menu box displayed to the right of the < Load fields...> menu. The cursor keys, [PAGE UP] and [PAGE DOWN] are used to highlight each field in turn, the [RETURN] key to select the highlighted field and the [EXIT/SAVE] key to save the selections. The selected fields will be displayed on the form on consecutive rows starting at the current row/column position. The <Aspect is horizontal/vertical> menu option on the <Options> menu will change the way fields are loaded onto the form. The field name or description (SET DESCRIPTIONS ON) is displayed, and an edit region the length of the field is displayed in reverse video, one space to the right.
Create database	Allows you to create a new table while in the Forms Designer.
Quick form	Select all the fields from the current table and place them onto the form.
Select next	Selects the next workarea.
Current workarea	Displays the name of the table open in the current workarea.
Select previous	Selects the previous workarea.

MODIFY

The <MODIFY> option of the menu bar is used to add or change any of the following:-

- Constant SAY text on the form.
- GET fields on the form.
- MENU options on the form.
- Validations and triggers for GET fields on the form.
- Check boxes and radio buttons.

When selected, <MODIFY> displays a menu. The menu items displayed depend on the current cursor position. If the cursor is currently positioned at the start of a field edit region, the menu will contain a list

of field validation criteria and ‘triggers’. If the cursor is positioned on a menu option, then the menu contains a list of attributes for the menu option. Otherwise, <MODIFY> displays a blank menu for a GET action. The action can also be changed to SAY, MENU, or BUTTON. You need only enter the first character to display the appropriate menu (G, S, M, or B).

The validation and triggers that can be specified for a GET action are as follows:-

Item	Description
Action:	Get (can be Get/Say/Menu/Button)
Source:	Table alias name (or ‘m’ for memory variables). If an existing <Source> table is not currently open, it will be opened automatically.
Content:	Name of field or memory variable. Fields will be searched for in the currently selected table, and if found, the ‘Type’ and ‘Width’ fields will be filled in automatically.
Type:	Data type
Width:	Width of field
Decimal:	Number of decimal places
Picture:	Picture validation string. If @s <expN> is specified for a character field, then the field will be scrollable. The <expN> specifies the display width. The field will be scrollable up to the field width.
Range:	For numeric and date fields, a range in the form <low>,<high>. This item may also be used to specify a pop-up choicelist for a character field by preceding the choicelist definition with an “@” character. Use two “@” characters to specify a “dynamic” browse menu. To specify a UDF to return a value to place in a field, precede the UDF name with an “@” character and a question mark: @?UDF()
Help:	Help message to be displayed when the field has focus.
Relation:	Defines this field as a key field for a related table. The alias name is specified to indicate the target of the relation. Changing the field value causes the pointer in the related table to move to that value.
Lookup:	Defines a validity check using a cross-table lookup. The alias name of the lookup table must be specified. Changing the field value causes the lookup table to be scanned for the new value. If the value does not exist in the lookup table, a validation failed error message is displayed.
Validation:	Any of the following validation clauses can be specified: an input validation procedure name, an @ sign followed by a boolean condition, or a validation string prefixed with a ‘\$’.
Error:	The error message to be displayed if validation fails..
Calculated:	An expression that will be evaluated to provide the field’s value. The field will be read only.
Recalculate:	Sets a flag so that if the field value changes, all CALCULATED expressions will be reevaluated and the form will be refreshed.
Must enter:	Force data to be input in this field.
Read when:	Specify a boolean condition. The field will be read only unless the condition evaluates to true (.T.).
PreTrigger:	Specify a procedure name for the PreField trigger. The PreField trigger procedure is called as the field is entered. Once a procedure name has been entered, pressing the [HELP] key with the cursor on the procedure name pops up a notepad to allow the procedure to be edited.
PostTrigger:	Specify a procedure name for the PostField trigger. The PostField trigger procedure is called as the field is exited. Once a procedure name has been entered, pressing the [HELP] key with the cursor on the procedure name pops up a notepad to allow the procedure to be edited.

If the <Action> item is an @...SAY, this field will not be refreshed as you page up and down through the form. The command SET PCSAYS must be set ON for @...SAY to be refreshed.

When in the <MODIFY> menu, you can move between the fields on the form using the [PAGE DOWN] and the [PAGE UP] keys. When you have completed the field definition, you should commit the changes to the form by exiting from the modify menu using the [EXIT/SAVE] key. Pressing the [ABANDON] key discards the changes made to the definition, and exits the menu.

When you specify a MENU <Action>, then the following menu items will be displayed.

Item	Description
Action:	Menu
Item:	The menu option to be displayed on the form.
Command:	A list of commands to be executed when the menu is selected. Multiple commands can be specified, by separating each command with a semicolon.
Help:	A help message to be displayed in the message line when the cursor is positioned on the menu option
Pulldown:	A pulldown menu definition which consists of a list of pulldown options separated by commas, or @<procedure-name>. See @...MENU for details

When you specify a BUTTON <Action>, the following menu items will be requested:

Item	Description
Source	Table name (or 'm' for memory variables)
Content	Name of field or memory variable
Name	The name of the button which will be passed to the User Defined Function (UDF).
Label	Enter a descriptive label for button. This will be displayed next to the button when SET DESCRIPTIONS is ON.
Group	If you are creating a radio button, enter a group name. Each radio button in a list of choices must be given the same group name.
Help	A help message to be displayed in the message line when the cursor moves on to this button.
Trigger	The name of a trigger procedure to execute when this button is selected.

Check boxes and radio buttons are selection objects that can help end users enter data quickly and without error. Selected with the [SPACEBAR] key, check boxes and radio buttons display an asterisk (*) in their entry fields when selected. Check boxes are represented by square brackets: []. Radio buttons are represented by round brackets (). Check boxes are used when you have a list of choices from which any combination of those choices can be selected. The user can select one, all or none of these options depending on their selection needs. Each check box corresponds to an individual logical field or memory variable. Radio buttons are used when only one choice out of a group of choices may be selected at a time. Each radio button corresponds to a possible value for the same character field or memory variable.

OPTIONS

The <OPTIONS> menu contains items to add lines and boxes to the form, create or edit triggers and programs, change current settings for the Forms Designer, and access table, environment, and memory variable information.

Item	Description
Box/Line	Boxes and lines can be added to the form using this option. Select the option, then position the cursor to the top left corner and press [RETURN], finally, position the cursor to the lower right corner and press [RETURN] again.
Program	The <Program...> menu item allows you to create or modify program files. It prompts for a filename then calls the default editor.
Descriptions are ON/OFF	The <Descriptions are ON/OFF> menu item allows you to toggle between names and descriptions as your field labels. When you select this item, the setting will change from ON to OFF or vice versa.

Labels are ON/OFF	Toggle field labels display on or off. Labels must be off for table fields.
Aspect is vertical/horizontal	Toggle between and vertical or horizontal field loading. Horizontal field loading is required for table fields
Mconfirm is ON/OFF	Toggle the SET MCONFIRM ON OFF command. When MCONFIRM is OFF, the menu bar operates as pulldown menus. When MCONFIRM is ON, menus must be selected with the [RETURN] key.
Display database structure	Displays a screen containing information on the currently active table. (DISPLAY STRUCTURE)
Print database structure	Prints the above information.
Display Environment Status	Displays information regarding the environment. (DISPLAY STATUS).
Print environment status	Prints the above information.
Display memory variables	Displays a screen containing information on current memory variables. (DISPLAY MEMORY)
Print memory variables	Prints the above information.

TRIGGERS

The <TRIGGERS> option allows you to specify or remove triggers for your form. Selecting the <TRIGGERS> option displays a pulldown menu of the following triggers:

Trigger	Description
Preform	Execute a procedure before the form is displayed
Postform	Execute a procedure as the format file is exited
Prerecord	Execute a procedure prior to the first @...GET
Postrecord	Execute a procedure as a record is updated
PreMenu	Execute a procedure before a menu is entered.
PostMenu	Execute a procedure after a menu is exited.

If the [HELP] key is pressed when on a name of trigger program, the program can be edited in a popup notepad. For more information on triggers see the SET PREFORM, SET POSTFORM, SET PRERECORD, SET POSTCORD, SET PREMENU and SET POSTMENU commands.

COLORS

The <COLORS> option displays a menu for setting colors on individual @...GET, @...MENU and @...SAY objects. If the cursor is currently positioned at the start of a field edit region or menu item, then this object will be displayed, otherwise the first object from position 0,0 will be displayed. If there are no objects on the Forms Designer., the <COLORS> menu will not display. The [PAGEUP] and [PAGE DOWN] keys are used to move between all the objects on the work surface. Both the foreground and background colors can be set from this menu. The following colors may be defined:-

Color	Attribute Code
BLACK	N or blank
BLUE	B
GREEN	G
CYAN	BG
BLANK	X
GREY	N+
RED	R
MAGENTA	RB
BROWN	GR
YELLOW	GR+
WHITE	W

If the [HELP] key is pressed, then a pop-up choice list of available colors will be displayed.

TABLES

The <TABLES> option is for defining a table field. Table fields are used for browsing and updating multiple 'child' records related to a 'parent' table. Before using this option, the relevant tables, indexes and relations should be set up. The number of table fields you may define for one form is limited only by the number of available workareas and the screen space. After the <TABLES> menu option is selected, a form appears prompting for the following information:

Item	Description
Table name:	Enter in a unique name for the table. Names must start with a letter (a-z) or underscore and can include letters, digits and underscores. Names must be a maximum of ten characters long.
Database:	Enter the alias name of the source database table. Pressing the [HELP] key will display a choice list of database tables in the current directory.
#of rows:	Enter the number of rows the table will occupy.
Related field:	Enter the key field for records in the child table. The table must be indexed on this field and this must be the master index order.
Related by:	Enter the key expression from the parent table including the alias name and alias pointer, e.g. parent->linkfield or parent.linkfield.
Column titles:	Enter in a title for each column in the table. Individual titles should be comma separated. This is optional and if used, a box will be drawn around the table.
Foreground:	Used to specify the foreground color for the table. Pressing the [HELP] key will display a choice list of colors. If a color is not specified, the foreground color of FIELDS will be used.
Background:	Used to specify the background color for the table. Pressing the [HELP] key will display a choice list of colors. If a color is not specified, the background color of FIELDS will be used.
BoxForeground:	Used to specify the foreground color of the box. Pressing the [HELP] key will display a choice list of colors. If a color is not specified, the foreground color of BOX will be used. Column titles must be specified for the box to be drawn.
BoxBackground:	Used to specify the background color of the box. Pressing the [HELP] key will display a choice list of colors. If a color is not specified, the background color of BOX will be used. Column titles must be specified for the box to be drawn.
Shadow:	If titles have been specified you may give the box a shadow on the right hand side and the bottom edge by specifying "Yes".
PreTrigger:	Used to specify the name of a trigger procedure that will execute as the cursor enters the table field.
PostTrigger:	Used to specify the name of a trigger procedure that will execute as the cursor exits the table field.

The [EXIT/SAVE] key should be used to exit the <TABLES> menu and save the definition. Pressing the [ABANDON] key will discard the definition. To define a second or further table field, press the [PAGE DOWN] key and this will redisplay a blank form.

The <Aspect is horizontal> and <Labels are OFF> options should be selected before loading fields to go into the table field, so that the fields form a horizontal row with a space between each edit region. The <Source> of the @...GETS fields that belong to the table field must be updated. To do this, move the cursor to the first field edit region that belongs to the table field and choose the <MODIFY> option from the menu. The alias name for each @...GET is displayed in the <Source> field. Prefix the alias name for each @...GET in the table field with the table name followed by a "!". This identifies the @...GETS as belonging to the specified table.

For example, if the table name for the table field is HW and the alias name of HARDWARE, the SOURCE field for each @...GET should read HW!HARDWARE.

Child tables that relate to a parent record on the same form must be positioned below the first accessible parent record field.

DICTIONARY

The <DICTIONARY> menu contains two options: <Display> and <Print>. The <Display> option displays the Application Data Dictionary for the currently active database in read-only mode. If there is no dictionary for the currently active, there will be no display. You may scroll through the fields by pressing the [PAGE DOWN] and [PAGE UP] keys. The data values for each field are displayed beneath the field values. The <Print> option prints the currently active Application Data Dictionary.

HELP

The <HELP> menu contains two options: <Help...> and <Keys help...>. The <Help...> option accesses the on-line Help system. The <Key help...> option displays the Function key usage chart.

The following keys are active in CREATE SCREEN:

Key	Action
EXIT SAVE	Save the changes and exit the Forms Designer
ABANDON	Abandon changes and exit the Forms Designer
CURSOR UP	Move up one line
CURSOR DOWN	Move down one line
CURSOR LEFT	Move one column to the left
CURSOR RIGHT	Move one column to the right
HELP	Display a popup key usage help page
RETURN	When the cursor is positioned on the start of a field, allows repositioning of the field on the form; when positioned on a menu option, allows repositioning on the menu option; when positioning on a box or line, allows them to be dragged
DELETE CHAR	Delete the character under the cursor
BACKSPACE	Delete the character before the cursor
MENUBAR	Activate the menu bar
^Z	Move to the start of the line
^B	Move to the end of the line
^T	Delete from the cursor to the next word
^Y	Delete the current line
^U	If the cursor is positioned on an object - field, menu option, box or table field – the object will be removed from the form
^N	Insert a blank line
^V	Toggle between insert and overwrite mode
^K	Insert a blank column from the current cursor position to the end of the form
^L	Delete the column from the current cursor position to the end of the form

When the [EXIT/SAVE] key is pressed, the following three operations are performed:

The screen image is saved in a file with a “.scr” extension

A screen format is generated to a file with a “.fmt” extension.

The Forms Designer is exited.

The screen format file can be viewed or edited using a text editor. Care must be taken, however if changes are made to the format file. These changes will not be reflected in the <.scr filename> and any subsequent use of the MODIFY SCREEN command will use the original ‘.scr’ file not the updated ‘.fmt’ file.

Example

create screen myform

Products

Recital Terminal Developer

CREATE VIEW

Class

Terminal Developer Development Tools

Purpose

Activate the View Designer

Syntax

CREATE VIEW <filename> | (<expC>)
[FROM ENVIRONMENT]

See Also

SET VIEW TO, USE

Description

The CREATE VIEW command activates the full screen View Designer which is used to create a view file, which can be processed using the SET VIEW TO command. View files allow you to treat multiple databases and their associated files as a single object. Views are very useful for performing ad-hoc queries on data that is in multiple related databases.

<filename> | (<expC>)

The file name can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then '.vue' is used.

FROM ENVIRONMENT

If the optional FROM ENVIRONMENT clause is specified, a view file is automatically created for the established environment.

Without the FROM ENVIRONMENT clause, the CREATE VIEW command displays a screen for each workarea (1–10), prompting for the following details:

Field	Description
Database	The name of the table to be opened in this workarea
Alias	The alias name for the table. By default this is the workarea letter A-J
Format	The screen format file to be used. (CREATE SCREEN, SET FORMAT TO)
Index (1-7)	The names of up to seven single index files (.ndx)
Filter	The filter expression for the table
Relation to	The name of a field in the current table used to link to another table
Into	The name of the table that is the target of the relation

The menubar for the View Designer can be activated by pressing the [MENUBAR] key. The following options are available:

Menu	Description
Workarea	Displays a DIALOG GET to specify the workarea number (1-10) to select
Databases	Displays a popup choicelist of tables in the current directory
Formats	Displays a popup choicelist of screen format files in the current directory
Indexes	Displays a popup choicelist of single index files in the current directory
Query	Displays a popup query builder. This is used for constructing a filter expression
Help	Displays the on-line help system

A specific view file can be opened directly using the SET VIEW TO <filename> command, or by creating a VIEW BRIDGE with the CREATE BRIDGE command then issuing the USE <view bridge> command.

All Recital products support the SQL CREATE VIEW command. This creates a logical view of one or more tables based on a SELECT statement. Please see the SQL documentation for further information.

Example

```
create view patrons  
set view to patrons
```

Products

Recital Terminal Developer

DEACTIVATE MENU

Class

Menus

Purpose

Deactivate the currently activated Xbase style menu

Syntax

DEACTIVATE MENU

See Also

ACTIVATE MENU, CLEAR MENUS, DEFINE MENU, SET COMPATIBLE

Description

The DEACTIVATE MENU command deactivates the currently activated Xbase style menu. All related pop-ups and menus as defined in the ON PAD commands are deactivated as well. The menu is removed from the screen but not released from memory, so that it can be displayed again using the ACTIVATE MENU command. The command SET COMPATIBLE should be set ON when using Xbase style menus.

Example

```
on selection pad endit of sort_men;  
  deactivate menu
```

Products

Recital Mirage Server, Recital Terminal Developer

DEACTIVATE POPUP

Class

Menus

Purpose

Remove an Xbase style pop-up menu from the screen

Syntax

DEACTIVATE POPUP

See Also

ACTIVATE POPUP, CLEAR POPUPS, DEFINE POPUP, ON SELECTION, RELEASE POPUPS, SET COMPATIBLE

Description

The DEACTIVATE POPUP command removes the currently activated Xbase style pop-up menu from the screen. The affected pop-up menu is not released from memory when deactivated, and may be activated again using the ACTIVATE POPUP command. The command SET COMPATIBLE should be set ON when using the Xbase style menus.

Example

```
on selection pad endit of popup_4;  
  deactivate popup
```

Products

Recital Mirage Server, Recital Terminal Developer

DEACTIVATE WINDOW

Class

Screen Windows

Purpose

Deactivate and erase a window from the screen

Syntax

DEACTIVATE WINDOW <window-name | window-name list> | ALL

See Also

ACTIVATE WINDOW, DEFINE WINDOW, RELEASE WINDOWS

Description

The DEACTIVATE WINDOW command deactivates a window that has been defined with the DEFINE WINDOW command, and activated with the ACTIVATE WINDOW command. A window is an area of the screen designated for output and input. There is no limit to the number of defined windows.

A defined window is displayed to the screen with the ACTIVATE WINDOW or SHOW WINDOW commands. The SHOW WINDOW command displays a window without activating it. The ACTIVATE WINDOW command displays a window and activates it. When a window is activated, all subsequent output is displayed in that window. Only one window may be activated at a time. The DEACTIVATE WINDOW command clears the display of activated windows, but leaves the window definition in memory. Once a window has been deactivated, all subsequent output is directed to the last activated screen, or to the full screen if there are no active windows.

The DEACTIVATE WINDOW command can be used to deactivate a single window, a list of windows, or all currently defined windows. To deactivate a single window, specify the name of the window that you wish to deactivate. The <window-name> is the name in the window definition created with the DEFINE WINDOW command. The <window-name list> is a list of window names, each separated by a comma. When deactivating a list of windows, the DEACTIVATE WINDOW command deactivates the windows in the order that they are listed. To deactivate all currently defined windows, use the keyword ALL.

If you wish to clear all windows from the screen and from memory, use the CLEAR WINDOWS or RELEASE WINDOWS command. The SAVE and RESTORE WINDOW commands may be used to keep window definitions in a file that can be reused at any time.

Example

```
@ 17,1 menu "Quit Inventory";  
    help "Exit from Inventory System";  
    command "deactivate window inv_win"
```

Products

Recital Mirage Server, Recital Terminal Developer

DEBUG

Class

Error Handling and Debugging

Purpose

Monitor programs during execution

Syntax

DEBUG <program | procedure> [WITH <parameter-list>]

See Also

CANCEL, DISPLAY MEMORY, DISPLAY STATUS, DO, QUIT, RESUME, SET COMPILE, SET DEVELOPMENT, SET HISTORY, SET STEP, SUSPEND

Description

The DEBUG command displays a pop-up debugger allowing the specified program to be monitored during execution.

The debugger consists of four lines of information about the current program and eighteen push buttons. Program information consists of the following lines:

OPERATION	Displays one of the following DEBUG operations: STEP, BREAKPOINT or WATCHPOINT.
PROGRAM	Displays the name of the current procedure or program.
LINE#	Displays the current line number.
COMMAND	Displays the next program line to be executed.

The eighteen push buttons are used to access information about the current environment and to specify memory variables and conditions to monitor during program execution. The pop-up debugger provides the following push buttons:

BUTTON	EFFECT
Step	Step through a line at a time.
Suspend	Suspend the program to go to the interactive prompt, RESUME to restart.
Cancel	Cancel program execution and create error.mem.
Memory	Display currently declared memory variables.
Status	Display currently open tables (and their indexes, current record, etc.).
Calls	Show program/procedure call stack.
Watch	Set a watch point. When Executing the program, execution will stop when the specified memory variable's value changes.
Break	Set a break point. When Executing the program, execution will stop when the specified condition becomes true.
History	Show command history trace.
Execute	Run program without stepping until watch point or break point reached.
Wpclear	Clear all watch points.
Bpclear	Clear all break points.
Bpmark	Mark the current line as a break point.
Bpdrop	Clear a particular break point.
Wpdrop	Clear a particular watch point.
Bpshow	Show all break points.
Wpshow	Show all watch points.

Quit	Exit the program and debugger.
------	--------------------------------

Use the [UP], [DOWN], [LEFT], and [RIGHT] arrow keys to navigate the push buttons, and press the [RETURN] key to select a button. You may also type the accelerator key to select a button. The accelerator keys are the highlighted letters in the label of each push button.

Debugger directives can also be hard-coded into program files. These are ignored in normal program execution but activated when the program runs in debug mode. Debug mode is in effect when the program is called with the DEBUG command or when SET STEP is ON. Directives start with a '//' comment prefix followed by a space, the DEBUG command, a colon (':'), and the condition. The following debugger directives are available:

DIRECTIVE	EFFECT
// DEBUG:BREAKPOINT	Acts as if a break point has been hit when it is encountered.
// DEBUG:BREAKPOINT:<expL>	Acts as if a break point has been hit when <expL> evaluates to .T. (True).
// DEBUG:WATCHPOINT:<memvar>	Sets a watch point on the specified memory variable.

Debugger directives can also be used when programs are run via the Recital Database and Mirage Servers.

Example

```
debug main
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

DECLARE

Class

Array Processing

Purpose

Declare an array

Syntax

```
DECLARE <memvar>
DECLARE <array>[<expN>] | (<expN>)
DECLARE <array>[<expN1>,<expN2>] | (<expN1>,<expN2>)
```

See Also

APPEND FROM ARRAY, COPY TO ARRAY, DIMENSION, GATHER, PRIVATE, PUBLIC, RELEASE, RESTORE FROM, SAVE TO, SCATTER, AVERAGE(), ACHOICE(), ACOPY(), ADEL(), ADIR(), AFIELDS(), AFILL(), AINS(), ALLEN(), AMAX(), AMIN(), ASORT(), ASUM()

Description

The DECLARE command is used to declare memory variables and fixed one or two-dimensional arrays. Memory variables declared using the DECLARE command are initialized to .F. and are private to the declaring procedure.

The size of an array is fixed at its declaration. Both one and two dimensional arrays subscripts can be referenced using square brackets or round brackets.

[<expN>] | (<expN>)

For one-dimensional arrays, [<expN>] specifies the total number of elements in the array. Elements are subsequently referenced using the notation <array>[<expN>] or <array>(<expN>).

[<expN1>,<expN2>] | (<expN1>,<expN2>)

For two-dimensional arrays, <expN1> represents the number of rows and <expN2> represents the number of columns in the array. Elements are subsequently referenced by <array>[<expN1>,<expN2>] or <array>(<expN1>,<expN2>). The elements of a two dimensional array can also be referenced, as if the array were one dimensional, using <array>[<expN1>] or <array>(<expN>).

Arrays can be declared as any size. Values are assigned into arrays using the '=' operator. Arrays can then be used in a similar way to memory variables. Complete arrays can be initialized with one assignment. Array references start at 1,1 for two-dimensional arrays, and 1 for one-dimensional arrays. They can also be declared as private or public by using the PRIVATE and PUBLIC commands. Arrays can be saved to memory files with the SAVE TO command and then later restored with the RESTORE FROM command.

Notes: The brackets shown for this command do not indicate optional expressions but are a necessary part of the syntax.

Example

```
// Declare one-dimensional array of 4000 elements
declare aTable[4000]
// Assign 0 to all elements
aTable = 0
// Insert individual element values into array
aTable[1] = 10
aTable[2] = "Hello"
aTable[3] = "World"
```

```
aTable[4] = date()  
// Print value of element 2  
? aTable[2]  
Hello
```

```
// Another example  
declare twodim[3,3]  
twodim[2,3] = "hello world"  
? twodim[6]  
hello world
```

```
// Another example  
use payroll  
declare aPayroll[reccount(), fcount()]  
copy to array aPayroll for city = "LONDON"
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

DECRYPT

Class

DES3 Encryption

Purpose

Decrypt the specified table or tables

Syntax

DECRYPT <.dbf> | <skeleton> KEY <expC1>

See Also

APPEND FROM, COPY FILE, COPY STRUCTURE, COPY TO, DIR, ENCRYPT, USE, SET ENCRYPTION

Description

The DECRYPT command is used to decrypt the data in the specified table, <.dbf> or tables matching the <skeleton>. The <expC1> must contain the three part comma-separated key used to previously encrypt the table. The key may optionally be enclosed in angled brackets. The <skeleton> syntax can only be used if all tables matching the <skeleton> have the same key.

The DECRYPT command decrypts the data and removes the table's .dkf file. After decryption, the key need no longer be specified to gain access to the table.

Example

```
decrypt accounts key "key1,key2,key3"  
decrypt salaries key "<key_1,key_2,key_3>"
```

```
// decrypt all .dbf files in the directory  
decrypt *.dbf key "key1,key2,key3"
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

#DEFINE

Class

Memory Variables

Purpose

Defines a constant

Syntax

```
#DEFINE <memvar> <exp>
```

See Also

#IFDEF..ENDIF, LOCAL, PRIVATE, PUBLIC

Description

The #DEFINE command is used to define FoxPro compatible constants. Constants declared using #DEFINE can be overridden by a memory variable of the same name, but cannot be modified or manually released after their initial declaration. Constants are automatically updated if the value of <exp> changes and are released on exit from the session.

Example

```
#DEFINE NEXT_LOOP
for i = 1 to NEXT_LOOP
    ? i
next
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

DEFINE BAR

Class

Menus

Purpose

Defines an option for an Xbase style pop-up menu

Syntax

```
DEFINE BAR <expN> OF <expC1> PROMPT <expC2>  
[KEY <key>, [<expC3>]]  
[MESSAGE <expC4>]  
[SKIP [FOR <expL>]]
```

See Also

ACTIVATE POPUP, BAR(), DEFINE POPUP, DIALOG FIELDS, DIALOG FILES LIKE, DIALOG QUERY, DIALOG SCOPE, MENU FIELDS, MENU FILES LIKE, MENU QUERY, MENU SCOPE, ON SELECTION POPUP, POPUP(), PRMBAR(), PROMPT(), SET COMPATIBLE

Description

The DEFINE BAR <expN> defines a single option in a pre-defined Xbase style pop-up <expC1> menu. If you have used the PROMPT FIELD, PROMPT FILES, or the PROMPT STRUCTURE options in your pop-up menu definition, the DEFINE BAR command is unnecessary. The command SET COMPATIBLE should be set ON when using Xbase style menus. The PROMPT <expC2> defines the text that will appear in the menu item.

KEY <key>, [<expC3>]

The KEY <key>, [<expC3>] is provided for Xbase language compatibility only. The value <key> is the key combination defined to access the particular option. The <expC> defines the label displayed on the option bar.

MESSAGE <expC4>

The optional MESSAGE <expC4> displays a message centered on the last line of the screen when the bar is selected by the user.

SKIP FOR [<expL>]

The optional SKIP command allows the display of a defined bar but does not allow selection of that bar. The optional SKIP FOR <expL> allows selection of a defined bar only when the specified condition is true.

Example

```
define popup popup_4;  
  from 1,26  
define bar 1 of popup_4;  
  prompt "\<Bar 1"  
define bar 2 of popup_4;  
  prompt "B\<ar 2"  
define bar 3 of popup_4;  
  prompt "Ba\<r 3"  
activate popup popup_4
```

Products

Recital Mirage Server, Recital Terminal Developer

DEFINE CLASS

Class

Objects

Purpose

Create a user-defined class

Syntax

```
DEFINE CLASS <class name> [AS <base class> | CUSTOM [OLEPUBLIC]]  
    [[PROTECTED | HIDDEN PropertyName1, PropertyName2 ...]  
        [<object>.<property> = <exp> ...]  
    [ADD OBJECT [PROTECTED] <object name> AS <base class> [NOINIT]  
        [WITH <property-list>]]...  
    [[PROTECTED | HIDDEN] FUNCTION | PROCEDURE <proc-name>[_ACCESS | _ASSIGN]  
        | THIS_ACCESS [NODEFAULT]  
        <command statements>  
    [ENDFUNC | ENDPROC]]...  
ENDDEFINE
```

See Also

CLASS, PUBLIC, PRIVATE, LOCAL, STATIC, METHOD, SET COMPATIBLE

Description

The DEFINE CLASS command is the Visual FoxPro class definition command. Within the DEFINE CLASS...ENDDEFINE block all aspects of the class – its name, events, methods and properties can be specified. The CREATEOBJECT() function is used to create an object based on a defined class.

The <class name> defines the reference for the class.

AS <base class> | CUSTOM [OLEPUBLIC]

The AS <base class> clause is used to specify the parent system class for the current class being defined. To specify a user-defined class, use CUSTOM. If the OLEPUBLIC keyword is included, this means that the class in an Automation server can be accessed by an Automation client.

PROTECTED

This affects subsequent property declarations and ensures that PROTECTED properties cannot be accessed or changed outside the scope of the current class or subclasses based on this class.

HIDDEN

This affects subsequent property declarations and ensures that HIDDEN properties cannot be accessed or changed outside the scope of the current class, not even by sub-classes.

Member Declaration <property> = <exp>

Properties can be assigned values when an object based on this class is instantiated by including the assignments in the class definition.

ADD OBJECT <object name>

This adds the specified object to the class definition from a Visual FoxPro base class, a user-defined class or an ActiveX custom control.

PROTECTED

This affects the properties of the object and ensures that they cannot be accessed or changed outside the scope of the current class or subclasses based on this class.

AS <base class>

The AS <base class> clause is used to specify the class on which the object is based.

NOINIT

If the NOINIT keyword is specified, the init method of the object will not be called when the object is added.

WITH <property-list>

The WITH <property-list> clause specifies the object's properties and their values. The <property-list> consists of comma separated property=value pairs.

FUNCTION | PROCEDURE <proc-name>
<command statements>

ENDFUNC | ENDPROC

The FUNCTION or PROCEDURE clause is used to define the class's events and methods. The <command statements> are the operations to be performed. The FUNCTION or PROCEDURE can optionally be terminated with the appropriate ENDFUNC or ENDPROC command. Events and methods are called using the object.method | object.event syntax.

_ACCESS | _ASSIGN

If the _ACCESS or the _ASSIGN suffix is added to the name of a procedure or function, this will create an ACCESS method or an ASSIGN method for the property with the same name. ACCESS methods are called whenever the property value is requested and ASSIGN methods are called whenever the property value is changed.

THIS_ACCESS

If THIS_ACCESS is specified the procedure or function will be called whenever an attempt is made to change the value of a member of an object and whenever a member of an object is queried.

NODEFAULT

Including the NODEFAULT keyword prevents the default event or method being performed.

ENDDEFINE

The class definition is terminated with the ENDDEFINE command.

Example

```
define class myClass as custom
    productname = "Recital Mirage"
    version = "2.0"
enddefine
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

DEFINE MENU

Class

Menus

Purpose

Define an Xbase style main menu bar

Syntax

```
DEFINE MENU <expC1>  
[COLOR SCHEME <expN>]  
[IN [WINDOW] <window name> | IN SCREEN]  
[MESSAGE <expC2>]
```

See Also

DEFINE PAD, ON PAD, ON SELECTION PAD, ACTIVATE MENU, DEACTIVATE MENU, HIDE MENU, RELEASE MENUS, SHOW MENU, DEFINE POPUP, DEFINE BAR, ON SELECTION POPUP, ACTIVATE POPUP, DEACTIVATE POPUP, RELEASE POPUPS, CLEAR POPUPS, SHOW POPUP, SET COMPATIBLE

Description

The DEFINE MENU command defines the menu name <expC1> of the main menu bar for Xbase style menus. The command does not create a menu bar on its own, but serves as the first step in the process. The next steps include creating options along the menu bar with the DEFINE PAD command and associating actions with the options with the ON PAD and ON SELECTION PAD commands. The command SET COMPATIBLE should be set ON when using Xbase style menus.

COLOR SCHEME <expN>

The optional COLOR SCHEME command will display the menu items in the colors defined in scheme<expN>. Color schemes are defined by the command SET COLOR OF SCHEME command.

IN [WINDOW] <window> | IN SCREEN

The optional IN [WINDOW] <window> | IN SCREEN clause allows the menu to be defined for display in a specified window, IN [WINDOW] <window name> or to be defined for display in the main screen when a window is currently active, IN SCREEN.

MESSAGE <expC2>

The optional MESSAGE command will display <expC2> in the message line.

Example

```
define menu sort_men message "Main Menu"
```

Products

Recital Mirage Server, Recital Terminal Developer

DEFINE PAD

Class

Menus

Purpose

Define pad locations for the main menu bar of an Xbase style menu

Syntax

```
DEFINE PAD <expC1> OF <expC2> PROMPT <expC3>
[AT <expN1>,<expN2>]
[COLOR <color code> | COLOR SCHEME <color scheme>]
[KEY <key label> [,<expC4>]
[MESSAGE <expC5>]
[SKIP [FOR <expL>]]
```

See Also

DEFINE MENU, ON PAD, ON SELECTION PAD, ACTIVATE MENU, DEACTIVATE MENU, RELEASE MENUS, SHOW MENU, DEFINE POPUP, DEFINE BAR, ON SELECTION POPUP, ACTIVATE POPUP, DEACTIVATE POPUP, RELEASE POPUPS, CLEAR POPUPS, SHOW POPUP, SET COMPATIBLE

Description

The DEFINE PAD command defines the pad name <expC1> for the menu name <expC2> for Xbase style menus. The command SET COMPATIBLE should be set ON when using Xbase style menus. The menu name must first be defined with the DEFINE MENU command.

PROMPT <expC3>

The PROMPT <expC3> is displayed inside the menu option, placing a blank space to each side of the prompt when it is displayed.

AT <expN1>,<expN2>

The optional AT coordinates position the pad at row <expN1> and column <expN2> on the screen. If the AT coordinates are not supplied, the pads start in the upper left corner of the screen and place one space in between each prompt.

COLOR <color code> | COLOR SCHEME <color scheme>

The optional COLOR or COLOR SCHEME clause will display the pad in the colors defined by the <color code> foreground/background color pair or the numeric <color scheme>. For more information about color settings, please see the SET COLOR and SET COLOR OF SCHEME commands.

KEY <key label> [,<expC4>]

The optional KEY clause allows the pad to be chosen using the specified <key label>. For more information on key labels, please see the ON KEY LABEL command. If <expC4> is included, this character expression will be displayed instead of the default key label.

MESSAGE <expC5>

The optional MESSAGE clause displays <expC4> in the message line when the cursor is placed on the pad.

SKIP [FOR <expL>]]

The SKIP clause allows a particular pad or to be disabled. If the optional FOR <expL> clause is included, the <expL> is evaluated. If the evaluation results in .T. (true) the pad is disabled, if .F. (false), the pad is enabled. If the FOR <expL> clause is not included, the pad is enabled.

Example

```
define pad file_nm of sort_men;  
  prompt "Files";  
  at 0,0;  
  message "Use <- and -> keys to navigate menus"  
define pad srt_type of sort_men;  
  prompt "Structure";  
  at 0,7;  
  message "Use <- and -> keys to navigate menus"  
define pad key_nm of sort_men;  
  prompt "Fields";  
  at 0,18;  
  message "Use <- and -> keys to navigate menus"  
define pad prfrm of sort_men;  
  prompt "User-defined";  
  at 0,26;  
  message "Use <- and -> keys to navigate menus"  
define pad endit of sort_men;  
  prompt "Quit";  
  at 0,40;  
  message "Press ENTER to exit"
```

Products

Recital Mirage Server, Recital Terminal Developer

DEFINE POPUP

Class

Menus

Purpose

Define an Xbase style pop-up menu.

Syntax

```
DEFINE POPUP <expC1> FROM <expN1>, <expN2> [TO <expN3>, <expN4>]  
[COLOR SCHEME <expN5>]  
[FOOTER <expC2>]  
[MESSAGE <expC3>]  
[PROMPT FIELD <expC4>]  
[PROMPT FILES [LIKE <skeleton>]]  
[PROMPT QUERY]  
[PROMPT SCOPE]  
[PROMPT STRUCTURE]
```

See Also

ACTIVATE POPUP, BAR(), DEFINE BAR, DIALOG FIELDS, DIALOG FILES LIKE, DIALOG QUERY, DIALOG SCOPE, MENU FIELDS, MENU FILES LIKE, MENU QUERY, MENU SCOPE, ON SELECTION POPUP, POPUP(), PRMBAR(), PROMPT(), SET COMPATIBLE

Description

The DEFINE POPUP command names, shapes and defines an Xbase style pop-up menu. The name of the popup is specified using <expC1>. The FROM <expN1>, <expN2> coordinates are used to align the pop-up menu beneath the appropriate pad of the menu bar. The <expN1> specifies the top left row coordinate, the <expN2> the top left column coordinate. The bottom right row and column coordinates can be specified using the optional TO <expN3>, <expN4>. If these coordinates are not supplied, the pop-up menu will adjust to the longest field and will include the maximum number of lines.

COLOR SCHEME <expN5>

The optional COLOR SCHEME command will display the menu items in the colors defined in scheme<expN>. Color schemes are defined by the command SET COLOR OF SCHEME command.

FOOTER <expC2>

The FOOTER <expC2> option causes the specified text, <expC2> to be displayed centered at the bottom of the popup.

MESSAGE <expC3>

The MESSAGE option centers the character expression, <expC3>, on the screen message line.

PROMPT FIELD <expC4>

The PROMPT FIELD <expC4> fills the pop-up window with the contents of the specified field for every record in the table.

PROMPT FILES

The PROMPT FILES option displays a list of filenames. Specifying the optional LIKE <skeleton> clause, restricts the files to those that match the skeleton.

PROMPT STRUCTURE

The PROMPT STRUCTURE option fills the pop-up window with fields from the current table

PROMPT QUERY

The PROMPT QUERY option displays the query menu from the current table on the screen

PROMPT SCOPE

The PROMPT SCOPE option displays the scope menu on the screen

Example

```
define popup popup_1;  
  from 1,0;  
  prompt files  
define popup popup_2;  
  from 1,7;  
define popup popup_3;  
  from 1,18;  
  prompt field name
```

Products

Recital Mirage Server, Recital Terminal Developer

DEFINE TABLE

Class

Screen Forms

Purpose

Define a table field in a format or program file for multiple record display

Syntax

```
DEFINE TABLE <table name> FOR <.dbf filename> AS <expN> ROWS
RELATING <field> BY <key expression>
[BOXCOLOR <color>]
[COLOR <color>]
[PRETABLE <procedure name> |<expC1>]
[POSTTABLE <procedure name> |<expC2>]
[SHADOW]
[TITLE <expC3>]
```

See Also

CREATE BRIDGE, CREATE SCREEN, CREATE VIEW, SET RELATION @...GET

Description

The DEFINE TABLE command designates a table of multiple record based 'child' table information to be viewed on the same form with other related 'child' information and, if desired, the 'parent' table information. One to many relationships are represented and maintained in a form. As users page through records in a parent table, records in the related table fields refresh accordingly.

The tables, indexes and relational links should be created prior to issuing the DEFINE TABLE command. This may be accomplished by creating a VIEW, an APPLICATION BRIDGE, or a program file.

The number of TABLES that may be defined is limited only by the number of available workareas. The parent table for the form may have several one-to-many relationships and each table may have one-to-many relationships to other table fields. Parent records do not need to be included in the form, therefore you may build forms which contain only table fields. If a screen is to contain a parent record and a child table, the child table must be placed below the first accessible parent field.

Once you have defined all the necessary information in the DEFINE TABLES command, you need to identify the @...GET statements that belong to each table. The syntax for each @...GET belonging to a table field is:

TABLE_NAME!ALIAS->FIELDNAME

Fields included in table fields bring with them the rules defined in their respective Applications Data Dictionary settings. For example, a choice list defined for a field in the Applications Data Dictionary is still accessible from within a table field. Fields belonging to the same table are all placed on the same row.

Users may move to any of the TABLES on the screen by pressing the [NAVIGATE TABLE] key and paging through the records which are displayed. Records may be appended to any of the tables by pressing the [APPEND TABLE RECORD] key. The [TABLE AUTO APPEND] toggles automatic APPEND mode each time the [RETURN] key is pressed on the last record of the table field. The [FREEZE TABLE COLUMN] key freezes the cursor within a specific column. A key usage pop-up window displays when the user presses the [TAB] key. The [ABANDON] key backs the user out of the table fields. When used with a manual READ, table fields are read-only, and can not be navigated as described above.

Table fields may also be created in the Forms Designer. See the CREATE SCREEN command for full details.

<table name>

Each table field is distinguished by a unique <table name> qualifier. The <table name> must start with a letter or underscore and can consist of letters, underscores and digits (0-9). The <table name> must not exceed ten characters.

FOR <.dbf filename>

The FOR clause specifies the name of the 'child' table, <.dbf filename>, from which the records should be displayed.

AS <expN> ROWS

The numeric expression <expN> states the number of rows, or records, to display in the TABLE.

RELATING <field> BY <key expression>

The RELATING clause specifies the relationship between the 'parent' and the 'child' tables. This is in addition to the SET RELATION command, not instead of it. The <field> is the name of the field in the 'child' table. The 'child' table must be indexed on this field and this must be the master index order. This field must also exist in the 'parent' table and is the basis of the relationship between the two tables. The <key expression> consists of the name of the 'parent' table, plus the alias pointer (-> or .), plus the name of the matching <field> in the 'parent' table.

BOXCOLOR <color>

COLOR <color>

By default, the table will be the color of FIELDS and the box will be the color of BOXES. The optional COLOR and BOXCOLOR commands specify alternate colors for the table and the box, respectively. A TITLE <expC> must be specified for the box to be displayed. Assigned colors take the form: foreground/background, and are set using the following letter codes:

Color	Attribute Code
BLACK	N or blank
BLUE	B
GREEN	G
CYAN	BG
BLANK	X
GREY	N+
RED	R
MAGENTA	RB
BROWN	GR
YELLOW	GR+
WHITE	W

PRETABLE <expC1>

POSTTABLE <expC2>

The optional PRETABLE and POSTTABLE keywords associate trigger procedures with the entry and exit of the table field. The READVAR() function is useful within trigger procedures as it returns a blank when the table is displayed, and the table name when the table is activated.. The procedure names may be specified with character expressions <expC> that return the names of valid trigger procedures.

SHADOW

The SHADOW keyword causes a shadow to display along the right and bottom edges of the box. A TITLE <expC> must be specified for the box to be displayed.

TITLE <expC3>

The TITLE option is used to specify field or column headings for the table. If the TITLE option is used, a box will be drawn around the fields and the headings. The TITLE character expression must contain column headings, separated by commas, for every field in the table.

Example

```

****
**** [Screen format file 'customer.fmt' created by Recital Version 8.0]
****

define table acc;
  for accounts;
  as 7 rows;
  title 'Pd#,Ord#,Date,Price,Received,Paid,Payment,Balance';
  posttable order_posttrig;
  relating account_no by customer->account_no
set preform to orderspreform
set postform to orderspostform
set prerecord to orders_prerec
@01,00 to 06,79
@02,01 say [Account code]
@02,18 get customer->account_no
@02,25 say [Start Date]
@02,36 get customer->start_date
@03,01 say [Customer's Name]
@03,18 get customer->title
@03,22 get customer->first_name
@03,33 get customer->initial
@03,36 get customer->last_name
@04,01 say [Street]
@04,18 get customer->street
@05,01 say [City]
@05,18 get customer->city
@05,30 say [,]
@05,31 get customer->state
@05,34 get customer->zip
@10,01 get acc!accounts->product_no
@10,06 get acc!accounts->ord_no;
  picture [ @S4X]
@10,11 get acc!accounts->ord_date
@10,22 get acc!accounts->ord_value
@10,34 get acc!accounts->rec_date
@10,45 get acc!accounts->paid_date
@10,56 get acc!accounts->paid_value
@10,68 get acc!accounts->balance
@18,00 to 20,79
@19,01 say [Total:]
@19,13 get m->m_orders;
  picture [99999999];
  when .f.
@19,22 get m->m_ord_val;
  picture [$$$$,$$9.99];
  when .f.
@19,56 get m->m_paid_val;
  picture [$$$$,$$9.99];
  when .f.
@19,68 get m->m_balance;
  picture [$$$$,$$9.99];

```

```
when .f.  
*****  
// Open demo.dbf bridge.  
// This opens the four demo tables and sets up  
// indexes, relations and format files  
use demo  
edit
```

Products

Recital Terminal Developer

DEFINE WINDOW

Class

Screen Windows

Purpose

Define the coordinates and attributes of a window

Syntax

```
DEFINE WINDOW <window-name>
FROM <expN1>, <expN2> TO <expN3>, <expN4>
[CLOSE | NOCLOSE]
[COLOR <color code> | COLOR SCHEME <color scheme>]
[COMMAND | ERROR | SYSTEM | TRACE]
[DOUBLE | NONE | PANEL]
[FILL <expC1>]
[FLOAT | NOFLOAT]
[FOOTER <expC2>]
[GROW | NOGROW]
[MINIMIZE | NOMINIMIZE]
[PROPERTIES <expC3>]
[SHADOW]
[TITLE <expC4>]
[ZOOM | NOZOOM]
```

See Also

ACTIVATE SCREEN, ACTIVATE WINDOW, CLEAR WINDOWS, DEACTIVATE WINDOW, HIDE WINDOW, MOVE WINDOW, MODIFY MEMO, RELEASE WINDOWS, RESIZE WINDOW, RESTORE WINDOW, SAVE WINDOW, SHOW WINDOW, SET COLOR, SET COMMANDWINDOW, SET ERRORWINDOW, SET STATUS, SET TRACEWINDOW, SET WINDOW OF EDIT, SET WINDOW OF MEMO, WROWS(), WCOLS(), WEXIST(), WVISIBLE(), WONTOP(), WOUTPUT()

Description

The DEFINE WINDOW command is used to specify the coordinates, and attributes for a window. A window is an area of the screen designated for output and input. There is no limit to the number of defined windows. The <window-name> is the name that will be used to identify the window when using the other WINDOW commands.

FROM <expN>,<expN> TO <expN>,<expN>

Initial display of the window is positioned with the numeric expressions <expN1-expN4>. These numeric expressions define the coordinates by row and column number for the top left and bottom right corners of the window.

CLOSE | NOCLOSE

The CLOSE | NOCLOSE keywords are included for FoxPro compatibility only.

COLOR <color code> | COLOR SCHEME <color scheme number>

The COLOR <color code> can define colors for standard characters, enhanced characters, and the frame of the window. Each of the three definitions consists of “/” separated color codes. The definitions are comma separated. Colors are assigned using the following letter codes:

Color	Attribute Letter
BLACK	N or blank
BLUE	B
GREEN	G
CYAN	BG
BLANK	X
GREY	N+
RED	R
MAGENTA	RB
BROWN	GR
YELLOW	GR+
WHITE	W

The first set of letter codes, <foreground1,background1> , assigns the standard colors. The standard colors are used for @...SAYs that display when the window is activated. The second set of letter codes, <foreground2,background2> , assigns the enhanced colors that are used for the @...GETs in the activated window. The third set of letter codes, <foreground3,background3> , assigns colors for the frame, or border of the window. The default colors for window frames, @...SAYs, and @...GETs, are assigned with the SET COLOR command. When assigning color codes to the window, you may omit any of the three sets of codes by putting a comma in its place.

COMMAND | ERROR | SYSTEM | TRACE

The COMMAND keyword designates the window as a command window. Command windows contain the interactive command prompt, and when active, allow the input of commands. If the TITLE <expC> clause is not specified, command windows are labeled "Recital - command window." The SET COMMANDWINDOW command must be ON in order for command windows to operate.

The ERROR keyword designates the window as an error window. Error windows are automatically activated when an error occurs. Error windows display the appropriate error message, and a button labeled "Confirm." Pressing the [RETURN] key deactivates the error window. If the TITLE <expC> clause is not specified, error windows are labeled "Recital - error window." The SET ERRORWINDOW command must be ON in order for error windows to operate.

The SYSTEM clause will define a window with the same characteristics as the Application Workbench system window.

The TRACE keyword designates the window as a trace window. Trace windows display each executing line of a currently running program. If the TITLE <expC> clause is not specified, trace windows are labeled with "Recital - trace window." The SET TRACEWINDOW command must be ON in order for trace windows to operate.

DOUBLE | NONE | PANEL

The DOUBLE keyword is used to border the window with a double rather than a single line. The PANEL keyword is used to draw the window border in reverse video. The NONE keyword is used to display the window without a border.

FILL <expC1>

The FILL clause will fill the background of the window with the <expC1>.

FLOAT | NOFLOAT

The FLOAT keyword enables the window to move on the screen. The MOVE WINDOW command specifies coordinates for moving a window. Windows may be moved to different coordinates, or moved from their current position by specified ordinates. Windows defined with the NOFLOAT keyword cannot be moved.

FOOTER <expC2>

The FOOTER <expC2> clause is included for language compatibility only.

GROW | NOGROW

The GROW keyword enables the window size to be changed. Windows may be resized to different coordinates, or resized based on specified ordinates using the RESIZE WINDOW command. A window defined with the NOGROW keyword cannot be resized.

MINIMIZE | NOMINIMIZE

The CLOSE | NOCLOSE keywords are included for FoxPro compatibility only.

PROPERTIES <expC3>

The DEFINE WINDOW command can include the optional PROPERTIES clause to configure the following properties of the window:

Property	Description
backColor	A valid color name
foreColor	A valid color name
borderStyle	Raised, Recessed, EtchedOut, EtchedIn, GroupBox, Solid
text	Title text for a groupBox
pixelX	Pixel X position of window
pixelY	Pixel Y position of window
pixelWidth	Pixel Width of window
pixelHeight	Pixel Height of window

The <expC3> consists of a character string containing semi-colon (;) separated property=value pairs, e.g.

“backcolor=black;forecolor=red;borderStyle=GroupBox;text="Window"”

The PROPERTIES clause only has effect when running under Recital Mirage.

SHADOW

The SHADOW keyword causes a shadow to appear along the right and bottom edges of the window. The color of the shadow can be specified with the SET COLOR command.

TITLE <expC4>

The TITLE <expC4> clause centers and highlights the specified character expression on the top line of the window. If the character expression is longer than the specified width of the window, the title is truncated on the right side.

ZOOM | NOZOOM

The ZOOM | NOZOOM keywords are included for language compatibility only.

A defined window is displayed to the screen with the ACTIVATE WINDOW or SHOW WINDOW commands. The SHOW WINDOW command displays a window without activating it. The ACTIVATE WINDOW command displays a window and activates it. When a window is activated, all subsequent output is displayed in that window. Only one window may be activated at a time. Activating additional windows does not clear the display of previously activated windows. The DEACTIVATE WINDOW command clears the display of activated windows, but leaves the window definition in memory. The RELEASE WINDOW command clears both the display and the definition of windows from memory. The SAVE and RESTORE WINDOW commands may be used to keep window definitions in a file that can be reused at any time.

Example

```
define window browse;  
  from 2,2 to 12,43;  
  title "BROWSE";  
  color n/bg;  
  float;  
  grow;  
  shadow
```

Products

Recital Mirage Server, Recital Terminal Developer

DELETE

Class

Fields and Records

Purpose

Mark records in the active table for deletion

Syntax

```
DELETE [<scope>]  
[FOR <condition>]  
[WHILE <condition>]
```

See Also

ERASE, RECALL, PACK, ZAP, SET FILTER, SET DELETED

Description

The DELETE command marks records in the active table for deletion. The default <scope> deletes the current record. If the FOR clause is specified, the default scope is ALL. If the WHILE clause is specified, the default scope is REST.

If SET FILTER TO <condition> is in effect, then only those records that satisfy the filter are deleted. If a record is already marked for deletion then the <scope> will be extended. Any records which have been marked for deletion with the DELETE command can be reinstated using the RECALL command.

Records that are marked for deletion are only physically removed from the table after the PACK command has been issued. If the table is indexed, then the Recital/4GL will process the records in the table in the order of the master index file. If the active table is shareable, then the Recital/4GL will automatically lock each record in turn, mark it for deletion if required, then unlock the record.

At the end of a DELETE operation with a FOR or WHILE condition, the record pointer is positioned to EOF() if SET COMPATIBLE TO <XBASE> is in effect.

FOR <condition>

If the FOR clause is specified, then only those records which satisfy the <condition> are deleted. If FOR and WHILE clauses are used at the same time, the default <scope> is ALL.

WHILE <condition>

The WHILE clause can be used to restrict the number of records checked against a particular FOR <condition>, therefore optimizing the deletion process.

Example

```
use patrons index names, events  
delete all for event = "HAMLET"
```

```
use patrons index events, names  
seek "OPERA"  
delete rest;  
  for date<date();  
  while event = "OPERA"
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

DELETE FILE

Class

Disk and File Utilities

Purpose

Delete specified file

Syntax

DELETE FILE <filename>

See Also

SET SAFETY, ERASE

Description

The DELETE FILE command deletes the specified file, <filename> . If no directory specification is present in the <filename>, then the file is deleted from the current directory. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. The DELETE FILE command does not follow the directory search path specified with the SET PATH command. On OpenVMS, all versions of the file will be deleted.

Example

delete file orders.dbf

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

DELETE TAG

Class

Indexing

Purpose

Delete an index tag

Syntax

DELETE TAG <tagname>

[OF <.dbx filename>]

[, <tagname2 [OF <.dbx filename>]...]

See Also

INDEX ON, CREATE STRUCTURE, COPY INDEXES, COPY STRUCTURE TO, MODIFY STRUCTURE, SET INDEX, USE, TAGNO(), TAGNAME()

Description

The DELETE TAG removes an index tag from a multiple index file. More than one tag may be deleted by separating each <tagname> with a comma. If the production index file contains only one tag, the file will be deleted after the tag has been removed. The table file that is associated with the <.dbx filename> file must be opened exclusively in order to delete a tag.

OF <.dbx filename>

The OF <.dbx filename> clause is used to specify the name of the .dbx file from which the tag should be deleted. By default, the production index file is searched for <tagname>. If no <.dbx filename> is specified and the tag does not exist in the production index file, the following message is displayed: “Tag not found.”.

Example

use accounts exclusive

delete tag zip

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

DESIGN

Class

Information Center

Purpose

Initiate the Recital Information Center in Design mode

Syntax

DESIGN

See Also

CATALOG(), SET CATALOG, SET TITLE

Description

The DESIGN command initiates the Recital Information Center in Design mode. The INFO command initiates the Recital Information Center with Design mode off.

The Recital Information Center is a powerful workspace, from which any type of data may be accessed, viewed, modified and organized. Within the Recital Information Center, you can organize files into catalogs that represent a single application, an application interface or development project. The rich functionality of the Recital Information Center includes a graphical-like user interface with CUA compliant components, access to Recital workspaces, object oriented design tools and connectivity to foreign databases.

Design mode allows access to CREATE and MODIFY workspaces as well as the complete range of design tools. Design mode may also be enabled with the SET DESIGN command. When in Design mode, the Recital Information Center displays a [DESIGN] key that is used to access the development workspaces. Design mode also displays the names of program and application files. When Design mode is disabled, files with '.app' or '.prg' extensions are only displayed in catalogs to which they have been added.

The SET CATALOG TO <expC> command allows the specified catalog file to be used when the Recital Information Center is activated. If a catalog with the name specified in <expC> does not exist, it will be created. SET CATALOG ON causes all newly created files to be added automatically to the currently open catalog. When SET CATALOG is OFF, the currently open catalog is closed and automatic file adding is disabled. Catalogs may also be opened from within the Recital Information Center and files added to a catalog using the appropriate menu items.

The Recital Information Center menu bar provides access to a wide range of decision support tools, application objects and work surfaces. The menu bar contains the following options:

Menu	Description
File	Catalog file tasks
Edit	Database table record options
Options	Catalog file options
Tools	Popup Calculator, Time Manager and Notepad
Customize	Environment customization toggles
Summary	Financial and statistical calculations
Organize	Database table operations
Report	Quick and Custom Report and Label options
Help	Help System, Key Help and Technical Support

The Recital Information Center contains six file panels providing vertical listings of filenames without extensions. Each panel represents a different category or file type and contains a <create> button giving access to the appropriate create worksurface.

Panel	File Types
Data	Table (.dbf), gateway (.gtw), bridge (.brg), view (.vue).
Text	Text (.txt)
Form	Screen format (.fmt)
Report	Report format (.frm) Treport (.trf)
Label	Label format (.lbl)
Program	Application (.app), program (.prg)

Example

set catalog to customer
design

Products

Recital Terminal Developer

DIALOG BOX

Class

Screen Dialogs

Purpose

Display a text message in a dialog box

Syntax

DIALOG BOX <expC>
[LABEL <expC>]

See Also

DIALOG MESSAGE, DIALOG QUERY, DIALOG FIELDS, DIALOG SCOPE, DIALOG GET, SET MESSAGE, MESSAGE, SET COLOR, SET COLOR OF DIALOGFRAME

Description

This dialog is used to display the specified text message <expC>. A “Confirm” button will also be displayed for you to acknowledge the displayed message. If SET DIALOG is OFF, the message will be displayed in the message line.

LABEL <expC>

The default title of “MESSAGE” can be replaced with an optional title <expC>, using the LABEL clause.

Example

dialog box “Transaction Completed”

Products

Recital Mirage Server, Recital Terminal Developer

DIALOG FIELDS

Class

Screen Dialogs

Purpose

Display a dialog for selecting fields from a table

Syntax

DIALOG FIELDS

[LABEL <expC>]

See Also

DIALOG MESSAGE, DIALOG QUERY, DIALOG FILES LIKE, DIALOG SCOPE, DIALOG GET, MENU FIELDS, SET COLOR, SET COLOR OF DIALOGFRAME

Description

This dialog displays a menu of field names of fields from the current table. Field descriptions rather than field names are displayed if SET DESCRIPTION is ON.

The SET COLOR and SET COLOR OF DIALOGFRAME commands may be used to specify the colors for the dialog box and frame.

The cursor and [PAGE UP], [PAGE DOWN] keys are used to navigate the menu, the [RETURN] key to select a filename. Once a filename is selected, the 'OK' and 'Cancel' buttons or the [EXIT SAVE] and [ABANDON] keys can be used to exit the dialog with or without saving.

On exit from DIALOG FIELDS, the MENUITEM() function can be used to return the selected field name as a character string. If the user pressed the 'Cancel' button or the [ABANDON] key, MENUITEM() will return a null string.

LABEL <expC>

The LABEL clause is used to change the default dialog title to the text specified in <expC>.

Example

```
use demo
dialog fields
? menuitem()
```

Products

Recital Terminal Developer

DIALOG FILES LIKE

Class

Screen Dialogs

Purpose

Display a menu of filenames in a dialog

Syntax

DIALOG FILES LIKE <skeleton>

[LABEL <expC>]

[TRIM]

See Also

DIALOG MESSAGE, DIALOG QUERY, DIALOG FIELDS, DIALOG SCOPE, DIALOG GET, MENU FILES, ADIR(), DIR, SET COLOR, SET COLOR OF DIALOGFRAME

Description

The DIALOG FILES LIKE command displays a menu listing the files matching the pattern <skeleton>. The menu is centered on the screen and the screen is automatically saved and restored. The following ‘wild card’ characters may be used

Character	Description
?	Matches any one character.
%	Matches any one character.
*	Matches zero or more characters.

The default title of “Files” can be replaced with the specified <expC>, using the LABEL clause. If the optional TRIM clause is used, the filenames are displayed without file extensions. The SET COLOR and SET COLOR OF DIALOGFRAME commands may be used to specify the colors for the dialog box and frame.

The cursor and [PAGE UP], [PAGE DOWN] keys are used to navigate the menu, the [RETURN] key to select a filename. Once a filename is selected, the ‘OK’ and ‘Cancel’ buttons or the [EXIT SAVE] and [ABANDON] keys can be used to exit the dialog with or without saving.

On exit from DIALOG FILES LIKE, the MENUITEM() function can be used to return the selected filename as a character string. If the user pressed the ‘Cancel’ button or the [ABANDON] key, MENUITEM() will return a null string.

Example

```
dialog files like *.frm trim label "Reports"
if not empty(menuitem())
    store menuitem() to rpt
    report form &rpt to print
else
    dialog box "No Report chosen"
endif
```

Products

Recital Terminal Developer

DIALOG GET

Class

Screen Dialogs

Purpose

Display a dialog prompting for user input

Syntax

DIALOG GET <character memvar>

[HELP <expC1>]

[LABEL <expC2>]

[PICTURE <expC3>]

[TITLE <expC4>]

See Also

@...GET, DIALOG MESSAGE, DIALOG QUERY, DIALOG FIELDS, DIALOG SCOPE, SET COLOR, SET COLOR OF DIALOGFRAME

Description

The DIALOG GET command prompts for user input using a dialog box. The box contains confirmation buttons labeled “OK” and “CANCEL”. The screen is automatically saved and restored when the DIALOG GET command is used.

The SET COLOR and SET COLOR OF DIALOGFRAME commands may be used to specify the colors for the dialog box and frame.

If the <character memvar> does not exist, DIALOG GET will initialize it as a character string. If the memory variable does exist and is of character data type, its value will be retained. If it does exist, but is of any other data type, it will be changed to an empty character string with a length of 80 characters. The user enters the required value in the editing region then presses the [RETURN] key. Confirmation buttons, “OK” and “Cancel” are then highlighted. The [EXIT/SAVE] and [ABANDON] keys can also be used to save and exit or cancel and exit. The [CURSOR UP] key can be used to return to the editing region.

On exit from the DIALOG GET, the <character memvar> will contain the new value. If the [ABANDON] key or the “Cancel” button was pressed, <character memvar> will return an empty string.

HELP <expC1>

The HELP clause will display the text message <expC1> in the system message line.

LABEL <expC2>

The LABEL clause specifies the input prompt <expC2> for the dialog. The character expression can be up to 15 characters long and replaces the default prompt, “Enter Value”.

PICTURE <expC>

The optional PICTURE clause is used to create a picture-editing template, see @...GET for more details.

TITLE <expC>

The TITLE clause will replace the default title, “INPUT DATA” in the GET dialog with <expC4>. The <expC4> should be no longer than 30 characters, or it will display outside the boundaries of the box.

Example

dialog get m_var

Products

Recital Mirage Server, Recital Terminal Developer

DIALOG MESSAGE

Class

Screen Dialogs

Purpose

Display a text message in a dialog box

Syntax

DIALOG MESSAGE <expC>
[DEFAULT <expN>]

See Also

@...SAY, DIALOG QUERY, DIALOG FIELDS, DIALOG SCOPE, DIALOG GET, SET COLOR, SET COLOR OF DIALOGFRAME, SET MESSAGE, MESSAGE, LASTKEY()

Description

This DIALOG MESSAGE command is used to display the specified text message <expC> in a dialog box centered on the screen. The screen is automatically saved and restored when using the DIALOG MESSAGE command. The SET COLOR and SET COLOR OF DIALOGFRAME commands may be used to specify the colors for the dialog box and frame.

The maximum width of <expC> is 79 characters.

“Yes” and “No” buttons will also be displayed for the user to respond the displayed message. The LASTKEY() function will return 89, which is the ASCII character “Y” if the “Y” or “y” keys were pressed or if the [RETURN] key was pressed on the “Yes” button.

If SET DIALOG is OFF, the message will be displayed in the message line.

DEFAULT <expN>

The DEFAULT clause can be used to set the default button. The default button will be activated when the user hits the [RETURN] key. If DEFAULT 1 is included, or if no DEFAULT clause is included, the ‘No’ button will be the default. If DEFAULT 2 is included, the ‘Yes’ button will be the default.

Example

```
dialog message “Do you want to continue?” default 2
if lastkey() = 89
    //... continue
else
    return
endif
```

Products

Recital Mirage Server, Recital Terminal Developer

DIALOG QUERY

Class

Screen Dialogs

Purpose

Display a query dialog for interactive query creation

Syntax

DIALOG QUERY [LOCK]

See Also

MENU QUERY, DIALOG FIELDS, DIALOG SCOPE, DIALOG GET, SET COLOR, SET COLOR OF DIALOGFRAME, MENU FIELDS, MENU SCOPE, MENUITEM(), SEEK, FIND, FOUND(), LASTKEY()

Description

The DIALOG QUERY command displays a dialog used for performing queries on the active table. If the active table is indexes, the dialog will contain two prompts, one labeled KEY and the other labeled WHERE. If the table is not indexed, on the WHERE prompt is displayed. The screen is automatically saved and restored when DIALOG QUERY is used.

The SET COLOR and SET COLOR OF DIALOGFRAME commands may be used to specify the colors for the dialog box and frame.

LOCK

If the LOCK clause is present, the located record will be placed into update mode if a lock can be placed.

To specify a search key for the master index, place the cursor on the KEY field and enter in the value followed by the [RETURN] key.

To specify a WHERE condition, place the cursor on the WHERE field and enter in the boolean condition. The [HELP] key can be pressed for popup query menus allowing the query to be constructed through menu selections (see MENU QUERY for more information). The menus contain choice lists of fields, operators and connectors. If an invalid query condition is entered in the WHERE field, an error message will be displayed on exit from the DIALOG QUERY.

There are four dialog buttons in the DIALOG QUERY: "OK", "CANCEL", "LOAD" and "SAVE". A dialog button can be selected by moving the cursor onto the desired button and pressing the [RETURN] key. The [CURSOR UP] key moves the cursor back into the DIALOG QUERY to change the existing query. The [ABANDON] and [EXIT/SAVE] keys may also be used to exit from the DIALOG QUERY.

If the "SAVE" button is selected, then the SAVE QUERY dialog is displayed, containing two input fields. The file name for the query to be saved must be entered into the FILE field. A descriptive name for the file can be entered into the TITLE field. If the [EXIT/SAVE] key is pressed or the "OK" button is selected, then the query will be saved with a '.qry' extension.

If the "LOAD" button is selected, the LOAD QUERY dialog is displayed, containing a list of available queries. The previously saved query files are displayed along with their titles. The [PAGE UP], [PAGE DOWN] and cursor keys can be used to navigate the list. The [RETURN] key is used to select the highlighted query. The "OK" and "CANCEL" buttons are then selectable. If "OK" is selected, the query is loaded into the main dialog for selection or modification.

On exit from the DIALOG QUERY, the query will be performed and the record pointer moved to the located record if the query is successful. If the query fails, the record pointer remains at its original position. If the query is successful, the LASTKEY() function will return 89 (ASCII "Y").

Example

```
use patrons index names, events
dialog query
if lastkey() = 89
    edit
endif
```

Products

Recital Terminal Developer

DIALOG SCOPE

Class

Screen Dialogs

Purpose

Display a dialog for selecting a transaction scope

Syntax

DIALOG SCOPE

See Also

MENU SCOPE, MENU QUERY, DIALOG FIELDS, DIALOG GET, SET COLOR, SET COLOR OF DIALOGFRAME, MENU FIELDS, MENUITEM()

Description

The DIALOG SCOPE command displays a menu containing possible record selection scopes. The menu is framed and labeled “Scope” and has “OK” and “CANCEL” confirmation buttons.

The SET COLOR and SET COLOR OF DIALOGFRAME commands may be used to specify the colors for the dialog box and frame.

The user selects the scope by moving the highlight bar with the [PAGE UP], [PAGE DOWN] and cursor keys to the desired record scope, then pressing the [RETURN] key.. Confirmation buttons, “OK” and “Cancel” are then highlighted. The [EXIT/SAVE] and [ABANDON] keys can also be used to save and exit or cancel and exit. The [CURSOR UP] key can be used to return to the scope menu.

On exit from DIALOG SCOPE, the MENUITEM() function will return the selected record scope as a character string. If the [ABANDON] key was pressed or the “CANCEL” button selected, MENUITEM() will return a null string.

Example

```
dialog scope
store menuitem() to m_scope
report form patrons &m_scope to print
```

Products

Recital Terminal Developer

DIMENSION

Class

Array Processing

Purpose

Declare an array

Syntax

```
DIMENSION <memvar>
DIMENSION <array>[<expN>] | (<expN>)
DIMENSION <array>[<expN1>,<expN2>] | (<expN1>,<expN2>)
```

See Also

APPEND FROM ARRAY, COPY TO ARRAY, DECLARE, GATHER, PRIVATE, PUBLIC, RELEASE, RESTORE FROM, SAVE TO, SCATTER, AVERAGE(), ACHOICE(), ACOPY(), ADEL(), ADIR(), AFIELDS(), AFILL(), AINS(), ALLEN(), AMAX(), AMIN(), ASORT(), ASUM()

Description

The DIMENSION command is synonymous with the DECLARE command and is used to declare memory variables and fixed one or two-dimensional arrays. Memory variables declared using the DIMENSION command are initialized to .F. and are private to the declaring procedure.

The size of an array is fixed at its declaration. Both one and two dimensional arrays subscripts can be referenced using square brackets or round brackets.

[<expN>] | (<expN>)

For one-dimensional arrays, [<expN>] specifies the total number of elements in the array. Elements are subsequently referenced using the notation <array>[<expN>] or <array>(<expN>).

[<expN1>,<expN2>] | (<expN1>,<expN2>)

For two-dimensional arrays, <expN1> represents the number of rows and <expN2> represents the number of columns in the array. Elements are subsequently referenced by <array>[<expN1>,<expN2>] or <array>(<expN1>,<expN2>). The elements of a two dimensional array can also be referenced, as if the array were one dimensional, using <array>[<expN1>] or <array>(<expN>).

Arrays can be declared as any size. Values are assigned into arrays using the '=' operator. Arrays can then be used in a similar way to memory variables. Complete arrays can be initialized with one assignment. Array references start at 1,1 for two-dimensional arrays, and 1 for one-dimensional arrays. They can also be declared as private or public by using the PRIVATE and PUBLIC commands. Arrays can be saved to memory files with the SAVE TO command and then later restored with the RESTORE FROM command.

Notes: The brackets shown for this command do not indicate optional expressions but are a necessary part of the syntax.

Example

```
// Declare one-dimensional array of 4000 elements
dimension aTable[4000]
// Assign 0 to all elements
aTable = 0
// Insert individual element values into array
aTable[1] = 10
aTable[2] = "Hello"
aTable[3] = "World"
```

```
aTable[4] = date()  
// Print value of element 2  
? aTable[2]
```

Hello

```
// Another example  
dimension twodim[3,3]  
twodim[2,3] = "hello world"  
? twodim[6]
```

hello world

```
// Another example  
use payroll  
dimension aPayroll[reccount(), fcount()]  
copy to array aPayroll for city = "LONDON"
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

DIR

Class

Disk and File Utilities

Purpose

Display a directory of files

Syntax

DIR [<skeleton>]

See Also

!, DECRYPT, DISPLAY FILES, ENCRYPT, LIST FILES, RUN, ADIR(), DIR(), SET FILETYPE

Description

The DIR command displays filenames in the current directory and path (see SET PATH) matching the specified <skeleton>. The total number of files matching the <skeleton>, their total size and the remaining drive space in Megabytes are also displayed.

If DIR is issued with no <skeleton> specified, then it will list details of the table files only. The table file listing includes the following information:

- File type if not Recital, e.g. (FP2) for FoxPro table
- DES3 Encryption status, i.e. (DES3) if table is encrypted
- Number of Records
- Last Update date
- Size in bytes
- Dictionary status, i.e. Yes or None
- Triggers status, i.e. Yes or None
- Security status, i.e. Yes or None
- Total size of tables in Megabytes
- Number of tables
- Drive space remaining in Megabytes

Please see the CREATE command for information on the Dictionary, Triggers and Security.

The DIR command is a synonym of the DISPLAY FILES command. The DIR | DISPLAY FILES commands will pause at each page display, waiting for a key press. For listings without user intervention, please use the LIST FILES command.

Example

```
dir
dir *.prg
```

Products

Recital Terminal Developer

DISPLAY

Class

Input/Output

Purpose

Display the contents of the active table and any related tables

Syntax

DISPLAY [<scope>]
[FIELDS <field list>|<exp list>]
[FOR <condition>]
[HEADING]
[OFF]
[TO FILE <.txt filename> | (<expC>)]
[TO PRINT]
[WHILE <condition>]

See Also

DISPLAY STATUS, DISPLAY MEMORY, DISPLAY STRUCTURE, LIST, DIR

Description

The DISPLAY command is a general purpose Recital/4GL query command that retrieves and displays the contents of table files on the screen. The DISPLAY command pauses every 17 lines until a key is pressed to continue displaying, or the [ABANDON] key to cancel. When displaying a record that is longer than the screen width, the contents to the right of the display normally will not be displayed unless you have set your terminal to wrap. Consult the relevant manual for your terminal regarding this feature. Where the output is sent to a file or printer, the pause is disabled.

DISPLAY is more powerful than it looks initially. The expressions that you specify can be any valid Recital/4GL expression, including the use of alias pointers into other workareas. If you have SET RELATION TO another table, for each record that is read from the active table, the related table will have its record pointer positioned, and the appropriate record read into its workarea.

If SET FILTER TO <condition> is in effect, only those records that satisfy the filter <condition> will be displayed. If SET DESCRIPTIONS and SET HEADING are ON and the FIELDS clause is specified, the table field descriptions will be used as the headings to each of the display columns, rather than the field names. The command SET HEADING TO SINGLE | DOUBLE | NONE controls the underlining of the column headings.

Keyword	Description
<scope>	If the <scope> is not specified, only the current record will be displayed, unless the WHILE clause is used, in which case the <scope> will default to REST..
FOR <condition>	Only those records that satisfy the <condition> are displayed.
OFF	Disables the display of the record number in the first column of the results.
FIELDS <exp list>	Restricts the fields displayed to those specified.
HEADING	A heading corresponding to either the field names or the expression will be displayed above each column even if SET HEADING is OFF.
TO <file>	The display output will be sent to the specified file. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then ".txt" will be used. The command SET PAGELength governs the output file pagination and SET PAGEWIDTH defines the width of each page. Page numbers are centered on the bottom of the page according to width.

TO PRINT	The display output will be sent to a printer. The TO PRINT option will default to a local printer unless the command SET PRINTER TO \\SPOOLER is issued. The print request will then be spooled to the system printer, which is defined by the environment variable DB_PRINT.
WHILE <condition>	The <scope> defaults to REST and records are displayed until the <condition> becomes false.

Example

use patrons index events, names

display all fields name, event for event = "BALLET"

seek "OPERA"

display rest name, event, seats, price, seats * price while event = "OPERA"

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

DISPLAY DATABASE

Class

Databases

Purpose

Display information about the active database

Syntax

DISPLAY DATABASE

[TO FILE <.txt filename> | (<expC>)]

[TO PRINT[ER]]

See Also

ALTER TABLE, ALTER INDEX, CLOSE DATABASES, CLOSE TABLES, COMPILE DATABASE, CREATE DATABASE, CREATE INDEX, CREATE TABLE, CREATE VIEW, DISPLAY INDEXES, DISPLAY TABLES, DROP DATABASE, DROP INDEX, DROP TABLE, LIST DATABASE, LIST INDEXES, LIST TABLES, OPEN DATABASE, USE, SET EXCLUSIVE, SET SQL, ADATABASES(), DBUSED(), GETENV()

Description

The DISPLAY DATABASE command is used to display information about the currently active database. Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directories are sub-directories of the Recital data directory. The environment variable / symbol DB_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality. Databases are opened using the OPEN DATABASE command.

The DISPLAY DATABASE command displays the following information:

- Database Name, e.g. southwind
- Database Path, e.g. /usr/recital/data/southwind

and for each table in the database the equivalent of DISPLAY STRUCTURE INDEX followed by DISPLAY DICTIONARY:

- Table file name
- Number of records
- Date of creation
- Date of last update
- Encryption status
- Field names, types, sizes and description
- Total record length
- Production DBX file name
- Index tag names, keys, types and lengths
- Dictionary information

DISPLAY commands differ from LIST commands in that they pause every 17 lines until a key is pressed. You can cancel any further output at this point by pressing the [ABANDON] key. Where the output is sent to a file or printer, the pause is disabled.

Keyword	Description
TO <file>	The display output will be sent to the specified file. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then “.txt” will be used. The command SET PAGELength governs the output file pagination and SET PAGEWIDTH defines the width of each page. Page numbers are centered on the bottom of the page according to width.
TO PRINT	The display output will be sent to a printer. The TO PRINT option will default to a local printer unless the command SET PRINTER TO \\SPOOLER is issued. The print request will then be spooled to the system printer, which is defined by the environment variable DB_PRINT.

Example

```
set sql to vfp
open database southwind
display database to file sw_info
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

DISPLAY DICTIONARY

Class

Environment

Purpose

Display the currently active dictionary

Syntax

DISPLAY DICTIONARY

[TO FILE <.txt filename> | (<expC>)]

[TO PRINT]

See Also

CREATE, LIST DICTIONARY, SET PRINTER

Description

The DISPLAY DICTIONARY command displays the currently active dictionary. DISPLAY commands differ from LIST commands in that they pause every 17 lines until a key is pressed. You can cancel any further output at this point by pressing the [ABANDON] key. Where the output is sent to a file or printer, the pause is disabled.

Keyword	Description
TO <file>	The display output will be sent to the specified file. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then “.txt” will be used. The command SET PAGELength governs the output file pagination and SET PAGEWIDTH defines the width of each page. Page numbers are centered on the bottom of the page according to width.
TO PRINT	The display output will be sent to a printer. The TO PRINT option will default to a local printer unless the command SET PRINTER TO \\SPOOLER is issued. The print request will then be spooled to the system printer, which is defined by the environment variable DB_PRINT.

Example

use demo

display dictionary

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

DISPLAY FILES

Class

Disk and File Utilities

Purpose

Display a directory of files

Syntax

DISPLAY FILES [<skeleton>]

[TO FILE <.txt filename> | (<expC>)]

[TO PRINT]

See Also

DIR

Description

The DISPLAY FILES command displays filenames in the current directory and path (see SET PATH) matching the specified <skeleton>. The following ‘wild card’ characters may be used

Character	Description
?	Matches any one character.
%	Matches any one character.
*	Matches zero or more characters.

If DISPLAY FILES is issued with no <skeleton> specified, then it will list details of table files only.

DISPLAY commands differ from LIST commands in that they pause every 17 lines until a key is pressed. You can cancel any further output at this point by pressing the [ABANDON] key. Where the output is sent to a file or printer, the pause is disabled.

Keyword	Description
TO <file>	The display output will be sent to the specified file. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then “.txt” will be used. The command SET PAGELENGTH governs the output file pagination and SET PAGEWIDTH defines the width of each page. Page numbers are centered on the bottom of the page according to width.
TO PRINT	The display output will be sent to a printer. The TO PRINT option will default to a local printer unless the command SET PRINTER TO \\SPOOLER is issued. The print request will then be spooled to the system printer, which is defined by the environment variable DB_PRINT.

Example

```
set printer to \\spooler
display files *.prg to print
set printer to
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

DISPLAY HISTORY

Class

Error Handling and Debugging

Purpose

Display a list of previously entered commands

Syntax

DISPLAY HISTORY

[LAST <expN>]

[TO FILE <.txt filename> | (<expC>)]

[TO PRINT]

See Also

!, ALIAS, LIST HISTORY, SET HISTORY

Description

The DISPLAY HISTORY command displays a list of commands currently held in the command history. When SET HISTORY is ON, all commands entered in interactive command mode are stored in a command history list. The SET HISTORY TO <expN> command can be used to specify the size of the history list. If SET DOHISTORY is also ON, then commands executed in program files are also stored in the command history. DISPLAY differs from the LIST command in that it pauses every 17 lines until a key is pressed. You can cancel any further output at this point by pressing the [ABANDON] key. Where the output is sent to a file or printer, the pause is disabled.

Keyword	Description
LAST <expN>	Displays the last <expN> of previously entered commands
TO <file>	The display output will be sent to the specified file. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then “.txt” will be used. The command SET PAGELENGTH governs the output file pagination and SET PAGEWIDTH defines the width of each page. Page numbers are centered on the bottom of the page according to width.
TO PRINT	The display output will be sent to a printer. The TO PRINT option will default to a local printer unless the command SET PRINTER TO \\SPOOLER is issued. The print request will then be spooled to the system printer, which is defined by the environment variable DB_PRINT.

Example

```
set history on
dir
use payroll index events, names
display history
```

1 dir

2 use payroll index events, names

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

DISPLAY INDEXES

Class

Indexing

Purpose

Display index information about the current table

Syntax

DISPLAY INDEXES

[TO FILE <.txt filename> | (<expC>)]

[TO PRINT]

See Also

FIND, CLOSE INDEX, COPY INDEXES, COPY TAG, CREATE VIEW, DELETE TAG, DBXDESCEND(), DESCEND(), DTOS(), LOOKUP(), MDX(), LTOS(), REINDEX, RLOOKUP(), SEEK(), SEEK, SET INDEX, SET ORDER TO, STR(), STRZERO(), SYS(), TAG(), TAGCOUNT(), TAGNO(), USE

Description

The DISPLAY INDEXES command is used to display index information about the currently active table. Information is displayed for both production and single index files.

The DISPLAY INDEXES command displays the following information:

- Production DBX file name

and for each tag:

- Tag name
- Key
- Type
- Length

and for each open single index:

- Index file name
- Key
- Index cache size

The master index tag is flagged as such.

DISPLAY commands differ from LIST commands in that they pause every 17 lines until a key is pressed. You can cancel any further output at this point by pressing the [ABANDON] key. Where the output is sent to a file or printer, the pause is disabled.

Keyword	Description
TO <file>	The display output will be sent to the specified file. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then “.txt” will be used. The command SET PAGELength governs the output file pagination and SET PAGEWIDTH defines the width of each page. Page numbers are centered on the bottom of the page according to width.
TO PRINT	The display output will be sent to a printer. The TO PRINT option will default to a local printer unless the command SET PRINTER TO \\SPOOLER is issued. The print request will then be spooled to the system printer, which is defined by the environment variable DB_PRINT.

Example

```
// Recital/4GL
use example
display indexes to file ind_info
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

DISPLAY MEMORY

Class

Environment

Purpose

Display the contents of the current memory variables

Syntax

DISPLAY MEMORY

[LIKE <skeleton>]

[TO FILE <.txt filename> | (<expC>)]

[TO PRINT]

See Also

ALIAS, DISPLAY STATUS, RELEASE, SAVE, RESTORE, PUBLIC, PRIVATE, DECLARE

Description

The DISPLAY MEMORY command displays the contents of the memory variables and array elements currently defined. Subject to the available system memory, there is no limit to the number of memory variables that can be declared in the Recital/4GL or to the amount of memory that can be used for memory variables. DISPLAY differs from the LIST command in that it pauses every 17 lines until a key is pressed. You can cancel any further output at this point by pressing the [ABANDON] key. Where the output is sent to a file or printer, the pause is disabled.

Keyword	Description
LIKE <skeleton>	Displays all the current memory variables that match the wildcard <skeleton> specification.
TO <file>	The display output will be sent to the specified file. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then “.txt” will be used. The command SET PAGELENGTH governs the output file pagination and SET PAGEWIDTH defines the width of each page. Page numbers are centered on the bottom of the page according to width.
TO PRINT	The display output will be sent to a printer. The TO PRINT option will default to a local printer unless the command SET PRINTER TO \\SPOOLER is issued. The print request will then be spooled to the system printer, which is defined by the environment variable DB_PRINT.

Example

display memory

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

DISPLAY PROCEDURE

Class

Applications

Purpose

Display the currently active procedures and functions

Syntax

DISPLAY PROCEDURE

[TO FILE <.txt filename> | (<expC>)]

[TO PRINT]

See Also

SET PROCEDURE TO, DISPLAY STATUS

Description

The DISPLAY PROCEDURE command displays on screen the currently active procedures and functions. DISPLAY commands differ from LIST commands in that they pause every 17 lines until a key is pressed. You can cancel any further output at this point by pressing the [ABANDON] key. Where the output is sent to a file or printer, the pause is disabled.

Keyword	Description
TO <file>	The display output will be sent to the specified file. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then “.txt” will be used. The command SET PAGELENGTH governs the output file pagination and SET PAGEWIDTH defines the width of each page. Page numbers are centered on the bottom of the page according to width.
TO PRINT	The display output will be sent to a printer. The TO PRINT option will default to a local printer unless the command SET PRINTER TO \\SPOOLER is issued. The print request will then be spooled to the system printer, which is defined by the environment variable DB_PRINT.

Example

set procedure to yourlib

display procedure

display procedure to print

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

DISPLAY PROTECTION

Class

Table Basics

Purpose

Display current protection and security settings

Syntax

DISPLAY PROTECTION

[TO FILE <.txt filename> | <expC>]

[TO PRINT]

See Also

CREATE, LIST PROTECTION, STR(), GETGID(), GETPID(), GETUID()

Description

The DISPLAY PROTECTION command is used to display to the screen protection and security access control strings (ACS) for the currently active table. An access control string is a range of user identification codes used to allow groups or individuals to perform certain table operations. Access control strings are specified in the CREATE or MODIFY STRUCTURE work surface under the <SECURITY> and <PROTECTION> menu options.

DISPLAY commands differ from LIST commands in that they pause every 17 lines until a key is pressed. You can cancel any further output at this point by pressing the [ABANDON] key. Where the output is sent to a file or printer, the pause is disabled.

Keyword	Description
TO <file>	The display output will be sent to the specified file. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then “.txt” will be used. The command SET PAGELength governs the output file pagination and SET PAGEWIDTH defines the width of each page. Page numbers are centered on the bottom of the page according to width.
TO PRINT	The display output will be sent to a printer. The TO PRINT option will default to a local printer unless the command SET PRINTER TO \\SPOOLER is issued. The print request will then be spooled to the system printer, which is defined by the environment variable DB_PRINT.

Example

display protection

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

DISPLAY REPORT

Class

Reports

Purpose

Display the contents of a report format file

Syntax

```
DISPLAY REPORT <.frm filename> | (<expC>)  
[TO FILE <.txt filename> | (<expC>)]  
[TO PRINT]
```

See Also

SET PRINTER, CREATE REPORT, REPORT

Description

The DISPLAY REPORT command provides a listing of the contents of the specified report <.frm filename>. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then “.frm” is assumed. This command is primarily used in preparing system documentation.

DISPLAY commands differ from LIST commands in that they pause every 17 lines until a key is pressed. You can cancel any further output at this point by pressing the [ABANDON] key. Where the output is sent to a file or printer, the pause is disabled.

Keyword	Description
TO <file>	The display output will be sent to the specified file. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then “.txt” will be used. The command SET PAGELength governs the output file pagination and SET PAGEWIDTH defines the width of each page. Page numbers are centered on the bottom of the page according to width.
TO PRINT	The display output will be sent to a printer. The TO PRINT option will default to a local printer unless the command SET PRINTER TO \\SPOOLER is issued. The print request will then be spooled to the system printer, which is defined by the environment variable DB_PRINT.

Example

```
create report creditlist  
set printer to \\spooler  
display report creditlist to print  
set printer to
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

DISPLAY STATUS

Class

Environment

Purpose

Display the complete status of the session

Syntax

DISPLAY STATUS

[TO FILE <.txt filename> | (<expC>)]

[TO PRINT]

See Also

DISPLAY DICTIONARY, DISPLAY MEMORY, DISPLAY STRUCTURE, DISPLAY USERS, LIST STATUS

Description

The DISPLAY STATUS command displays detailed information about the status of the session, including the following:

- Active status of workareas, including indexes, locks, journals, relations, current record, number of records
- Language setting
- Printer setting
- Path setting
- Programmable function keys

DISPLAY commands differ from LIST commands in that they pause every 17 lines until a key is pressed. You can cancel any further output at this point by pressing the [ABANDON] key. Where the output is sent to a file or printer, the pause is disabled.

Keyword	Description
TO <file>	The display output will be sent to the specified file. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then “.txt” will be used. The command SET PAGELENGTH governs the output file pagination and SET PAGEWIDTH defines the width of each page. Page numbers are centered on the bottom of the page according to width.
TO PRINT	The display output will be sent to a printer. The TO PRINT option will default to a local printer unless the command SET PRINTER TO \\SPOOLER is issued. The print request will then be spooled to the system printer, which is defined by the environment variable DB_PRINT.

Example

```
set view to patrons
set printer to \\spooler
display status to print
set printer to
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

DISPLAY STRUCTURE

Class

Table Basics

Purpose

Display the structure of the active table

Syntax

DISPLAY STRUCTURE

[IN <alias>]

[INDEX]

[TO FILE <.txt filename> | (<expC>)]

[TO PRINT]

See Also

DISPLAY STATUS, DISPLAY MEMORY, CREATE, MODIFY STRUCTURE, DISPLAY DICTIONARY, DISPLAY USERS

Description

The DISPLAY STRUCTURE command displays the structure of the active table. DISPLAY commands differ from LIST commands in that they pause every 17 lines until a key is pressed. You can cancel any further output at this point by pressing the [ABANDON] key. Where the output is sent to a file or printer, the pause is disabled.

Keyword	Description
IN <alias>	The IN <alias> clause is used to display the structure of an open table in a workarea that is not currently selected. Alias names may be assigned to tables with the USE command, or default to the table basename.
INDEX	The INDEX keyword is used to display index tag information along with the structure details.
TO <file>	The display output will be sent to the specified file. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then “.txt” will be used. The command SET PAGELENGTH governs the output file pagination and SET PAGEWIDTH defines the width of each page. Page numbers are centered on the bottom of the page according to width.
TO PRINT	The display output will be sent to a printer. The TO PRINT option will default to a local printer unless the command SET PRINTER TO \\SPOOLER is issued. The print request will then be spooled to the system printer, which is defined by the environment variable DB_PRINT.

Example

use patrons
display structure

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

DISPLAY TABLES

Class

Databases

Purpose

Display table information about the active database

Syntax

DISPLAY TABLES

[TO FILE <.txt filename> | (<expC>)]

[TO PRINT[ER]]

See Also

ALTER TABLE, ALTER INDEX, CLOSE DATABASES, CLOSE TABLES, COMPILE DATABASE, CREATE DATABASE, CREATE INDEX, CREATE TABLE, CREATE VIEW, DISPLAY DATABASE, DISPLAY INDEXES, DROP DATABASE, DROP INDEX, DROP TABLE, LIST DATABASE, LIST INDEXES, LIST TABLES, OPEN DATABASE, USE, SET EXCLUSIVE, SET SQL, ADATABASES(), DBUSED(), GETENV()

Description

The DISPLAY TABLES command displays the base name and file name including the full path for each table in the currently active database.

Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directories are sub-directories of the Recital data directory. The environment variable / symbol DB_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality. Databases are opened using the OPEN DATABASE command.

DISPLAY commands differ from LIST commands in that they pause every 17 lines until a key is pressed. You can cancel any further output at this point by pressing the [ABANDON] key. Where the output is sent to a file or printer, the pause is disabled.

Keyword	Description
TO <file>	The display output will be sent to the specified file. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then “.txt” will be used. The command SET PAGELength governs the output file pagination and SET PAGEWIDTH defines the width of each page. Page numbers are centered on the bottom of the page according to width.
TO PRINT	The display output will be sent to a printer. The TO PRINT option will default to a local printer unless the command SET PRINTER TO \\SPOOLER is issued. The print request will then be spooled to the system printer, which is defined by the environment variable DB_PRINT.

Example

VFP/SQL > OPEN DATABASE southwind

VFP/SQL > DISPLAY TABLES

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

DISPLAY TRIGGERS

Class

Table Basics

Purpose

Display triggers associated with current table

Syntax

DISPLAY TRIGGERS

[TO FILE <filename>]

[TO FILE <expC>]

[TO PRINT]

See Also

CREATE, MODIFY STRUCTURE, CREATE SCREEN, MODIFY SCREEN, CREATE REPORT, MODIFY REPORT SET PREFORM TO, SET PRERECORD TO, SET POSTFORM TO, SET POSTRECORD TO, @...GET PREFIELD, @...GET POSTFIELD, LIST TRIGGERS

Description

The DISPLAY TRIGGERS command is used to display to the screen all triggers that are associated with the currently active table. A trigger is used to call a procedure written in the Recital/4GL. Accessible through the CREATE | MODIFY work surfaces, and through SET commands, triggers may be inserted at table, field, record, form, and report levels.

DISPLAY commands differ from LIST commands in that they pause every 17 lines until a key is pressed. You can cancel any further output at this point by pressing the [ABANDON] key. Where the output is sent to a file or printer, the pause is disabled.

Keyword	Description
TO <file>	The display output will be sent to the specified file. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then ".txt" will be used. The command SET PAGELENGTH governs the output file pagination and SET PAGEWIDTH defines the width of each page. Page numbers are centered on the bottom of the page according to width.
TO PRINT	The display output will be sent to a printer. The TO PRINT option will default to a local printer unless the command SET PRINTER TO \\SPOOLER is issued. The print request will then be spooled to the system printer, which is defined by the environment variable DB_PRINT.

Example

display triggers

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

DISPLAY USERS

Class

Environment

Purpose

Display all the active users

Syntax

DISPLAY USERS

[TO FILE <.txt filename> | (<expC>)]

[TO PRINT]

See Also

DISPLAY STATUS, DISPLAY MEMORY, DISPLAY DICTIONARY, DISPLAY STRUCTURE, LIST USERS

Description

The DISPLAY USERS command displays all the active systems users. DISPLAY commands differ from LIST commands in that they pause every 17 lines until a key is pressed. You can cancel any further output at this point by pressing the [ABANDON] key. Where the output is sent to a file or printer, the pause is disabled.

Keyword	Description
TO <file>	The display output will be sent to the specified file. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then “.txt” will be used. The command SET PAGELENGTH governs the output file pagination and SET PAGEWIDTH defines the width of each page. Page numbers are centered on the bottom of the page according to width.
TO PRINT	The display output will be sent to a printer. The TO PRINT option will default to a local printer unless the command SET PRINTER TO \\SPOOLER is issued. The print request will then be spooled to the system printer, which is defined by the environment variable DB_PRINT.

Example

display users

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

DO

Class

Applications

Purpose

Execute a Recital procedure, program or stored procedure

Syntax

```
DO <.prg filename> | <procedure-name> | (<expC>)  
[WITH <parameter> [,<parameter>]...]  
[WITH <array> [,<array>]...]
```

See Also

ALIAS, COMPILER, DEBUG, DO CASE, DO WHILE, LINK, MODIFY COMMAND, PARAMETERS, PROCEDURE, SET COMPATIBLE, SET MAXDBO, SET PSHARE

Description

The DO command is used to execute the Recital/4GL program <.prg filename> or procedure <procedure-name>. The filename or procedure name can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified then '.prg' is used. If the full path is not given, the file is assumed to be in the current directory. The path (SET PATH) will also be searched if the file is not found in the current directory. When a program encounters an end of file or a RETURN statement, control returns back to the calling program. Control returns back to the keyboard if the <.prg filename> was called from interactive command mode.

The Recital/4GL supports 20 DO levels unless the environment symbol DB_MAXLEV (maximum 40) is set higher. This is also dependent on Operating System limits controlling the number of open files. Program files can be linked together if required, using the LINK command or the 'dbl' Recital linker utility (Recital Terminal Developer).

WITH <parameter> | <array>

You may optionally pass parameters including arrays to the called program. Parameters passed using the DO WITH <parameters> syntax are passed by reference and can be changed by the called procedure. Procedures can also be called using function syntax procedure(parameter1,parameter2...), in which case the parameters are passed by value and cannot be changed by the called procedure. Procedures expecting parameters should have a PARAMETERS statement as the first executable statement after the procedure declaration. The number of parameters passed to a procedure can be checked using the PCOUNT() function. If CLIPPER is set ON and not all parameters are passed, the variables in the PARAMETER command not passed will be defined as undefined, type "U" instead of logical .F..

Example

```
procedure name  
parameter a,b,c  
a = b*c  
return  
*  
name(a,10,20)  
* is equivalent to  
do name with a,10,20
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

DO CASE

Class

Applications

Purpose

Multiple choice selection command

Syntax

DO CASE

CASE <condition>

[<commands>]

[CASE <condition>

[<commands>]]...

[OTHERWISE

[<commands>]]

ENDCASE

See Also

@...MENU COMMAND, DO WHILE, IF, ICASE(), IF(), IIF()

Description

The DO CASE command selects one course of action out of many alternatives. The Recital/4GL evaluates each CASE <condition> in turn. As soon as one of the conditions evaluates to true (.T.) the <commands> for that CASE are executed and any further case statements are ignored. Following execution of the <commands>, the program continues after the ENDCASE statement.

OTHERWISE

If an OTHERWISE statement is present and no CASE <condition> evaluates to .T., the OTHERWISE <commands> are executed.

ENDCASE

If no CASE <condition> is .T., and there is no OTHERWISE statement specified, then control skips to the next command following the ENDCASE.

CASE statements, as with all of the other Recital/4GL statements can be nested. In other words, a CASE statement can contain further DO CASE commands.

Example

```
accept "Enter a command: " to command
do case
  case command = "BROWSE"
    browse
  case command = "DIR"
    dir
  otherwise
    set message to "Unknown command."
endcase
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

DO WHILE

Class

Applications

Purpose

Repeat a block of commands while a specified condition is true

Syntax

```
DO WHILE <condition>
[EXIT]
[LOOP]
ENDDO
```

See Also

DO CASE, IF, DO, LOOP, EXIT, FOR...NEXT

Description

The DO WHILE command repeats the commands between the DO WHILE and the ENDDO statement, until the specified <condition> becomes .F. The maximum number of nested DO WHILE loops allowed is 16.

If the specified <condition> result is .T., then all commands within the DO WHILE loop will be executed. If the specified condition returns an .F., the Recital/4GL will skip down to the first statement following the ENDDO to continue execution.

Note: Use of redefined macros in the body of the loop is not supported.

EXIT

If an EXIT statement is encountered then the DO WHILE loop is exited.

LOOP

If a LOOP statement is encountered, then control returns to the head of the DO WHILE loop.

ENDDO

The ENDDO statement terminates the DO WHILE loop. Commands within the DO WHILE must be properly nested.

Example

```
use patrons index events, names
seek "OPERA"
do while event = "OPERA"
    display name, event, seats*price off
    skip
enddo
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

ED

Class

Terminal Developer Development Tools

Purpose

Execute a text editor to edit program files

Syntax

ED <prg filename> | (<expC>)

See Also

MODIFY COMMAND, TEXTEDIT(), VI

Description

ED provides the facility to create or modify program files and other text files. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is present in the file name, then '.prg' is used.

The default editors are: the 'vi' editor under UNIX and Linux; the 'edt' editor under VAX/VMS. You may override these defaults using the SET TEDIT TO command.

ED is a synonym of the MODIFY COMMAND and VI commands.

Example

```
modify command myprogram
ed myprogram
vi myprogram
```

Products

Recital Terminal Developer

EDIT

Class

Screen Forms

Purpose

Full screen editing of records through a form

Syntax

EDIT

[<scope>]

[FIELDS <field list>]

[FOR <condition>]

[KEY <exp>]

[NOAPPEND]

[NOCLEAR]

[NODELETE]

[NOEDIT]

[NOFOLLOW]

[NOINIT]

[NOMENU]

[NOORGANIZE]

[NOWAIT]

[WHILE <condition>]

See Also

APPEND, BROWSE, CHANGE, READ, SET FORMAT, SET UPDATE

Description

The EDIT command provides the facility to edit records in the active table that match specified record selection criteria. The EDIT command uses a default form to display and allow updating of the specified records.

When memo fields are displayed, the memo field label is in lower case if the field is empty, and upper case if the field contains text. If a memo window (SET WINDOW OF MEMO) is active, the memo can be displayed and edited in the specified window. Otherwise, pressing the [INSERT] key or the [HELP] key on the memo field will popup a notepad editor. Pressing the [HELP] key within the notepad editor displays the memo editing keys available. These keys include facilities for reading from and writing to external files and printing on the system printer.

You can design your own forms using the Forms Designer (CREATE SCREEN). The Form Designer will automatically generate a format file that can contain @...GET, @...SAY, @...MENU and other display objects. This form will be used by the EDIT command if it has been activated using the SET FORMAT command.

If the active table is shared, then automatic record locking will take place for each of the tables referenced on the form. The records are automatically unlocked when you skip to another record, or exit from the EDIT command. If UPDATE is ON then 'Upd' appears in the 3rd box of the status bar at the bottom of the screen. When UPDATE is ON records are automatically locked before they are read from the database. When UPDATE is OFF, no record locking is performed. It is recommended that you toggle UPDATE mode ON when you have a record displayed on the form which you want to update, and leave UPDATE OFF while you are browsing through your records. If SET LOCKWAIT is OFF, then whenever an attempt is made to lock a record that is already locked by another user, you are given the choice of waiting for the lock or continuing in query mode.

When UPDATE is toggled ON, the form is refreshed with the most current information from the table. As only one user at a time can be in UPDATE mode on a particular record, the information displayed on the form is always current at the time of editing. The changed data on the form is written to the table and the lock released when the [EXIT/SAVE], [NEXT SCREEN], [PREVIOUS SCREEN] or [MENUBAR] keys are pressed. If the [ABANDON] key is pressed and changes have been made to the data, SET VERIFY ON causes a dialog box to be displayed asking for confirmation.

Keyword	Description
<scope>	If no <scope> is specified, EDIT is activated on the current record and all records are accessible using the [NEXT RECORD] and [PREVIOUS RECORD] keys.
FIELDS <field list>	The active fields can be restricted to those specified in the comma separated <field list>.
FOR <condition>	Record navigation is restricted to those records that match the <condition>.
KEY <exp>	The active records can be restricted to those that match the specified <exp>. The <exp> must be based on the index key of the current master index.
NOAPPEND	The ability to append records is disabled.
NOCLEAR	Erasing of the screen on entry and exit from EDIT is disabled.
NODELETE	The ability to delete records is disabled.
NOEDIT	The ability to edit the record is disabled. Access is read only.
NOFOLLOW	The NOFOLLOW key word determines whether the record pointer follows a record whose position has changed. A record's position may change when fields that are part of the master index expression are modified. Normally, upon update, the record pointer will be moved to the new position, NOFOLLOW disables this.
NOINIT	The work surface retains the keywords and specifications from the last EDIT session.
NOMENU	The menu bar is disabled in the default EDIT form.
NOORGANIZE	Language compatibility only
NOWAIT	Control is returned to the executing program without waiting for the user to exit the EDIT worksurface.
WHILE <condition>	Record navigation is restricted to those records that match the specified <condition>. Navigation cannot continue beyond a record that does not match the <condition>.

The following keys are active in EDIT:

Key	Action
ABANDON	Discard current changes then exit from the form
CURSOR DOWN	Skip to next field
CURSOR LEFT	Skip to previous field
CURSOR RIGHT	Skip to next field
CURSOR UP	Skip to previous field
DELETE FIELD	Initialize field
DELETE RECORD	Delete / Recall the record
EDIT FIELD	Enter field edit mode
EXIT/SAVE	Write current changes then exit from the form
FIND	Find record by key or condition
FIND NEXT	Find next record matching specified key or condition
HELP	Activate pop-up help
MENUBAR	Activate the EDIT menu bar
NEXT RECORD	Write current changes then skip to next record
PREVIOUS RECORD	Write current changes then skip to next record

REFRESH	Redraw the form
TAB	Toggle function key menu on and off
UPDATE MODE	Toggle update mode on and off

If SET MOUSE is ON, cursor keys will move the cursor anywhere on the screen rather than just from field to field. If SET NAVIGATE is ON, the cursor moves to fields following the direction specified by the key being pressed, rather than following the order of the GETS on the form. When the RETURN key is pressed, the cursor moves to the nearest field when SET NAVIGATE is ON.

The following keys are active in field edit mode:

Key	Action
BACKSPACE	Delete character before cursor
CURSOR LEFT	Skip to previous character
CURSOR RIGHT	Skip to next character
DELETE CHAR	Delete character under cursor
DELETE FIELD	Delete from cursor to end of field
DELETE WORD	Delete current word
INSERT MODE	Toggle insert / overwrite mode
WORD LEFT	Skip left a word
WORD RIGHT	Skip right a word

The following menu options are available from the EDIT menu bar in the default form:

Menu Item	Action
Descriptions	Toggle the field descriptions on and off
Top	Position to the top of the table or selected records
Bottom	Position to the bottom of the table or selected records
Order	Select index order
Query	Query the table for selected records
Help	Activate on-line help system

Example

use demo
edit

Products

Recital Mirage Server, Recital Terminal Developer

EJECT

Class

Printing

Purpose

Advance to the top of the next page on the printer

Syntax

EJECT

See Also

SET PRINT, SET PRINTER, SETPRC()

Description

The EJECT command causes the printer to advance a page. You must have initially set the print head correctly at the head of the paper. The EJECT command works by sending a form feed character to the printer. If SET PRINT is OFF, the EJECT command causes SET PRINT to be temporarily set ON in order to output the form feed.

Example

```
eject  
list status to print  
eject
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

ENCRYPT

Class

DES3 Encryption

Purpose

Encrypt the specified table or tables

Syntax

ENCRYPT <.dbf> | <skeleton> KEY <expC1>

See Also

APPEND FROM, COPY FILE, COPY STRUCTURE, COPY TO, DECRYPT, DIR, USE, SET ENCRYPTION

Description

The ENCRYPT command is used to encrypt the data in the specified table, <.dbf> or tables matching the <skeleton>. If the <skeleton> syntax is used, then all matching tables will be given the same encryption key. The <expC1> must contain a three part comma-separated key. The key may optionally be enclosed in angled brackets. Each part of the key can be a maximum of 8 characters. The key is DES3 encrypted and stored in a .dkf file with the same basename as the table. After encryption, the three parts of the key must be specified correctly before the table can be accessed.

Example

```
encrypt accounts key "key1,key2,key3"  
encrypt salaries key "<key_1,key_2,key_3>"
```

```
// encrypt all .dbf files in the directory  
encrypt *.dbf key "key1,key2,key3"
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

END TRANSACTION

Class

Transaction Processing

Purpose

End transaction Before Image Journaling

Syntax

```
BEGIN TRANSACTION [<path name>]
<commands>
END TRANSACTION
```

See Also

SET ROLLBACK, ISMARKED(), RESET IN, ROLLBACK, ROLLBACK(), COMPLETED()

Description

The END TRANSACTION command is used to flag the end of a transaction for Before Image Journaling (BIJ). A ‘transaction’ consists of all file modifications that occur within the commands BEGIN TRANSACTION and END TRANSACTION.

When BEGIN TRANSACTION is issued all currently open files and all files opened between BEGIN and END TRANSACTION will have BIJ invoked automatically. If BIJ is not required on a particular table then the RESET IN command should be issued for the relevant workarea so that journaling will no longer occur in that workarea. The journals are stored in a log file (with a file extension of ‘.log’) which the Recital/4GL generates automatically. You can optionally specify the disk and directory <path name> where the log file will be stored when the BEGIN TRANSACTION command is issued.

A “rollback” causes record contents to be restored to their value before modification (i.e. at the time BEGIN TRANSACTION was issued). This is particularly useful if an error occurs during a program that modifies files.

If SET ROLLBACK is ON, the Recital/4GL will automatically execute the ROLLBACK command if an error occurs during the transaction process. Otherwise, error trapping should be handled manually using the ON ERROR command.

The COMPLETED() function can be used after the END TRANSACTION command to determine if any errors occurred during processing of the commands between BEGIN and END TRANSACTION.

Please note that the following commands are not allowed during a transaction:

```
CLEAR ALL
CLOSE ALL
CLOSE DATABASE
CLOSE INDEX
MODIFY STRUCTURE
PACK
ZAP
```

Example

```
procedure recovery
rollback
if rollback()
    dialog box "Rollback was ok."
else
```

```
        dialog box "Rollback not completed."
    endif
return

use accounts
on error do recovery
begin transaction
    delete first 15
    insert
    replace all t1 with (t2*t3)/100
    list
end transaction
if completed()
    dialog box "Transaction completed"
else
    dialog box "Errors occurred during transaction"
endif
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

ENDFUNC

Class

Applications

Purpose

Return from a function

Syntax

ENDFUNC

See Also

FUNCTION, RETURN, SET COMPATIBLE

Description

The ENDFUNC statement closes the active function, releases memory variables and arrays defined as private, and passes control back to the calling program assuming no RETURN statement has already been called.

If the function is exited using the ENDFUNC command or other implicit RETURN, the function will return .T. (true). The command SET COMPATIBLE TO VFP must be in effect to ensure Visual FoxPro compatibility.

Example

```
function example_1
dialog box [has return statement]
return .t.
//already exited function
endfunc
```

```
function example_2
dialog box [has no return statement]
endfunc
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

ENDPROC

Class

Applications

Purpose

Return from a procedure or program

Syntax

ENDPROC

See Also

LINK, PROCEDURE, RETURN

Description

The ENDPROC statement closes the active program file, releases memory variables and arrays defined as private, and passes control back to the calling program assuming no RETURN statement has already been called.

If the procedure is exited using the ENDPROC command or other implicit RETURN, the procedure will have a return value of .T. (true). The command SET COMPATIBLE TO VFP must be in effect to ensure Visual FoxPro compatibility.

Example

```
procedure example_1
dialog box [has return statement]
return
//already exited function
endproc
```

```
procedure example_2
dialog box [has no return statement]
endproc
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

ERASE

Class

Disk and File Utilities

Purpose

Delete a file or files

Syntax

ERASE <filename> | (<expC>) | <file list>

See Also

DELETE, RUN, ALIAS, SET SAFETY

Description

The ERASE command deletes the specified file or comma-separated file list. If no directory specification is present in the filename, then the file is deleted from the current directory. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. When deleting a file, ERASE does not follow the directory search path specified with the SET PATH command. On OpenVMS, all versions of the file will be deleted.

Example

```
if file("backup.old")
    erase backup.old
endif
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

ERROR

Class

Error Handling and Debugging

Purpose

Create a user defined error message

Syntax

ERROR <expN>, <expC>

See Also

ON ERROR, ERROR(), MESSAGE()

Description

The ERROR command is used to define and signal a user-defined error. When the ERROR command is executed, a run-time error occurs and if no ON ERROR procedure is present, an error.mem file will be generated. If an ON ERROR procedure is present the error procedure will be called. The ERROR() function returns the number of the last error encountered. The MESSAGE () function returns the message of the last error encountered. The ON ERROR command may be used to execute specified commands when an error occurs.

<expN>

The numeric expression <expN> is the user defined error number.

<expC>

The character expression <expC> is the error message to be associated with the error number.

Example

```
mfile = "customer"  
//...
```

```
if mfile <> "customer"  
    error 6000, "Wrong file"  
endif
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

EXIT

Class

Applications

Purpose

Force exit from a DO WHILE loop

Syntax

EXIT

See Also

DO WHILE, LOOP

Description

The EXIT command causes control to drop out of a DO ... WHILE or FOR ... NEXT loop.

Example

```
use patrons index events
seek "BALLET"
do while .not. eof()
    if event<>"BALLET"
        exit
    endif
    display name, event, seats, seats * price off
    skip
enddo
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

EXTERNAL

Class

Xbase Compatibility

Purpose

External procedure list

Syntax

EXTERNAL <procedure list>

See Also

SET COMPATIBLE

Description

The EXTERNAL command is provided for language compatibility with Xbase product only and is ignored.

Example

external myproc

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

FIND

Class

Indexing

Purpose

Search for a key in the master index file

Syntax

FIND <key expression>

See Also

INDEX ON, LOCATE, SEEK, SET EXACT, SET TALK, DBXDESCEND(), DESCEND(),
DESCENDING(), DTOS(), EOF(), FOUND(), LTOS(), STR()

Description

The FIND command searches for the specified <key expression> in the master index file. If the <key expression> is found, then the FOUND() function will return .T., and the EOF() function will return .F.. If the <key expression> is not found, then the FOUND() function will return .F., and the EOF() function will return .T.. Character expressions used as <key expressions> may include blanks.

The & macro function must be used to substitute the required <key expression> into the quoted string when the <key expression> is contained in a field or variable.

If the DESCEND() function is used to create the index key, it must also be used in the search <key expression>. Tag indexes built with the DESCENDING keyword require the use of the DBXDESCEND() function in the <key expression>. The DESCENDING() function can be used to determine whether a particular tag was built with the DESCENDING keyword.

Example

```
use patrons index events, name
m_find = "OPERA"
find &m_find
if found()
    edit
else
    dialog message "Record not found."
endif
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

FLUSH

Class

Screen Forms

Purpose

Flushes the terminal buffer

Syntax

FLUSH

See Also

SET TBUFSIZE

Description

The flush command flushes all the characters in the terminal buffer to the terminal. To optimize terminal I/O, the Recital/4GL will buffer terminal I/O until a command that requires user input is issued, or the terminal buffer fills up. The terminal buffer size is determined by the SET TBUFSIZE command.

Example

```
for i = 1 to 5000
    @ 13,1 say "Loop " + ltrim(str(i))
    flush
next
```

Products

Recital Mirage Server, Recital Terminal Developer

FOR ... NEXT

Class

Applications

Purpose

Processes a list of commands in a loop for a specified number of times

Syntax

```
FOR <memvar> = <exp1> TO <exp2>
[STEP <expN1>]
[EXIT]
[LOOP]
NEXT
```

See Also

DO WHILE

Description

The FOR ... NEXT command repeats the commands between the FOR and the NEXT statement. The <exp1> specifies the loop start point and <exp2> the loop end point. <exp1> and <exp2> may be integer or date values. The FOR...NEXT command is equivalent to a counter based DO WHILE ... ENDDO set of commands but FOR ... NEXT is faster.

STEP <expN>

If the optional STEP <expN1>, is specified, then the FOR ... NEXT loop will increment by <expN1>. This value can be a positive or negative number. If <expN1> is not specified then the FOR ... NEXT loop will increment by 1.

EXIT

The looping will continue until either <expN2> is reached or an EXIT command is encountered.

LOOP

If a LOOP command is encountered, then control returns to the start of the FOR ... NEXT loop.

Example

```
for i = 1 to 10 step 2
    ?i*2
next
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

FUNCTION

Class

Applications

Purpose

Declare a User Defined Function (UDF)

Syntax

FUNCTION <expC>[(<parameter-list>)]

See Also

ALIAS, DO, ENDFUNC, KEYWORD, LINK, PARAMETERS, PRIVATE, PROCEDURE, PUBLIC, RETURN, SET PROCEDURE

Description

A FUNCTION <expC> command is used to declare a User Defined Function (UDF). Recital/4GL UDFs can be used not only in programs, but also in the Applications Data Dictionary, the Report Writer and wherever a standard Recital/4GL function can be used.

A UDF can have a variable number of parameters passed to it. These are assigned to private variables in the optional PARAMETERS statement, which if used, must appear following the FUNCTION statement and before any other executable statement. If a PARAMETERS statement is present, then at least one parameter must be specified with the UDF call. The PCOUNT() function can be used to determine how many actual parameters were specified.

The functions in a procedure library file are made known to the Recital/4GL by using the SET PROCEDURE TO command. Functions can be included in program files, as well as in procedure library files. If a function is included in a program file, then it must be defined before it is used.

The FUNCTION command is terminated with a RETURN statement. The RETURN statement can optionally have an expression specified. If none is specified then .F. is returned. Other RETURN statements may be included in the syntax following a FUNCTION command, provided that they are properly nested between IF...ENDIF, DO WHILE...ENDDO, or DO CASE...ENDCASE. The Recital/4GL function names can be a maximum of 32 characters in length, and must begin with a letter or underscore, followed by any combination of letters (A-Z), digits (0-9), and underscores (_).

To execute a UDF, name it in an expression, followed by parameters in round brackets. It will behave just like any standard Recital/4GL function.

There is no limit to the number of functions that can be declared in the Recital/4GL. The commands LIST PROCEDURE or DISPLAY PROCEDURE will let you see all currently active functions and procedures.

If SET COMPATIBLE is set to VFP or CLIPPER5, the FUNCTION command can also include the parameter list declaration, instead of requiring a separate PARAMETERS statement. The parameters should be listed in comma-separated format within parentheses after the function name, e.g.

```
function myfunc(para1, para2, para3)
```

If SET COMPATIBLE is set to VFP, the FUNCTION declaration can be terminated with an ENDFUNC command rather than a RETURN. It will also be terminated when another PROCEDURE or FUNCTION command is reached.

Example

// Function to return product of two numbers

function atimesb

parameters a, b

result=a*b

return result

? atimesb(5,10)

50

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

GATHER

Class

Array Processing

Purpose

Replace fields with elements from an array or series of memory variables

Syntax

```
GATHER FROM <array> | MEMVAR  
[FIELDS <field list>]  
[MEMO]
```

See Also

SCATTER, COPY TO ARRAY, PRIVATE, PUBLIC, AFILL(), ADIR(), ASORT(), ALLEN(), AFIELDS(), AINS(), ADEL(), ACHOICE(), ASCAN()

Description

The GATHER command replaces fields in the current record with elements from the specified array <array> or, if the MEMVAR keyword is used, from a series of memory variables with the same names as the fields. If the current table is shared, then automatic record locking is performed for the update operation. The data types of the array elements or memory variables and corresponding fields must be compatible. If the array has fewer elements than the number of fields in the record, then fields are modified only up to the number of elements specified. If no matching memory variable is found for a field, the field is unchanged.

FIELDS <field list>

The optional FIELDS <field list> can be used to restrict the fields updated to those in the comma separated <field list>.

MEMO

By default, memo fields are ignored by the GATHER command. If the MEMO keyword is specified, memo fields will be included.

If SET LOCKTYPE TO OPTIMISTIC is active, an attempt to use the GATHER command on a record that has been modified since it was last read will generate an error.

Example

```
use demo  
goto 45  
scatter fields last_name, first_name to table_1  
append blank  
gather fields last_name, first_name from table_1
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

GENERATE

Class

Table basics

Purpose

Add new records and fill them with random information.

Syntax

GENERATE <expN>

See Also

APPEND, APPEND FROM

Description

The GENERATE command will append <expN> number of records into the currently selected table. The fields in the new records will contain random data.

Example

```
use newtable
generate 10
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

GOTO

Class

Fields and Records

Purpose

Position to a specific record in a table

Syntax

GOTO [RECORD] <expN> | BOTTOM | TOP [IN <alias>]

GO [RECORD] <expN> | BOTTOM | TOP [IN <alias>]

See Also

FIND, SEEK, LOCATE, SET RELATION, SET CLIPPER, SET PCFILTER

Description

The GOTO command is used to position the record pointer to the specified record in a table. The pointer is moved to the record whose number matches that specified in <expN>. By default, the GOTO command operates in the currently selected workarea. If the number specified in <expN> is not a valid record number, a 'Record is out of range' error message will be given.

BOTTOM

If the BOTTOM keyword is specified, then the record pointer is positioned to the last record in the table.

TOP

If the TOP keyword is specified, then the record pointer is positioned to the first record in the table.

IN <alias>

The IN <alias> clause is used to move the record pointer of an open table in another workarea.

If the command SET PCFILTER is ON, then the use of GOTO <expN> bypasses any FILTER <condition> that may be in effect with the SET FILTER TO command. The record pointer may be positioned to records that are marked for deletion, even if SET DELETED is ON. If SET CLIPPER is set ON, and GOTO RECORD <expN> is greater than the number of records in the table, the Recital/4GL does not report an error. Rather than reporting an error, the Recital/4GL sets EOF() to .T., and FOUND() to .F.. If SET RELATION TO is in effect for the active table, then defined relationships are maintained after the record pointer is positioned.

Example

```
goto top
browse
goto 10
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

HELP

Class

Menus

Purpose

On-line help menu

Syntax

HELP

See Also

SET HELP, SET HELPFILE, SET HELPWINDOW

Description

The command SET HELPFILE TO <.hlm filename> can be used to create an application specific help system. This can then be accessed using the HELP command or from the [HELP] menu bar option in worksurfaces such as APPEND, CHANGE, CREATE and EDIT.

NOTE: Recital 4GL Help is not longer provided in HELPFILE format. It is available in PDF and CHM format included in the software distributions or from the Recital web site <http://www.recital.com>.

Example

help

Products

Recital Terminal Developer

HIDE MENU

Class

Menus

Purpose

Remove a menu from the display

Syntax

HIDE MENU <expC1>[,<expC2>,...] | ALL
[SAVE]

See Also

DEFINE PAD, ON PAD, ON SELECTION PAD, ACTIVATE MENU, DEACTIVATE MENU, RELEASE MENUS, SHOW MENU, DEFINE POPUP, DEFINE BAR, ON SELECTION POPUP, ACTIVATE POPUP, DEACTIVATE POPUP, RELEASE POPUPS, CLEAR POPUPS, SHOW POPUP, SET COMPATIBLE

Description

The HIDE MENU command is included for language compatibility only.

Example

hide menu main

Products

Recital Mirage Server, Recital Terminal Developer

HIDE POPUP

Class

Menus

Purpose

Remove a popup or popups from the display

Syntax

HIDE POPUP <expC1>[,<expC2>,...]

See Also

DEACTIVATE POPUP

Description

The HIDE POPUP command removes the specified activated Xbase style popup menu(s) from the screen. The popup name or names are specified in <expC1>, <expC2>The affected pop-up menu is not released from memory when hidden and may be activated again using the ACTIVATE POPUP command. The command SET COMPATIBLE should be set ON when using the Xbase style menus.

Example

hide popup popup1

Products

Recital Mirage Server, Recital Terminal Developer

HIDE WINDOW

Class

Screen Windows

Purpose

Remove a window or group of windows from view

Syntax

HIDE WINDOW <window-name> | <window-name list> | ALL

See Also

ACTIVATE SCREEN, ACTIVATE WINDOW, CLEAR WINDOWS, DEACTIVATE WINDOW, DEFINE WINDOW, MOVE WINDOW, MODIFY MEMO, RELEASE WINDOWS, RESIZE WINDOW, RESTORE WINDOW, SAVE WINDOW, SHOW WINDOW, SET COMMANDWINDOW, SET ERRORWINDOW, SET STATUS, SET TRACEWINDOW, SET WINDOW OF EDIT, SET WINDOW OF MEMO, WROWS(), WCOLS(), WEXIST(), WVISIBLE(), WONTOP(), WOUTPUT()

Description

The HIDE WINDOW command is used to temporarily remove, or 'hide', a window or group of windows from the screen. A window is an area of the screen designated for output and input. There is no limit to the number of defined windows. Windows are defined with the DEFINE WINDOW command, and are activated with the ACTIVATE WINDOW command.

The <window-name> is the name of the window as previously specified with the DEFINE WINDOW command. A <window-name list> is a list of window names, each separated by a comma. To hide all currently defined windows, use the ALL keyword.

The HIDE WINDOW command does not deactivate windows. Output may be directed to a hidden window, as long as that window is active. Output to a hidden window remains hidden until the window is revealed with either the SHOW WINDOW or ACTIVATE WINDOW commands.

The HIDE WINDOW command may be used in a hot key procedure to switch the screen display from windows to full screen. Full screen display is enabled with the ACTIVATE SCREEN command. Hot keys enable users, while running an application that is waiting for keyboard input, to press a key to execute a specified procedure.

Example

```
// winproc.prg
```

```
procedure hide_win
activate screen
display all fields name, event
hide window browse
return
```

```
procedure show_win
show windows
activate window command
return
```

```
set procedure to winproc
set key -1 to hide_win
set key -2 to show_win
```

```
set status off
clear
define window browse;
    from 2,2 to 12,43;
    title "BROWSE";
    color n/bg;
    float;
    grow;
    shadow
activate window browse
```

Products

Recital Mirage Server, Recital Terminal Developer

IF

Class

Applications

Purpose

Conditional command execution

Syntax

```
IF <condition>
[ELSEIF<condition>]
[ELSE]
ENDIF
```

See Also

DO CASE, DO WHILE, IIF()

Description

The IF command provides a conditional selection of commands to execute based upon a logical <condition>. If the result of the <condition> is .T., then the commands following the IF, up to an ENDIF statement or an ELSE statement are executed. IF statements may be nested, i.e. IF statements may contain other IF statements, provided that the ELSE and ENDIF statements correspond with a valid IF.

ELSEIF

The ELSEIF clause can be added to the IF control structure allowing for the testing of more than one condition in the IF...ENDIF block. The IF block is now essentially the same as the DO CASE structure. ELSEIF is analogous with the CASE statement.

ELSE

The ELSE statement is analogous with the OTHERWISE statement. If no previous IF <condition> or ELSEIF <condition> is true, the commands following the ELSE statement up to the ENDIF statement are executed.

Example

```
use patrons index events, names
seek "BALLET"
if found()
    edit
else
    dialog message "Record not found."
endif
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

#IF...#ENDIF

Class

Applications

Purpose

Compiler directives to allow inclusion or exclusion of source code based on a condition

Syntax

```
#IF <expN1> | <expL1>
<statements1>
[#ELIF <expN2> | <expL2>
<statements2>...
#ELIF <expNN> | <expLN>
<statementsN>]
[#ELSE
<statements_else>]
#ENDIF
```

See Also

COMPILE, #DEFINE, DO, DO CASE, IF, #IFDEF...ENDIF, #INCLUDE, #UNDEF, SET COMPILE, SET DEVELOPMENT

Description

The #IF compiler directive can be used to allow inclusion or exclusion of source code based on a condition. The condition can be a numeric expression, <expN1> or any valid expression evaluating to a logical true (.T.) or false (.F.), <expL1>, and is evaluated at compile time. If the <expL1> evaluates to true or the <expN1> evaluates to a nonzero value, the <statements1> that follow are included in the compiled program file and the compilation continues after the #ENDIF. The <statements1> can be any valid Recital 4GL commands. If the <expL1> evaluates to false or the <expN1> evaluates to zero, the <statements1> are excluded and any included #ELIF directives are evaluated in turn in a similar way. If a #ELIF condition evaluates to a non-zero value or to true, the statements that immediately follow are included in the compiled program file and the compilation continues after the #ENDIF. If no #ELIF directives are specified, or if they all evaluate to zero or to false, a check is made for a #ELSE directive and its <statements_else> included in the compiled file if it exists.

This directive can only be used in compiled programs.

Example

```
#IF OS() = "Windows Servers"
  dirterm = "\"
#ELIF OS() = "Linux Servers"
  dirterm = "/"
  set filecase on
#ELIF OS() = "OpenVMS Servers"
  dirterm = "]"
#ELSE
  dirterm = "/"
  set filecase on
#ENDIF
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

#IFDEF...#ENDIF

Class

Applications

Purpose

Compiler directives to allow inclusion or exclusion of source code based on a compile-time constant

Syntax

```
#IFDEF <constant>
[<statements1>]
[#ELSE]
[<statements2>]
#endif
```

See Also

COMPILE, #DEFINE, DO, DO CASE, IF, SET COMPILE, SET DEVELOPMENT

Description

The #IFDEF compiler directive can be used to allow inclusion or exclusion of source code based on the existence of a compile-time constant. The constant, <constant>, is defined using the #DEFINE directive. At compile time, the compiler checks if the <constant> is defined. If it is defined, the <statements1> that follow are included in the compiled program file. The <statements1> can be any valid Recital 4GL commands. If the <constant> is not defined, the <statements1> are excluded. If the <constant> is not defined and there is a #ELSE directive specified, the <statements2> following the #ELSE directive are included in the compiled program file.

This directive can only be used in compiled programs.

Example

```
#DEFINE RECITAL 1
#ifdef RECITAL
    set compatible to recital
    set filetype to recital
#else
    set compatible to vfp
    set filetype to vfp
#endif
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

#INCLUDE

Class

Applications

Purpose

Include another source file in the current program

Syntax

```
#INCLUDE "<filename>"
```

See Also

COMPILE, DEBUG, DO, LINK, MODIFY COMMAND, PARAMETERS, PROCEDURE, SET CLIPPER, SET CLIPPER5, SET COMPATIBLE, SET MAXDBO, SET PSHARE

Description

The #INCLUDE directive is used to include another source file in the current program. This is the equivalent of issuing a DO <filename>. As with the DO command, the file extension must be specified if it is not the default '.prg'.

To make functions or procedures from a function library available to another program, the SET PROCEDURE command should be used.

Example

```
//Check user details before proceeding
public validuser
#include "checklogin.ch"
if not validuser
    return
endif
...
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

INDEX ON

Class

Indexing

Purpose

Create an index for a table on a specified key

Syntax

INDEX ON <key expression> TO <.ndx filename>
[FOR <condition>] [UNIQUE]

INDEX ON <key expression> TAG <tagname> [OF <.dbx filename>]
[FOR <condition>] [UNIQUE] [DESCENDING]

See Also

FIND, CLOSE INDEX, COPY INDEXES, COPY TAG, CREATE VIEW, DELETE TAG, DBXDESCEND(), DESCEND(), DTOS(), LOOKUP(), MDX(), LTOS(), REINDEX, RLOOKUP(), SEEK(), SEEK, SET INDEX, SET ORDER TO, STR(), STRZERO(), TAG(), TAGCOUNT(), TAGNO(), USE

Description

Indexing provides the means of ordering the way in which records are viewed in a table without actually physically rearranging them. Indexing is far more efficient than sorting, and provides the same logical ordering of the table.

The INDEX command creates an index file based upon the evaluation of the specified <key expression>. The Recital/4GL allows you to index on any valid character, date or numeric expression. You can construct an index on mixed data types using the conversion functions STR(), DTOS() and LTOS(). If the STR() function is used on a numeric field in the index expression, the index key storage sequence is unaffected.

Two types of index exist within Recital, single indexes and multiple indexes. The INDEX ON...TO <ndx filename> command creates single index files with a default file extension of '.ndx'. With single indexes, each different sort order is stored in a separate file and each single index must be opened whenever the table is opened to ensure that the indexes are kept up to date. A table can have up to 20 open single index files. If data is modified when the index file is not opened, the index file will have to be reindexed using the REINDEX command. Once an index file has been created, you can use it at any time with your table by specifying the INDEX option with the USE command when you open the table, or using the SET INDEX TO command.

The INDEX ON...TAG <tagname> command creates multiple, or tagged, index files. A multiple index file may contain up to 128 index tags, each with a unique tagname. Each index tag represents a separate key expression. Multiple index filenames have .dbx as the file extension. Tagnames must be a maximum of 32 characters long. They must start with a letter or underscore and can include letters, underscores and digits 0-9.

There are two types of .dbx files. One type, known as the production index, is associated with a DBF file and is opened automatically when the .dbf file is opened. A production index file will always have the same basename as its table. The second type of .dbx file can be created with the OF <.dbx filename> qualifier. This creates a .dbx file that has a different name than the .dbf file, and may be opened separately. This type of multiple index file must be opened with the SET INDEX command, or the INDEX clause of the USE command.

The chief benefit of multiple index files is that many tags representing many <key expressions> are contained in one file. Each tag is updated as the table is updated, providing quick access to alternate ways of ordering records. The master order may be set to any of the index tags using the SET ORDER TO TAG, SET INDEX TO ...ORDER <tagname> commands, or the ORDER clause of the USE ... INDEX command.

If for any reason an index is not consistent with the table file, the Recital/4GL will display an appropriate error message. You can rebuild the index file using the REINDEX command.

To look up indexed records by their key fields, use the FIND or SEEK commands or the RLOOKUP(), LOOKUP() or SEEK() functions. The Recital/4GL performs partial key searches unless SET EXACT ON is in effect.

The key expression that you specify is stored inside the index file in an area known as the index file prologue. If you have used memory variables or alias pointers (->) inside the key expression, you must make sure that these are available any time the index is open. If User Defined Functions (UDFs) are present in the index expression, they must be activated before the index is opened and be available while the index is open. The expression that the UDF returns must be of fixed length. The RPAD() function can be used to make sure of this.

The default file extensions for single and tagged indexes can be changed using the SET COMPATIBLE and SET INDEXEXT commands.

DESCENDING

Indexing is always performed in ascending order, but if an index is created using the DESCENDING keyword, records are ordered in descending sequence. The DESCENDING option is only available for multiple index file tags, for single indexes, the DESCEND() function can be used within character key expressions. The DESCENDING keyword can be used on character string, numeric, date or logical index keys. The DESCENDING() function can be used to determine whether an index tag was created using the DESCENDING keyword. The DBXDESCEND() function must be used when searching in indexes created using the DESCENDING keyword.

FOR <condition>

The optional FOR clause is used to create an index that will contain those records that match the specified <condition>. The <condition> is saved with the index <key expression>, so further updates to the table will add only those records matching the <condition> to the index. When an index is created with a FOR <condition>, all currently open single indexes are closed. The SET INDEX TO command may be used to reopen the original index list. Conditional indexes are useful for taking a 'snapshot' of tables. Subsequent processing of files that are 'filtered' in this manner is dramatically accelerated. The FOR() function can be used to return the FOR <condition> of the active index.

UNIQUE

If SET UNIQUE ON is in effect or the UNIQUE keyword is specified, then only the first key of a given key expression will be added to the index. Duplicates will be discarded. The uniqueness of the records that you subsequently add to the table is checked, and an appropriate message displayed if the record cannot be added or updated because of a pre-existing key in the index file. If SET UNIQUE OFF is in effect and the UNIQUE option is not specified when the index is created, then duplicate keys are allowed. With SET PCUNIQUE ON, duplicate records will be hidden but can be added to unique indexes.

Example

use patrons
index on event tag event

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

INFO

Class

Information Center

Purpose

Initiate the Recital Information Center

Syntax

INFO [TITLE <expC>]

See Also

CATALOG(), SET CATALOG, SET TITLE, DESIGN

Description

The INFO command initiates the Recital Information Center in the current directory.

The Recital Information Center is a powerful workspace, from which any type of data may be accessed, viewed, modified and organized. Within the Recital Information Center, you can organize files into catalogs that represent a single application, an application interface or development project. The rich functionality of the Recital Information Center includes a graphical-like user interface with CUA compliant components, access to Recital workspaces, object oriented design tools and connectivity to foreign databases.

TITLE <expC>

The TITLE keyword is used to specify a title for the Information Center. When not specified, the default title is "Recital Executive Information Center".

IF SET DESIGN is ON, the Information Center initializes in Design mode allows, which allows access to CREATE and MODIFY workspaces as well as the complete range of design tools. Design mode may also be enabled with the DESIGN command. When in Design mode, the Recital Information Center displays a [DESIGN] key that is used to access the development workspaces.

The SET CATALOG TO <expC> command allows the specified catalog file to be used when the Recital Information Center is activated. If a catalog with the name specified in <expC> does not exist, it will be created. SET CATALOG ON causes all newly created files to be added automatically to the currently open catalog. When SET CATALOG is OFF, the currently open catalog is closed and automatic file adding is disabled. Catalogs may also be opened from within the Recital Information Center and files added to a catalog using the appropriate menu items.

The Recital Information Center menu bar provides access to a wide range of decision support tools, application objects and work surfaces. The menu bar contains the following options:

Menu	Description
File	Catalog file tasks
Edit	Database table record options
Options	Catalog file options
Tools	Popup Calculator, Time Manager and Notepad
Customize	Environment customization toggles
Summary	Financial and statistical calculations
Organize	Database table operations
Report	Quick and Custom Report and Label options
Help	Help System, Key Help and Technical Support

The Recital Information Center contains six file panels providing vertical listings of filenames without extensions. Each panel represents a different category or file type and contains a <create> button giving access to the appropriate create worksurface.

Panel	File Types
Data	Table (.dbf), gateway (.gtw), bridge (.brg), view (.vue)
Text	Text (.txt)
Form	Screen format (.fmt)
Report	Report format (.frm) Treport (.trf)
Label	Label format (.lbl)
Program	Application (.app), program (.prg)

Example

set design on
info

Products

Recital Terminal Developer

INPUT

Class

Input/Output

Purpose

Read an expression from the keyboard

Syntax

INPUT [<expC>] TO <memvar>

See Also

ACCEPT, WAIT, STORE

Description

The INPUT command displays the prompt <expC> on the screen, or a “:” if none is specified, and then reads any valid expression from the keyboard. The expression is evaluated, and the result is stored in the specified <memvar>. If you want to INPUT a character string, then you must enclose the string in quotes, i.e. '...', “...”, or [...]. If you enter an invalid expression then the following message will be displayed:

Syntax error, try again

You must then re-input an answer to the prompt.

Example

input “Replace with what? ” to value

Replace with what? PRICE*1.2

Products

Recital Terminal Developer

INSERT

Class

Screen Forms

Purpose

Insert a record into the active table at the current position

Syntax

INSERT
[BEFORE]
[BLANK]
[NOCLEAR]
[NOORGANIZE]

See Also

@...GET, APPEND, APPEND BLANK, APPEND FROM, CREATE, CREATE SCREEN, CHANGE, SET CARRY, FMT()

Description

The INSERT command is a full screen command used to insert records into the active table. A default form with blank fields will be activated on the screen. If the active table is indexed or if you are currently at the bottom of the table, then INSERT works just like APPEND.

You can design your own forms for inserting records with the Forms Designer (CREATE SCREEN). Once the form has been activated with SET FORMAT TO, it will be used instead of the default form when the INSERT command is issued.

Normally, all of the fields are initialized to blank. This behavior can be overridden with the SET CARRY ON command or the [CARRY MODE] key from within the append form. If SET CARRY is ON, data from the previous record will be carried over as the default for the next append operation. Default information may also be automatically inserted into the form via the Applications Data Dictionary.

Using INSERT on a non-indexed table is not recommended, since each time a record is inserted existing records have to be physically moved in the table. The APPEND command is the preferred solution.

BEFORE

The BEFORE keyword has been added solely for language compatibility with Xbase products.

BLANK

If the BLANK option is specified, then a blank record is inserted after the current record and the INSERT form is not activated.

NOCLEAR

The NOCLEAR keyword disables the erasing of the screen on entry and exit from APPEND.

NOORGANIZE

The NOORGANIZE keyword has been added solely for language compatibility with Xbase products.

The following keys are active within an INSERT form.

Key	Action
ABANDON	Discard current record then exit from the form
CARRY MODE	Toggle CARRY on and off

CURSOR DOWN	Skip to next field
CURSOR LEFT	Skip to previous field
CURSOR RIGHT	Skip to next field
CURSOR UP	Skip to previous field
DELETE FIELD	Initialize field
EDIT FIELD	Enter field edit mode
EXIT/SAVE	Write current record then exit from the form
HELP	Activate pop-up help
MENUBAR	Activate the APPEND menu bar
NEXT RECORD	Write current record then exit from the form
REFRESH	Redraw the form
TAB	Toggle function key menu on and off

If SET MOUSE is ON, cursor keys will move the cursor anywhere on the screen rather than just from field to field. If SET NAVIGATE is ON, the cursor moves to fields following the direction specified by the key being pressed, rather than following the order of the GETS on the form. When the RETURN key is pressed, the cursor moves to the nearest field when SET NAVIGATE is ON.

The following keys are active in field edit mode:

Key	Action
BACKSPACE	Delete character before cursor
CURSOR LEFT	Skip to previous character
CURSOR RIGHT	Skip to next character
DELETE CHAR	Delete character under cursor
DELETE FIELD	Delete from cursor to end of field
DELETE WORD	Delete current word
INSERT MODE	Toggle insert / overwrite mode
WORD LEFT	Skip left a word
WORD RIGHT	Skip right a word

The following menu options are available from the INSERT menu bar in the default form:

Menu Item	Action
Descriptions	Toggle the field descriptions on and off
Help	Activate on-line help system

Example

use patrons index names
insert

Products

Recital Mirage Server, Recital Terminal Developer

INSTALL

Class

Table Basics

Purpose

Imports bridge files and tables and their associated files in ASCII format to allow them to be transferred from a binary incompatible platform

Syntax

INSTALL <filename> [FROM <directory>]

See Also

BUILD, SET FILETYPE

Description

The INSTALL command imports Recital bridge files and tables and their associated structure, data, memo, dictionary and multiple index files from ASCII format to allow them to be transferred from a binary incompatible format. The export on the source machine requires the use of the BUILD command.

<filename>

The <filename> is the name of a '.xat' file. This is a text created by the BUILD command on the source machine.

FROM <directory>

If the optional FROM <directory> clause is used, the .xat files and exported files from <directory> will be used to create the table or tables and their associated files in the specified directory.

Example

On Source machine demo.xaf contains the following lines:

```
dbf,customer.rdb
dbf,accounts.rdb
dbf,state.rdb
dbf,product.rdb
brg,cisamdemo.dbf
```

➤ build demo into ./transfer

On Target Machine, once files have been transferred

➤ install demo from ./transfer

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

JOIN

Class

Table Basics

Purpose

Create a new table by merging two existing ones together

Syntax

JOIN WITH <workarea | alias> TO <.dbf filename> | (<expC>) FOR <condition>
[FIELDS <field list>]

See Also

SET FILTER, SET RELATION

Description

The JOIN command creates the new table <.dbf filename>, by merging records from the active table, and another table selected in another workarea. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then “.dbf” will be used.

The JOIN is accomplished by reading through the active table, one record at a time, and for each of the records, evaluating the specified FOR <condition> against every record in the WITH table. If the <condition> is .T., all of the specified fields are constructed into a record, and this record is written to the new table.

It should be noted that if SET FILTER TO is in effect, then any records from the active table which do not satisfy the filter condition, will be discarded, and not included in the join. Also, if SET DELETED ON is in effect, any records that are marked for deletion, will be discarded, and not included in the join.

If no FIELDS clause is specified the new table will consist of both sets of fields from both tables. If two field names are the same in both tables, the first one will be used, and the second one discarded. The fields will be combined up to the maximum field limit of 256.

FIELDS <field list>

The FIELDS clause allows you to specify which fields you want the new table to consist of. You may reference fields from both of the tables. If a field exists by the same name in both tables, then the field from the active table will be used unless you reference the field name with an alias (->) pointer.

Example

```
select a
use patrons index events
select b
use addresses index names
select a
join with b to operalist for name = b->name
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

KEYBOARD

Class

Keyboard Events

Purpose

Put characters into the terminal input buffer

Syntax

KEYBOARD <expC>

[CLEAR]

See Also

SET PCKEYS ON, LASTKEY(), INKEY(), CTRL(), CHR()

Description

The KEYBOARD command can be used to ‘stuff’ characters, <expC>, into the terminal input buffer. Execution of the KEYBOARD command clears the typeahead buffer. This command is useful for automatically stuffing the keyboard if more key presses are needed than entered. Used in conjunction with the MENU() function or the SET KEY command, multiple selections can be automatically entered by pressing one key. For user defined keyboard macros, see the REPLAY and SET CAPTURE commands.

CLEAR

If the CLEAR keyword is used without a character expression, the keyboard buffer will be emptied. If the CLEAR keyword is used with a character expression, it is processed for Xbase language compatibility.

When PCKEYS is ON, the keys specified with the KEYBOARD command are automatically converted to their logical counterparts according to the current terminal definition.

Example

```
keyboard chr(ctrl('g'))+chr(ctrl('g'))+'e'
```

Products

Recital Mirage Server, Recital Terminal Developer

KEYWORD

Class

Applications

Purpose

Specify user-defined keywords (UDK)

Syntax

KEYWORD [<expC1> [<expC2>]]

See Also

ALIAS, DISPLAY STATUS, FUNCTION, LIST PROCEDURE

Description

User Defined Keywords (UDKs) can be specified with the KEYWORD <expC1> <expC2> command. The <expC1> refers to the new keyword name and the <expC2> refers to the keyword with which <expC1> is synonymous. If no <expC2> is specified, the keyword setting for <expC1> is removed. UDKs can be combined with User Defined Commands (UDCs). Used on its own, the KEYWORD command displays the current UDKs. In a similar way, the ALIAS command displays the current UDCs and the LIST PROCEDURE command displays the current User Defined Functions and procedures.

Example

keyword layout structure

list layout

alias retrieve “list”

retrieve layout

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

LABEL

Class

Input/Output

Purpose

Generate mailing labels from a specified label format definition

Syntax

LABEL FORM <.lbl filename> | (<expC>)

[<scope>]

[FOR <condition>]

[FRAME]

[SAMPLE]

[SEPARATE]

[TO FILE <.txt filename> | (<expC>)]

[TO PRINT]

[TO TERMINAL]

[WHILE <condition>]

[WIDTH <expN>]

See Also

CREATE LABEL, MODIFY LABEL, SET PRINTER, TREPORT

Description

The LABEL command is used to generate and print mailing and other labels. It operates by reading the format/layout of the labels from the <.lbl filename>. If no file extension is specified in the form file name, “.lbl” is assumed. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename.

Keyword	Description
<scope>	If no <scope> is specified, the default is ALL, unless the WHILE clause is used, in which case the <scope> will default to REST.
FOR <condition>	If the FOR clause is specified, only those records which satisfy the <condition> are selected.
FRAME	The FRAME keyword puts a box around each label.
SAMPLE	The SAMPLE option may be used to print sample labels, allowing you to line up the labels in the printer.
SEPARATE	The SEPARATE keyword causes each row of labels to be separated by a line.
TO <file>	The display output will be sent to the specified file. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then “.txt” will be used. The command SET PAGELength governs the output file pagination and SET PAGEWIDTH defines the width of each page. Page numbers are centered on the bottom of the page according to width.
TO PRINT	The display output will be sent to a printer. The TO PRINT option will default to a local printer unless the command SET PRINTER TO \\SPOOLER is issued. The print request will then be spooled to the system printer, which is defined by the environment variable DB_PRINT.
TO TERMINAL	The TO TERMINAL keyword outputs labels in pages to the issuing terminal. The user will be prompted to press a key between pages.
WHILE <condition>	The WHILE <condition> can be used to restrict the range of records which are scanned in a table, and should be used to optimize the execution of the

	command. If the WHILE <condition> is used, then the <scope> will default to REST.
WIDTH <expN>	The WIDTH clause specifies the number of labels to print across the page. The numeric expression <expN> denotes the number of labels.

Example

use patrons index events, names
 label form mail to print for event = "DANCE"
 set printer to \\spooler
 seek "BALLET"
 label form mail to print while event = "BALLET"
 set printer to

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

LINK

Class

Applications

Purpose

Link multiple source program files into a single file

Syntax

```
LINK FROM <.dbl file> | <skeleton> TO <.src file>
[MESSAGE <.map file>]
[COMMAND <expC>]
[COMMENTS]
```

See Also

COMPILE, DO, FUNCTION, PROCEDURE, SET MAXDBO, SET PROCEDURE, SET PSHARE, Recital Linker

Description

The LINK command is used to invoke the Recital linker to link multiple source program files into a single file that can then be compiled. This limits the number of files that need to be open at one time, thus using less Operating System file handles and making more efficient use of shared memory when running applications with SET PSHARE ON.

With SET PSHARE ON, compiled programs are loaded into shared memory when called. All users accessing a particular program can access the same area of shared memory rather than loading the program into private memory. The program is removed from shared memory when it no longer has any attached users. A single compiled program therefore, need only be loaded once and accessed by all users. Multiple smaller compiled programs cause an increased amount of loading and unloading activity in shared memory.

Keyword	Description
<.dbl file>	A text file containing a list of program files to link. These should be listed one to a line and must be unique. If no file extension is given, a .dbl extension is assumed.
<skeleton>	A skeleton pattern for the program files to link. This must include the "*" as a wildcard, e.g. "app1*.prg" or "*.prg"
<.src file>	The name of the output file to be created. If no file extension is specified, the file will be given a .src extension. By convention the .src extension is used to differentiate linked source files from individual .prg files. The output file will be created if it does not exist and overwritten if it does.
MESSAGE <.map file>	The optional MESSAGE clause allows linker message output to be sent to the specified <.map file>. If no file extension is specified, the file will be given a .map extension. This text file will be created if it does not exist and overwritten if it does.
COMMAND <expC>	If the optional COMMAND clause is specified, a 'do <expC>' line will be included in the output file as the first executable line. The <expC> should evaluate to the name of the first procedure to be run. The output file can then be called as a self-contained module rather than being used as a procedure library.
COMMENTS	The optional COMMENTS keyword will leave comments and indentations in the output file. By default these are not included by the linker.

The Recital Linker can also be called from the Operating System command line using the Recital Terminal Developer 'dbl' utility.

Example

```
// Creating linked file app1.src to include all files matching the "app1*.prg" skeleton.  
// Message output will be written to output.map.  
// First executable line will be 'do app1start'  
// Comments and indentations will be retained  
link from "app1*.prg" to app1 message output command "app1start" comments
```

```
// Another example  
***app1.dbl  
app1start.prg  
app1main.prg  
app1end.prg  
***end of app1.dbl  
// Creating linked file app1.src to include all files specified in app1.dbl.  
// Message output will be written to info.map.  
// First executable line will be 'do app1start'  
// Comments and indentations will be stripped  
link from app1 to app1 message info command "app1start"
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

LIST

Class

Fields and Records

Purpose

List the contents of the active table and any related tables

Syntax

```
LIST [<scope>]
[FIELDS <field list>|<exp list>]
[FOR <condition>]
[HEADING]
[OFF]
[TO FILE <.txt filename> | (<expC>)]
[TO PRINT]
[WHILE <condition>]
```

See Also

DISPLAY STATUS, DISPLAY MEMORY, DISPLAY STRUCTURE, LIST, DIR

Description

The LIST command is a general purpose Recital/4GL query command that retrieves and displays the contents of table files on the screen. The LIST command scrolls continuously unless halted by the [HOLD SCREEN] key, thereby differing from DISPLAY commands, which pause every 17 lines until a key is pressed. When displaying a record that is longer than the screen width, the contents to the right of the display normally will not be displayed unless you have set your terminal to wrap. Consult the relevant manual for your terminal regarding this feature.

LIST is more powerful than it looks initially. The expressions that you specify can be any valid Recital/4GL expression, including the use of alias pointers into other workareas. If you have SET RELATION TO another table, for each record that is read from the active table, the related table will have its record pointer positioned, and the appropriate record read into its workarea.

If SET FILTER TO <condition> is in effect, only those records that satisfy the filter <condition> will be displayed. If SET DESCRIPTIONS and SET HEADING are both ON and the FIELDS clause is specified, the field descriptions will be used as the column headings rather than the field names. The command SET HEADING TO SINGLE | DOUBLE | NONE controls the underlining of the column headings.

Keyword	Description
<scope>	If the <scope> is not specified, all records will be displayed, unless the WHILE clause is used, in which case the <scope> will default to REST.
FOR <condition>	Only those records that satisfy the <condition> are displayed.
OFF	Disables the display of the record number in the first column of the results.
FIELDS <list>	Restricts the fields displayed to those specified.
HEADING	A heading corresponding to either the field names or the expression will be displayed above each column even if SET HEADING is OFF.
TO <file>	The display output will be sent to the specified file. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then “.txt” will be used. The command SET PAGELength governs the output file pagination and SET PAGEWIDTH defines the width of each page. Page numbers are centered on the bottom of the page according to width.

TO PRINT	The display output will be sent to a printer. The TO PRINT option will default to a local printer unless the command SET PRINTER TO \\SPOOLER is issued. The print request will then be spooled to the system printer, which is defined by the environment variable DB_PRINT.
WHILE <condition>	The <scope> defaults to REST and records are displayed until the <condition> becomes false.

Example

use patrons index events, names

list fields name, event for event = "BALLET"

seek "OPERA"

list rest name, event, seats, price, seats * price while event = "OPERA"

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

LIST DATABASE

Class

Databases

Purpose

List information about the active database

Syntax

LIST DATABASE

[TO FILE <.txt filename> | (<expC>)]

[TO PRINT[ER]]

See Also

ALTER INDEX, ALTER TABLE, CLOSE DATABASES, CLOSE TABLES, COMPILE DATABASE, CREATE DATABASE, CREATE INDEX, CREATE TABLE, CREATE VIEW, DISPLAY DATABASE, DISPLAY INDEXES, DISPLAY TABLES, DROP DATABASE, DROP INDEX, DROP TABLE, LIST INDEXES, LIST TABLES, OPEN DATABASE, USE, SET EXCLUSIVE, ADATABASES(), DBUSED(), GETENV()

Description

The LIST DATABASE command is used to display information about the currently active database.

Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directories are sub-directories of the Recital data directory. The environment variable / symbol DB_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality. Databases are opened using the OPEN DATABASE command.

The LIST DATABASE command displays the following information:

- Database Name, e.g. southwind
- Database Path, e.g. /usr/recital/data/southwind

and for each table in the database the equivalent of LIST STRUCTURE INDEX followed by LIST DICTIONARY:

- Table file name
- Number of records
- Date of creation
- Date of last update
- Encryption status
- Field names, types, sizes and description
- Total record length
- Production DBX file name
- Index tag names, keys, types and lengths
- Dictionary information

Keyword	Description
TO <file>	The display output will be sent to the specified file. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then “.txt” will be used. The command SET PAGELength governs the output file pagination and SET PAGEWIDTH defines the width of each page. Page numbers are centered on the bottom of the page according to width.
TO PRINT	The display output will be sent to a printer. The TO PRINT option will default to a local printer unless the command SET PRINTER TO \\SPOOLER is issued. The print request will then be spooled to the system printer, which is defined by the environment variable DB_PRINT.

Example

VFP/SQL > OPEN DATABASE southwind

VFP/SQL > LIST DATABASE

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

LIST DICTIONARY

Class

Table Basics

Purpose

Display the currently active dictionary

Syntax

LIST DICTIONARY

[TO FILE <.txt filename> | (<expC>)]

[TO PRINT]

See Also

CREATE, LIST DICTIONARY, SET PRINTER

Description

The LIST DICTIONARY command displays the currently active dictionary.

Keyword	Description
TO <file>	The display output will be sent to the specified file. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then “.txt” will be used. The command SET PAGELength governs the output file pagination and SET PAGEWIDTH defines the width of each page. Page numbers are centered on the bottom of the page according to width.
TO PRINT	The display output will be sent to a printer. The TO PRINT option will default to a local printer unless the command SET PRINTER TO \\SPOOLER is issued. The print request will then be spooled to the system printer, which is defined by the environment variable DB_PRINT.

Example

use demo

list dictionary

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

LIST FILES

Class

Disk and File Utilities

Purpose

Display a directory of files

Syntax

LIST FILES [<skeleton>]

[TO FILE <.txt filename> | (<expC>)]

[TO PRINT]

See Also

DIR

Description

The LIST FILES command displays filenames in the current directory and path (see SET PATH) matching the specified <skeleton>. If LIST FILES is issued with no <skeleton> specified, then it will list details of table files only.

Keyword	Description
TO <file>	The display output will be sent to the specified file. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then “.txt” will be used. The command SET PAGELength governs the output file pagination and SET PAGEWIDTH defines the width of each page. Page numbers are centered on the bottom of the page according to width.
TO PRINT	The display output will be sent to a printer. The TO PRINT option will default to a local printer unless the command SET PRINTER TO \\SPOOLER is issued. The print request will then be spooled to the system printer, which is defined by the environment variable DB_PRINT.

Example

set printer to \\spooler

list files *.prg to print

set printer to

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

LIST HISTORY

Class

Error Handling and Debugging

Purpose

Display a list of previously entered commands

Syntax

LIST HISTORY

[LAST <expN>]

[TO FILE <.txt filename> | (<expC>)]

[TO PRINT]

See Also

!, ALIAS, SET DOHISTORY, SET HISTORY

Description

The LIST HISTORY command displays a list of commands currently held in the command history. When SET HISTORY is ON, all commands entered in interactive command mode are stored in a command history list. The SET HISTORY TO <expN> command can be used to specify the size of the history list. If SET DOHISTORY is also ON, then commands executed in program files are also stored in the command history.

Keyword	Description
LAST <expN>	Displays the last <expN> of previously entered commands
TO <file>	The display output will be sent to the specified file. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then “.txt” will be used. The command SET PAGELNGTH governs the output file pagination and SET PAGEWIDTH defines the width of each page. Page numbers are centered on the bottom of the page according to width.
TO PRINT	The display output will be sent to a printer. The TO PRINT option will default to a local printer unless the command SET PRINTER TO \\SPOOLER is issued. The print request will then be spooled to the system printer, which is defined by the environment variable DB_PRINT.

Example

set history on

dir

use payroll index events, names

list history

1 dir

2 use payroll index events, names

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

LIST INDEXES

Class

Indexing

Purpose

List index information about the current table

Syntax

LIST INDEXES

[TO FILE <.txt filename> | (<expC>)]

[TO PRINT]

See Also

FIND, CLOSE INDEX, COPY INDEXES, COPY TAG, CREATE VIEW, DELETE TAG, DBXDESCEND(), DESCEND(), DTOS(), LOOKUP(), MDX(), LTOS(), REINDEX, RLOOKUP(), SEEK(), SEEK, SET INDEX, SET ORDER TO, STR(), STRZERO(), SYS(), TAG(), TAGCOUNT(), TAGNO(), USE

Description

The LIST INDEXES command is used to display index information about the currently active table. Information is displayed for both production and single index files.

The LIST INDEXES command displays the following information:

- Production DBX file name

and for each tag:

- Tag name
- Key
- Type
- Length

and for each open single index:

- Index file name
- Key
- Index cache size

The master index tag is flagged as such.

Keyword	Description
TO <file>	The display output will be sent to the specified file. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then “.txt” will be used. The command SET PAGELength governs the output file pagination and SET PAGEWIDTH defines the width of each page. Page numbers are centered on the bottom of the page according to width.
TO PRINT	The display output will be sent to a printer. The TO PRINT option will default to a local printer unless the command SET PRINTER TO \\SPOOLER is issued. The print request will then be spooled to the system printer, which is defined by the environment variable DB_PRINT.

Example

// Recital/4GL

use example

list indexes to file ind_info

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

LIST MEMORY

Class

Environment

Purpose

Display the contents of the current memory variables

Syntax

LIST MEMORY

[LIKE <skeleton>]

[TO FILE <.txt filename> | (<expC>)]

[TO PRINT]

See Also

ALIAS, DISPLAY STATUS, RELEASE, SAVE, RESTORE, PUBLIC, PRIVATE, DECLARE

Description

The LIST MEMORY command displays the contents of the memory variables and array elements currently defined. Subject to the available system memory, there is no limit to the number of declared memory variables or to the amount of memory they use.

Keyword	Description
LIKE <skeleton>	Displays all the current memory variables that match the wildcard <skeleton> specification.
TO <file>	The display output will be sent to the specified file. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then “.txt” will be used. The command SET PAGELENGTH governs the output file pagination and SET PAGEWIDTH defines the width of each page. Page numbers are centered on the bottom of the page according to width.
TO PRINT	The display output will be sent to a printer. The TO PRINT option will default to a local printer unless the command SET PRINTER TO \\SPOOLER is issued. The print request will then be spooled to the system printer, which is defined by the environment variable DB_PRINT.

Example

list memory

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

LIST PROCEDURE

Class

Applications

Purpose

Display the currently active procedures and functions

Syntax

LIST PROCEDURE

[TO FILE <.txt filename> | (<expC>)]

[TO PRINT]

See Also

SET PROCEDURE TO, DISPLAY STATUS

Description

The LIST PROCEDURE command displays on screen the currently active procedures and functions.

Keyword	Description
TO <file>	The display output will be sent to the specified file. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then “.txt” will be used. The command SET PAGELength governs the output file pagination and SET PAGEWIDTH defines the width of each page. Page numbers are centered on the bottom of the page according to width.
TO PRINT	The display output will be sent to a printer. The TO PRINT option will default to a local printer unless the command SET PRINTER TO \\SPOOLER is issued. The print request will then be spooled to the system printer, which is defined by the environment variable DB_PRINT.

Example

set procedure to yourlib

list procedure

list procedure to print

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

LIST PROTECTION

Class

Table Basics

Purpose

Display current protection and security settings

Syntax

LIST PROTECTION

[TO FILE <.txt filename> | <expC>]

[TO PRINT]

See Also

CREATE, LIST PROTECTION, STR(), GETGID(), GETPID(), GETUID()

Description

The LIST PROTECTION command is used to display to the screen protection and security access control strings (ACS) for the currently active table. An access control string is a range of user identification codes used to allow groups or individuals to perform certain table operations. Access control strings are specified in the CREATE or MODIFY STRUCTURE work surface under the <SECURITY> and <PROTECTION> menu options.

Keyword	Description
TO <file>	The display output will be sent to the specified file. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then “.txt” will be used. The command SET PAGELength governs the output file pagination and SET PAGEWIDTH defines the width of each page. Page numbers are centered on the bottom of the page according to width.
TO PRINT	The display output will be sent to a printer. The TO PRINT option will default to a local printer unless the command SET PRINTER TO \\SPOOLER is issued. The print request will then be spooled to the system printer, which is defined by the environment variable DB_PRINT.

Example

list protection

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

LIST REPORT

Class

Reports

Purpose

Display the contents of a report format file

Syntax

```
LIST REPORT <.frm filename> | (<expC>)  
[TO FILE <.txt filename> | (<expC>)]  
[TO PRINT]
```

See Also

SET PRINTER, CREATE REPORT, REPORT

Description

The LIST REPORT command provides a listing of the contents of the specified report <.frm filename>. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then “.frm” is assumed. This command is primarily used in preparing system documentation.

Keyword	Description
TO <file>	The display output will be sent to the specified file. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then “.txt” will be used. The command SET PAGELength governs the output file pagination and SET PAGEWIDTH defines the width of each page. Page numbers are centered on the bottom of the page according to width.
TO PRINT	The display output will be sent to a printer. The TO PRINT option will default to a local printer unless the command SET PRINTER TO \\SPOOLER is issued. The print request will then be spooled to the system printer, which is defined by the environment variable DB_PRINT.

Example

```
create report creditlist  
set printer to \\spooler  
list report creditlist to print  
set printer to
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

LIST STATUS

Class

Environment

Purpose

Display the complete status of the session

Syntax

LIST STATUS

[TO FILE <.txt filename> | (<expC>)]

[TO PRINT]

See Also

DISPLAY DICTIONARY, DISPLAY MEMORY, DISPLAY STRUCTURE, DISPLAY USERS, DISPLAY STATUS

Description

The LIST STATUS command displays detailed information about the status of the session, including the following:

- Active status of workareas, including indexes, locks, journals, relations, current record, number of records
- Language setting
- Printer setting
- Path setting
- Programmable function keys

Keyword	Description
TO <file>	The display output will be sent to the specified file. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then “.txt” will be used. The command SET PAGELength governs the output file pagination and SET PAGEWIDTH defines the width of each page. Page numbers are centered on the bottom of the page according to width.
TO PRINT	The display output will be sent to a printer. The TO PRINT option will default to a local printer unless the command SET PRINTER TO \\SPOOLER is issued. The print request will then be spooled to the system printer, which is defined by the environment variable DB_PRINT.

Example

```
set view to patrons
set printer to \\spooler
list status to print
set printer to
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

LIST STRUCTURE

Class

Table Basics

Purpose

Display the structure of the active table

Syntax

LIST STRUCTURE

[IN <alias>]

[INDEX]

[TO FILE <.txt filename> | (<expC>)]

[TO PRINT]

See Also

DISPLAY STATUS, DISPLAY MEMORY, CREATE, MODIFY STRUCTURE, DISPLAY DICTIONARY, DISPLAY USERS

Description

The LIST STRUCTURE command displays the structure of the active table.

Keyword	Description
IN <alias>	The IN <alias> clause is used to display the structure of an open table in a workarea that is not currently selected. Alias names may be assigned to tables with the USE command, or default to the table basename.
INDEX	The INDEX keyword is used to list index tag information along with the structure details.
TO <file>	The display output will be sent to the specified file. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then “.txt” will be used. The command SET PAGELENGTH governs the output file pagination and SET PAGEWIDTH defines the width of each page. Page numbers are centered on the bottom of the page according to width.
TO PRINT	The display output will be sent to a printer. The TO PRINT option will default to a local printer unless the command SET PRINTER TO \\SPOOLER is issued. The print request will then be spooled to the system printer, which is defined by the environment variable DB_PRINT.

Example

```
use patrons
list structure
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

LIST TABLES

Class

Databases

Purpose

List table information about the active database

Syntax

LIST TABLES

[TO FILE <.txt filename> | (<expC>)]

[TO PRINT[ER]]

See Also

ALTER TABLE, ALTER INDEX, CLOSE DATABASES, CLOSE TABLES, COMPILE DATABASE, CREATE DATABASE, CREATE INDEX, CREATE TABLE, CREATE VIEW, DISPLAY DATABASE, DISPLAY INDEXES, DISPLAY TABLES, DROP DATABASE, DROP INDEX, DROP TABLE, LIST DATABASE, LIST INDEXES, OPEN DATABASE, USE, SET EXCLUSIVE, SET SQL, ADATABASES(), DBUSED(), GETENV()

Description

The LIST TABLES command displays the base name and file name including the full path for each table in the currently active database.

Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directories are sub-directories of the Recital data directory. The environment variable / symbol DB_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality. Databases are opened using the OPEN DATABASE command.

Keyword	Description
TO <file>	The display output will be sent to the specified file. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then ".txt" will be used. The command SET PAGELength governs the output file pagination and SET PAGEWIDTH defines the width of each page. Page numbers are centered on the bottom of the page according to width.
TO PRINT	The display output will be sent to a printer. The TO PRINT option will default to a local printer unless the command SET PRINTER TO \\SPOOLER is issued. The print request will then be spooled to the system printer, which is defined by the environment variable DB_PRINT.

Example

VFP/SQL > OPEN DATABASE southwind

VFP/SQL > LIST TABLES

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

LIST TRIGGERS

Class

Table Basics

Purpose

Display triggers associated with current table

Syntax

LIST TRIGGERS

[TO FILE <filename>]

[TO FILE <expC>]

[TO PRINT]

See Also

CREATE, MODIFY STRUCTURE, CREATE SCREEN, MODIFY SCREEN, CREATE REPORT, MODIFY REPORT SET PREFORM TO, SET PRERECORD TO, SET POSTFORM TO, SET POSTRECORD TO, @...GET PREFIELD, @...GET POSTFIELD

Description

The LIST TRIGGERS command is used to display to the screen all triggers that are associated with the currently active table. A trigger is used to call a procedure written in the Recital/4GL. Accessible through the CREATE | MODIFY work surfaces, and through SET commands, triggers may be inserted at table, field, record, form, and report levels.

Keyword	Description
TO <file>	The display output will be sent to the specified file. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then “.txt” will be used. The command SET PAGELength governs the output file pagination and SET PAGEWIDTH defines the width of each page. Page numbers are centered on the bottom of the page according to width.
TO PRINT	The display output will be sent to a printer. The TO PRINT option will default to a local printer unless the command SET PRINTER TO \\SPOOLER is issued. The print request will then be spooled to the system printer, which is defined by the environment variable DB_PRINT.

Example

list triggers

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

LIST USERS

Class

Environment

Purpose

Display all the active users

Syntax

LIST USERS

[TO FILE <.txt filename> | (<expC>)]

[TO PRINT]

See Also

DISPLAY STATUS, DISPLAY MEMORY, DISPLAY DICTIONARY, DISPLAY STRUCTURE

Description

The LIST USERS command displays all the active systems users.

Keyword	Description
TO <file>	The display output will be sent to the specified file. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then “.txt” will be used. The command SET PAGELength governs the output file pagination and SET PAGEWIDTH defines the width of each page. Page numbers are centered on the bottom of the page according to width.
TO PRINT	The display output will be sent to a printer. The TO PRINT option will default to a local printer unless the command SET PRINTER TO \\SPOOLER is issued. The print request will then be spooled to the system printer, which is defined by the environment variable DB_PRINT.

Example

list users

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

LOCAL

Class

Memory Variables

Purpose

Declare a memory variable or array local to the current procedure

Syntax

LOCAL <memvar list> | <array name>

See Also

PUBLIC, PRIVATE, DISPLAY MEMORY, PROCEDURE, DO, FUNCTION, RELEASE, DECLARE, DIMENSION, SAVE TO, RESTORE FROM, STORE

Description

The LOCAL command declares memory variables or arrays to be local to a procedure, function or program. When the procedure, function or program returns, then all of the memory variables and arrays that were declared by the LOCAL command are released.

The memory variables are initially declared as logicals with the value .F., unless SET CLIPPER is ON, in which case they are undefined.

LOCAL variables differ from PRIVATE variables in that a LOCAL variable is not visible to lower level procedures or functions.

See DECLARE or DIMENSION for more details on array declaration.

Example

```
local cTmpbuf
? cTmpbuf
.F.
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

LOCATE

Class

Fields and Records

Purpose

Search the active table for records that satisfy a certain condition

Syntax

```
LOCATE [<scope>]  
[FOR <condition>]  
[WHILE <condition>]
```

See Also

CONTINUE, INDEX, SEEK, FIND, SET FILTER, SET RELATION, SCAN

Description

The LOCATE command evaluates the specified FOR <condition> against each of the records in the active table, within the specified <scope>. If a record is successfully located, the Recital/4GL FOUND() function will return .T., and the EOF() function will return .F.. If your table is large, you are advised to INDEX it and use the SEEK command followed by the LOCATE REST command. This method is much more efficient. If SET FILTER TO <condition> is in effect, only those records that satisfy the <condition> are processed. If no FOR clause is specified, then all records that are not filtered are located.

<scope>

If no <scope> is specified, the default ALL is used, unless the WHILE clause is specified in which case the default <scope> will be REST.

FOR <condition>

If no record is found to satisfy the FOR <condition>, the FOUND() function will return .F.. If the <end of scope> is encountered before the <end of file>, then the EOF() function will still return .F..

WHILE <condition>

The search starts from the current record and searches the rest of the records in the table for a record matching the specified condition. FOUND() will return .F. and EOF() will return .T. if no record matching the condition is located.

The CONTINUE command works in conjunction with LOCATE, in order to continue the search through the table. The SCAN...ENDSCAN command performs a similar function, but is more efficient.

Example

```
use patrons  
locate for event = "DREAM"  
do while found()  
    display event, name, seats, seats * price  
    continue  
enddo
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

LOCKF

Class

Manual Locking

Purpose

Lock a table for exclusive use

Syntax

LOCKF

See Also

UNLOCK, LOCKR, SET EXCLUSIVE

Description

The LOCKF command provides the facility to lock a table for exclusive use. LOCKF works in a similar way to the FLOCK() function, except that program execution is suspended until the lock is granted. In order to prepare a table for locking, SET EXCLUSIVE OFF must be in effect when you USE the table. If you only want to lock a particular record, then you should use LOCKR rather than LOCKF.

The LOCKF command works in conjunction with the UNLOCK command. Whenever you issue CLOSE DATABASES, UNLOCK, USE, QUIT or CLEAR LOCKS, then any active locks will be removed. You can see any active locks using the DISPLAY STATUS command. This command is not normally used, as the Recital/4GL supports automatic file and record locking.

Example

```
set exclusive off
use patrons index events, names
seek "OPERA"
if found()
    lockf
    replace price with price*1.2
    unlock
endif
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

LOCKR

Class

Manual Locking

Purpose

Lock a record for exclusive use

Syntax

LOCKR

See Also

UNLOCK, LOCKF, SET EXCLUSIVE

Description

The LOCKR command provides the facility to lock a record in the active table for exclusive use. LOCKR is similar to the RLOCK() function except that LOCKR suspends program execution until the lock is granted. In order to prepare a table for locking, SET EXCLUSIVE OFF must be in effect when you USE the table.

The LOCKR command works in conjunction with the UNLOCK command. Whenever you issue CLOSE DATABASES, UNLOCK, USE, QUIT or CLEAR LOCKS, then any active locks will be removed. You can see any active locks using the DISPLAY STATUS command. This command is not normally needed as the Recital/4GL performs automatic file and record locking.

Example

```
set exclusive off
use patrons index events, names
seek "ROMEO"
if found()
    lockr
    replace price with price*1.2
    unlock
endif
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

LOGIN

Data Connectivity

Purpose

Login to a gateway

Syntax

LOGIN [<server>,<nodename>,<username>,<password>[,<database>]]

See Also

CREATE GATEWAY, LOGOUT, SET GATEWAY, GATEWAY(), CONNECTED()

Description

The LOGIN command allows you to connect to an external database via the Recital Database Server. In Recital Terminal Developer environments, the login command can be issued without specifying all the connection parameters. In this case, a dialog box labeled “LOGIN TO DATABASE SERVER” will be displayed on the screen prompting for the missing parameters.

Parameter	Description
Server	Database server type, e.g. ORACLE, ODBC, RECITAL
Nodename	IP Address or hostname of the machine on which the database resides
Username	Username to login to the external database
Password	Password for username above
Database	Database to connect to. For Oracle databases this entry can be left blank or used to pass hostname information when SQL*NET is being used. The information required differs depending on the version of SQL*NET being used: SQL*NET 1 - 'database' = T:<node>:<SID> SQL*NET 2 - 'database' = <service name>

Example

login “oracle”, “hp”, “scott”, “tiger”
select * from emp;

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

LOGOUT

Class

Data Connectivity

Purpose

Logout from a gateway

Syntax

LOGOUT

See Also

CREATE GATEWAY, LOGIN, SET GATEWAY, GATEWAY(), CONNECTED()

Description

The LOGOUT command closes the open Client/Server gateway in the current workarea.

Example

logout

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

LOOP

Class

Applications

Purpose

Force control to beginning of a looping command

Syntax

LOOP

See Also

DO WHILE, EXIT

Description

The LOOP command forces program control to the top of the closest DO WHILE or FOR...NEXT loop.

Example

```
use patrons index events, names
do while not eof()
  if deleted()
    loop
  endif
  display event, name, seats, seats * price
  skip
enddo
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

LPARAMETERS

Class

Applications

Purpose

Declare formal local parameters to a procedure or program

Syntax

LPARAMETERS <parameter list>

See Also

DO, LOCAL, PARAMETES, PUBLIC, PRIVATE, PROCEDURE, SET CLIPPER, SET PROCEDURE, DECLARE, &, SET PROCEDURE ADDITIVE

Description

The LPARAMETERS command declares a list of local memory variables or arrays, and assigns them the values of the actual parameters specified on a DO <program | procedure> WITH command. The parameters are initially declared as logicals with the value .F.. The LPARAMETERS command must be the first executable command in a procedure or program. The PCOUNT() function is used to determine how many parameters were passed.

Parameters may be passed which are memory variables (i.e. they are not part of an expression). The contents of these memory variables will be updated when the procedure or program returns. This type of parameter passing is known as call by reference. This is the default for Recital/4GL with PROCEDURES and PROGRAMS. The '@' character may be placed in front of the memory variable name in User Defined Functions (UDF), so that they are called by reference.

If you do not wish the parameters to be modified by the called PROCEDURE or PROGRAM, you should enclose the memory variable in round brackets. This type of parameter passing is known as call by value. Any expressions that you specify as parameters are always call by value parameters. The default passing of parameters with User Defined Functions (UDF) is call by value. If COMPATIBLE is set ON then the parameters will be passed by reference. The limit to the number of parameters that you can pass is 40.

The local memory variables created by the LPARAMETERS command are always released when the procedure or program returns. If CLIPPER is set ON and not all parameters are passed, the variables in the LPARAMETERS command not passed will be defined as type 'U' instead of .F..

Example

```
procedure add
lparameters lpara1, lpara2
result = lpara1+ lpara2
return
```

```
private result
do add with 10, 40
? result
50
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

MENU

Class

Menus

Purpose

Activate the currently defined menu

Syntax

MENU
[EXIT]
[HELPPFILE <.hlp filename> | (<expC1>)]
[NOREFRESH]
[OFF]
[PULLRIGHT]
[QUIT]
[SAVE]
[SCREENMAP]

See Also

@...MENU, MENU AT, MENU COMMAND, MENU FIELDS, MENU FILES, MENU FORMAT, MENU FRAME, MENU QUERY, MENU SCOPE, SET SCREENMAP

Description

The MENU command activates a series of menu options which have previously been defined with the @...MENU command. The menu options are displayed at the locations on the screen as specified by the @...MENU commands. A menu option can be selected by moving to the required option with the cursor and pressing the [RETURN] key, or entering the first letter of the option. If there is more than one menu option starting with the same letter, then the nearest option with the specified letter is selected.

If a selection is made on a menu item which does not have any commands associated with it, the menu is exited, the MENU() function returns the menu option number selected, starting at 0, and the MENUITEM() function returns the menu option as a character string. If no selection was made, then MENU() returns -1 and MENUITEM() returns "". These two functions can be used on exiting from a MENU or in the menu COMMAND lines.

If SET MCONFIRM ON has been invoked, the cursor will wait on the menu option that matches the key pressed. That menu option will only be executed when the [RETURN] key is pressed.

Keyword	Description
EXIT	The menu is exited after the first selection rather than being reactivated and refreshed. If the command NOEXIT is included anywhere in the selected menu item commands, it will overwrite the EXIT keyword. When the EXIT keyword and the NOEXIT command are used in conjunction, they can cause some menu items to keep the MENU active and others to leave the MENU once the command has been executed.
HELPPFILE <.hlp>	A helpful text file can be associated with the menu. The <.hlp filename> will be displayed in a read-only window for viewing when the [HELP] key is pressed. The window will be labeled "Operating instructions." The file name can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then ".hlp" is assumed. The command, INSTRUCT, must be set ON when using this option.
NOREFRESH	Refreshing of a menu item after executing its command is disabled.
OFF	The display of messages in the message line is disabled.

PULLRIGHT	Any pulldown menus are displayed to the right of their corresponding menu options. If PULLRIGHT is not specified, pulldown menus are displayed directly below their corresponding menu options. Cursor movements handle subsequent selection of menu options.
QUIT	When the QUIT option is specified, the [ABANDON] key exits the menu.
SAVE	The menu options are not released and can be activated by another MENU command.
SCREENMAP	The screen is handled as if SCREENMAP is ON even when it is OFF.

When a pulldown menu is activated, the [CURSOR UP] and [CURSOR DOWN] keys operate within the pulldown menu itself and the [CURSOR RIGHT] and [CURSOR LEFT] keys move between the options of the menu from which the pulldown was activated.

Example

```
@0,0 menu "Status";
    command "do statotal";
    help "Total amount of goods ordered for the account prefix."
menu quit
```

Products

Recital Mirage Server, Recital Terminal Developer

MENU AT

Class

Menus

Purpose

Display a framed menu then activate it

Syntax

MENU AT <expN1>,<expN2> TO <expN3>,<expN4> WITH <exp list>

[BOLD]

[CLEAR]

[COMMAND <expC2>]

[LABEL <expC3>]

[OFF]

[QUIT]

[SELECT <expC4>]

See Also

@...MENU, MENU FRAME, MENU COMMAND, MENU FIELDS, MENU FILES, MENU FORMAT, MENU QUERY, MENU SCOPE

Description

The MENU AT command displays a static pop-up choice list made up of a series of characters separated by commas, "<expC>,<expC>" as defined by the WITH <exp list> clause. Each set of characters between the commas becomes a separate menu item. The MENU AT choice list is non-scrollable, and can take up to 19 item, framed with a box. The menu display is positioned with the AT row <expN1>, column <expN2> TO row <expN3>, column <expN4> coordinates.

Keyword	Description
BOLD	The menu frame is highlighted.
CLEAR	The inside of the menu frame is cleared before displaying the menu.
COMMAND <expC2>	The command <expC2> and all selections made from the menu are displayed in the action line (line 22).
LABEL <expC3>	The specified character string <expC3> is displayed at the top of the menu frame.
OFF	The display of messages in the message line is disabled.
QUIT	When the QUIT option is specified, the [ABANDON] key exits the menu.
SELECT <expC4>	Multiple selections can be made. The MENUITEM() function can be used to return a string containing the selections made, each separated with <expC4>. Selections are made by placing the cursor on the required menu item and pressing the [RETURN] key. Once all the required selections have been made, the [EXIT/SAVE] key is used to save them or the [ABANDON] key will cancel them. The ASTORE() function can be used to place all the selections into separate array items.

On completion of this command, the MENUITEM() function will return the menu option selected as a character string.

Example

```
menu at 10,10 to 15,18;  
  label "Options";  
  with "Exit","Browse"
```

```
if menuitem() = "Browse"  
    browse  
else  
    return  
endif
```

Products

Recital Mirage Server, Recital Terminal Developer

MENU BAR

Class

Menus

Purpose

Install a Foxbase style menu bar

Syntax

MENU BAR <array>,<expN>

See Also

@...MENU, MENU, MENU FRAME, MENU COMMAND, MENU FIELDS, MENU FILES, MENU FORMAT, MENU QUERY, MENU SCOPE, READ MENU BAR

Description

The MENU BAR command installs a menu bar <array> into the menu bar. All the elements in the <array> must be defined as character strings. The <array> may be defined as a two-dimensional array. Where the first element column is the menu title and the second element column is a message to be displayed in the message line. The number of menu titles is defined by <expN>. This command is used in conjunction with the READ MENU BAR and MENU commands.

Example

```
// Initialize the array for the menu bar
declare top[3,2]
top[1,1] = "FILE"
top[2,1] = "EDIT"
top[3,1] = "DATA"
top[1,2] = "FILE SELECTIONS"
top[2,2] = "EDIT SELECTIONS"
top[3,2] = "DATA SELECTIONS"
// Initialize arrays for the pull-down menus
// Install the pull-down menu system
// Declare top level menu
declare caTOP [2,2]
caTop [1,1] = "File"
caTop [1,2] = "File operations"
caTop [2,1] = "About"
caTop [2,2] = "Product information."

// Declare pulldown menu items
declare caFiles[3]
caFiles [1] = "Open"
caFiles [2] = "Close"
caFiles [3] = "Exit"

declare caAbout [2]
caAbout [1] = "Program"
caAbout [2] = "Recital"

// Initialize menu system
menu bar caTop, 2
menu 1, caFiles, 7, 3
menu 2, caAbout, 5, 2
```

```
// Activate menu system
nCol=1
nRow=1
read menu bar to nCol, nRow save

// Info about choice
? "You chose the ",caTop[nCol,1], "pulldown."
? "and will execute item number", nRow
?
```

Products

Recital Mirage Server, Recital Terminal Developer

MENU BROWSE

Class

Menus

Purpose

Provide a selection menu from the contents of one or more tables

Syntax

```
MENU BROWSE <expC1>
[AT <expN1>,<expN2> TO <expN3>,<expN4>]
[BOLD]
[CLEAR]
[COMMAND <expC4>]
[FOR <condition>]
[LABEL <expC2>]
[OFF]
[POPUP]
[QUIT]
[SELECT <expC3>]
[WHILE <condition>]
```

See Also

ASTORE(), MENU(), MENUITEM()

Description

The MENU BROWSE command provides a selection menu from the contents of one or more tables. You can browse on any character expression, <expC1>, against records in the currently selected workarea or related workareas.

Files and relationships are also satisfied during the BROWSE. The [PAGE UP],[PAGE DOWN], and the [CURSOR] keys can be used to move about in the menu. The table is browsed from its current position, so you must issue a GOTO TOP if you want to browse the entire table, or alternatively you could use SEEK to position to a specific record. If the target table for a MENU BROWSE is indexed, then pressing the [FIND] key results in a SEEK being performed rather than a sequential (LOCATE) search.

AT <expN1>,<expN2> TO <expN3>,<expN4>

The menu may be optionally positioned with at the specified row and coordinates, otherwise the menu is centered on the screen.

Keyword	Description
BOLD	The menu frame is highlighted.
CLEAR	The inside of the menu frame is cleared before displaying the menu.
COMMAND <expC4>	The command <expC4> and all selections made from the menu are displayed in the action line (line 22).
FOR <condition>	Only the records that satisfy the <condition> criteria are retrieved from the tables.
LABEL <expC2>	The specified character string <expC2> is displayed at the top of the menu frame.
OFF	The display of messages in the message line is disabled.
POPUP	The screen is automatically saved when the menu is displayed, and restored when the menu is cleared.
QUIT	When the QUIT option is specified, the [ABANDON] key exits the menu.

SELECT <expC3>	Multiple selections can be made. The MENUITEM() function can be used to return a string containing the selections made, each separated with <expC3>. Selections are made by placing the cursor on the required menu item and pressing the [RETURN] key. Once all the required selections have been made, the [EXIT/SAVE] key is used to save them or the [ABANDON] key will cancel them. The ASTORE() function can be used to place all the selections into separate array items.
WHILE <condition>	Records are retrieved while the <condition> is true. The record pointer should be positioned on a record matching the condition before issuing the MENU BROWSE.

The user can move to choices in the menu using the [DOWN ARROW] or [UP ARROW] keys to highlight the choice, or by typing the letters of the search string. As each letter is typed the highlight bar is moved to the first choice matching the typed letters. To reset the buffer, press {CTRL} and {Y} simultaneously.

Pop-up choice lists which use MENU BROWSE can be defined globally in the Applications Data Dictionary (see CREATE), or by using the CHOICELIST clause with the @...GET command. Choice lists may also be associated with @...GETS at the forms level as part of a validation procedure.

The MENU() function, when used with the MENU BROWSE command, returns the record number of the selected record. The MENUITEM() function returns the text of the selected menu item line. The length of the returned menu item is the total length of the concatenated fields in <expC1>, or the width of the menu as specified in the <row, col> coordinates. MENU BROWSE does not cause the record pointer to move to the selected record.

Example

```
// Menu browse wp files
proc menu-browse
if filecount("*. "+m_filetype) = 0
    dialog box "No files in this directory. Press any key to continue."
else
    go top
    set message to "Select file required.;
    Press ^K to Find or ^G to Abandon."
    menu browse file + descript + who_c + ;
    dtoc(date_c) + "" + who_m + dtoc(date_m);
    at 3,0 to 18,79; label "File Name;
    Description-----" + ;
    "Created by----Date---Modified by--Date---":
    clear;
    bold:
    quit
    if .not. empty(menuitem())
        m_file=left(menuitem(),10)
        seek m->m_file
    endif
endif
return
```

Products

Recital Mirage Server, Recital Terminal Developer

MENU COMMAND

Class

Menus

Purpose

Display a character string in the action line

Syntax

MENU COMMAND <expC>

See Also

@...MENU, MENU, MENU AT, MENU FIELDS, MENU FILES, MENU FORMAT, MENU FRAME, MENU QUERY, MENU SCOPE

Description

The MENU COMMAND can be used by developers to give further information to the user about the operation that is taking place. It displays the character expression <expC>, preceded by the string "Command: " in the action line (line22 on a 25-line terminal).

Example

Menu command "SELECT * WHERE"

Products

Recital Terminal Developer

MENU FIELDS

Class

Menus

Purpose

Select a field list from a menu

Syntax

MENU FIELDS

[AT <expN1>,<expN2>]

[BOLD]

[CLEAR]

[COMMAND <expC1>]

[INDEX]

[LABEL <expC2>]

[NUMERIC]

[OFF]

[POPUP]

[QUIT]

[SELECT <expC3>]

[SKIP]

See Also

@...MENU, MENU, MENU AT, MENU COMMAND, MENU FILES, MENU FORMAT, MENU FRAME, MENU QUERY, MENU SCOPE

Description

The MENU FIELDS command displays a menu of fields from the current table.

AT <expN1>,<expN2>]

The menu can optionally be positioned at the specified row <expN1> and column <expN2> positions.

Keyword	Description
BOLD	The menu frame is highlighted.
CLEAR	The inside of the menu frame is cleared before displaying the menu.
COMMAND <expC1>	The command <expC1> and all selections made from the menu are displayed in the action line (line 22).
INDEX	All of the selected fields are converted to character data types suitable for indexing on mixed data types.
LABEL <expC2>	The specified character string <expC2> is displayed at the top of the menu frame.
NUMERIC	Only the numeric fields from the table will be included in the menu.
OFF	The display of messages in the message line is disabled.
POPUP	The screen is automatically saved when the menu is displayed, and restored when the menu is cleared.
QUIT	When the QUIT option is specified, the [ABANDON] key exits the menu.
SELECT <expC3>	Multiple selections can be made. The MENUITEM() function can be used to return a string containing the selections made, each separated with <expC3>. Selections are made by placing the cursor on the required menu item and pressing the [RETURN] key. Once all the required selections have been made, the [EXIT/SAVE] key is used to save them or the [ABANDON] key will cancel them. The ASTORE() function can be used to place all the

	selections into separate array items.
SKIP	Selections can be made from more than one table. The [CURSOR RIGHT] key skips to the next workarea, and the [CURSOR LEFT] key skips to the previous workarea. Names of fields from other workareas will be prefixed with the workarea alias pointer, e.g. patrons->name.

If the SET DESCRIPTIONS command is set to ON, field descriptions are listed within the menu instead of field names. The currently selected field name, type and width are displayed above the field descriptions. When SET DESCRIPTIONS is set to OFF, the reverse is true: the currently selected field description displays above the menu of field names, types and widths.

The following keys are active in MENU FIELDS:

Key	Action
[PAGE DOWN]	Display the next page of fields
[PAGE UP]	Display the previous page of fields
[CURSOR UP]	Skip to the previous field or previous page if on the top field of a page
[CURSOR DOWN]	Skip to the next field or next page if on the last field of a page
[CURSOR RIGHT]	Display fields from next workarea
[CURSOR LEFT]	Display fields from previous workarea
[RETURN]	Select a field
[ABANDON]	Discard selected field and exit
[EXIT/SAVE]	Exit with selected fields

On exit from MENU FIELDS the MENUITEM() function will return the field list as a character expression. If the [ABANDON] key was pressed, or no fields were selected, then MENUITEM() will return a null string and MENU() will return -1.

Example

```
menu fields at 2,0;
  label "Fields" select ",", skip
store menuitem() to fieldlist
display all &fieldlist
```

Products

Recital Terminal Developer

MENU FILES LIKE

Class

Menus

Purpose

Display a menu of files for selection

Syntax

MENU FILES LIKE <skeleton>

[AT <expN1>,<expN2> TO <expN3>,<expN4>]

[BOLD]

[CLEAR]

[COMMAND <expC1>]

[INDEX]

[LABEL <expC2>]

[OFF]

[POPUP]

[QUIT]

[SELECT <expC3>]

[TRIM]

See Also

@...MENU, MENU, MENU AT, MENU COMMAND, MENU FIELDS, MENU FORMAT, MENU FRAME, MENU QUERY, MENU SCOPE

Description

The MENU FILES LIKE command displays a menu of file names that match the specified <skeleton> pattern.

AT <expN1>,<expN2> TO <expN3>,<expN4>

The menu may be optionally positioned with the AT row <expN1>, column <expN2> and TO row <expN4> clauses, otherwise the menu is centered on the screen.

Keyword	Description
BOLD	The menu frame is highlighted.
CLEAR	The inside of the menu frame is cleared before displaying the menu.
COMMAND <expC1>	The command <expC1> and all selections made from the menu are displayed in the action line (line 22).
INDEX	Only index files that match the active table are displayed in the menu. As the user scrolls through the menu items the corresponding index expression for the index file is displayed in the message line.
LABEL <expC2>	The specified character string <expC2> is displayed at the top of the menu frame.
OFF	The display of messages in the message line is disabled.
POPUP	The screen is automatically saved when the menu is displayed, and restored when the menu is cleared.
QUIT	When the QUIT option is specified, the [ABANDON] key exits the menu.
SELECT <expC3>	Multiple selections can be made. The MENUITEM() function can be used to return a string containing the selections made, each separated with <expC3>. Selections are made by placing the cursor on the required menu item and pressing the [RETURN] key. Once all the required selections have been made, the [EXIT/SAVE] key is used to save them or the [ABANDON] key

	will cancel them. The ASTORE() function can be used to place all the selections into separate array items.
TRIM	File extensions are not displayed in the menu.

A file can be selected by entering the first character of its name. The following keys are also active in MENU FILES LIKE:

Key	Action
[PAGE DOWN]	Display the next page of files
[PAGE UP]	Display the previous page of files
[CURSOR UP]	Skip to previous file
[CURSOR DOWN]	Skip to next file
[RETURN]	Select a file
[ABANDON]	Discard selected files and exit
[EXIT/SAVE]	Exit with selected files

On exit from MENU FILES LIKE the MENUITEM() function will return the file list as a character string. If the [ABANDON] key was pressed, or no files were selected, then MENUITEM() will return a null string and MENU() will return -1.

Example

```
menu files like *.frm at 2,0 to 10,18;
  label "Reports"
store menuitem() to rpt
report form &rpt to print
```

Products

Recital Terminal Developer

MENU FORMAT

Class

Menus

Purpose

Activate a menu format procedure

Syntax

MENU FORMAT <procedure> | <program> | (<expC1>)

[EXIT]

[HELPPFILE <.hlp filename> | (<expC2>)]

[NOREFRESH]

[OFF]

[QUIT]

[SAVE]

[SCREENMAP]

See Also

@...MENU, MENU, MENU AT, MENU COMMAND, MENU FIELDS, MENU FILES, MENU FRAME, MENU QUERY, MENU SCOPE

Description

The MENU FORMAT command executes a procedure or program containing a menu definition. The MENU command provides full details of how menus are defined. The specified procedure should contain a series of commands that are used to build a menu. A MENU FORMAT procedure differs from a normal procedure in the following way: when a menu option is selected and its associated commands executed, the MENU FORMAT procedure is automatically re-executed. MENU FORMAT procedures may call other MENU FORMAT procedures from selected menu options, but not as a command when the MENU FORMAT is being processed.

Keyword	Description
EXIT	The menu is exited after the first selection rather than being reactivated and refreshed.
HELPPFILE <.hlp>	A helpful text file can be associated with the menu. The <.hlp filename> will be displayed in a read-only window for viewing when the [HELP] key is pressed. The window will be labeled "Operating instructions." The file name can be substituted with a <expC2>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then ".hlp" is assumed. The command, INSTRUCT, must be set ON when using this option.
NOREFRESH	The menu is generated without executing the extra commands (e.g. frame drawing) associated with it.
OFF	The display of messages in the message line is disabled.
QUIT	When the QUIT option is specified, the [ABANDON] key exits the menu.
SAVE	The menu options are not released and can be activated by another MENU command.
SCREENMAP	The screen is handled as if SCREENMAP is ON even when it is OFF.

Example

```
procedure main
clear
@0,0 menu "Exit";
    command "quit";
    help "Exit the system."
@0,5 menu "Browse";
    command "browse";
    help "Browse the table."
return
menu format main
```

Products

Recital Mirage Server, Recital Terminal Developer

MENU FRAME

Class

Menus

Purpose

Draw a menu frame

Syntax

MENU FRAME AT <expN1>,<expN2> TO <expN3>,<expN4>
LABEL <expC>

See Also

@...MENU, MENU AT, MENU COMMAND, MENU FIELDS, MENU FILES, MENU FORMAT,
MENU QUERY, MENU SCOPE

Description

The MENU FRAME command draws a box with the specified label <expC> and can be used to ‘frame’ a menu. The frame is positioned at the coordinates specified in <expN1> to <expN4>.

Example

Menu frame at 10,10 to 15,18 label “Options”

```
@13,11 menu “Exit”;  
    help “Exit the system.”  
@14,11 menu “Browse”;  
    command “browse”;  
    help “Browse the table.”  
menu
```

Products

Recital Mirage Server, Recital Terminal Developer

MENU FROM

Class

Menus

Purpose

Display a framed menu of options from a database table file

Syntax

```
MENU FROM <mdf filename> | (<expC1>)  
AT <expN1>,<expN2> TO <expN3>,<expN4>  
[BOLD]  
[CLEAR]  
[EXIT]  
[HELPPFILE <hlp filename> | (<expC2>)]  
[LABEL <expC3>]  
[NOREFRESH]  
[OFF]  
[QUIT]  
[SELECT <expC4>]
```

See Also

@...MENU, MENU AT, MENU COMMAND, MENU FIELDS, MENU FILES, MENU FORMAT, MENU QUERY, MENU SCOPE

Description

The MENU FROM command uses the entries in a standard Recital table (but with a .mdf filename) to construct a framed menu of options. The filename can be substituted with a <expC1>, enclosed in round brackets, which returns a valid filename. The table has an extension of .mdf and must have the following structure:

Field	Type	Width
MENU	Character	25
COMMAND	Character	80
HELP	Character	80
NOREFRESH	Logical	1

Each record in the database is equivalent to a command in the from: @...MENU <item> COMMAND <command> HELP <help> [NOREFRESH]. When a menu is activated from an <.mdf filename> the items in the database are vertically scrollable as with the MENU FILES command. The command list specified with a menu item can be another MENU FROM command, or a DO <procedure> command which activates another menu using any of the set of menu commands available in the Recital 4GL.

AT <expN1>,<expN2> TO <expN3>,<expN4>

The upper left corner of the menu is positioned on the coordinates <expN1> and <expN2>, and the lower right corner of the menu is positioned on the coordinates <expN3> and <expN4>.

Keyword	Description
BOLD	The menu frame is highlighted.
CLEAR	The inside of the menu frame is cleared before displaying the menu.
EXIT	The menu is exited following selection of a menu item and execution of the associated commands.. If the NOEXIT command is executed as one of the

	commands associated with a menu item, then the menu will not be exited.
HELPPFILE <.hlp>	A helpful text file can be associated with the menu. The <.hlp filename> will be displayed in a read-only window for viewing when the [HELP] key is pressed. The window will be labeled "Operating instructions." The file name can be substituted with a <expC2>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then ".hlp" is assumed. The command, INSTRUCT, must be set ON when using this option.
LABEL <expC3>	The specified character string <expC3> is displayed at the top of the menu frame.
NOREFRESH	The menu is generated without executing the extra commands (e.g. frame drawing) associated with it.
OFF	The display of messages in the message line is disabled.
QUIT	When the QUIT option is specified, the [ABANDON] key exits the menu.
SELECT <expC4>	Multiple selections can be made. The MENUITEM() function can be used to return a string containing the selections made, each separated with <expC4>. Selections are made by placing the cursor on the required menu item and pressing the [RETURN] key. Once all the required selections have been made, the [EXIT/SAVE] key is used to save them or the [ABANDON] key will cancel them. The ASTORE() function can be used to place all the selections into separate array items.

Example

```
create menufile
rename menufile.dbf menufile.mdf
menu from menufile at 1,35 to 10,45;
    label "Options"
```

Products

Recital Mirage Server, Recital Terminal Developer

MENU QUERY

Class

Menus

Purpose

Construct a query condition through a series of menus

Syntax

MENU QUERY

[AT <expN1> [TO <expN2>]]

[BOLD]

[CLEAR]

[COMMAND <expC1>]

[HELPPFILE <.hlp filename> | (<expC2>)]

[OFF]

[POPUP]

[SCREENMAP]

See Also

@...MENU, MENU AT, MENU COMMAND, MENU FIELDS, MENU FILES, MENU FORMAT, MENU QUERY, MENU SCOPE

Description

The MENU QUERY command displays a series of four menus with a top line at the specified row <expN1>. The optional AT clause is used to specify the beginning row of the display. The menu query display starts on row 2 by default. The maximum row number for the AT clause is 9 for 24 line terminals, and 10 for 25 line terminals. The TO <expN2> clause is used to specify an ending row number for the FIELDS MENU. The height of the FIELDS MENU will adjust to row numbers specified with the AT and TO clauses.

The four menus displayed are the FIELDS MENU, the OPERATOR MENU, the CONNECTOR MENU, and the QUERY FILES MENU. Selections from these menus construct a query condition for the active table. When the menus display, the FIELDS MENU is automatically activated.

The FIELDS MENU contains fields from the active table. Selections from this menu provide the query condition with a specific field to select certain records from the table. If the SET DESCRIPTION command is set to ON, field descriptions are listed within the menu instead of field names. The currently selected field name, type and width are displayed above the field descriptions. When SET DESCRIPTIONS is set to OFF, the reverse is true: the currently selected field description is displayed above the menu of field names, types, and widths. After you have selected a field, the OPERATOR MENU is activated to allow specification of a value for that field.

The OPERATOR MENU contains operators with which to limit record selection by specifying a field value. After an operator is selected from this menu, a dialog box appears requesting a field value. Pressing the [HELP] key from this dialog box displays a pop-up choice list of field values from the currently active table. Character field values are automatically converted to upper case and enclosed in quotation marks.

The OPERATOR MENU also accepts specification of a character field name rather than a field value. When a field name is entered, the query condition compares the value of the field selected from the FIELDS MENU, and the value of the field name entered in the dialog box. The MENU QUERY command will only compare the values of two character fields.

The CONNECTOR MENU allows you to either mark the end of the query, or add another condition. If anything other than <End of query> is selected, the cursor returns to the FIELDS MENU.

The QUERY FILES MENU allows you to either save the current query condition to a file, or load an existing query file. The QUERY FILES MENU is accessed by pressing the [MENUBAR] key. Query file names have an extension of “.qry”.

Keyword	Description
BOLD	The menu frame is highlighted.
CLEAR	The inside of the menu frame is cleared before displaying the menu.
COMMAND <expC1>	The action line (line 22) is updated with the query condition as it is being constructed. The expression <expC1> displays at the beginning of the query condition. <expC1> should include a blank space at the end to separate the command from the query condition.
HELPPFILE <.hlp>	A helpful text file can be associated with the menu. The <.hlp filename> will be displayed in a read-only window for viewing when the [HELP] key is pressed. The window will be labeled “Operating instructions.” The file name can be substituted with a <expC2>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then “.hlp” is assumed. The command, INSTRUCT, must be set ON when using this option.
OFF	The display of messages in the message line is disabled.
POPUP	The screen is automatically saved when the menu is displayed, and restored when the menu is cleared.
SCREENMAP	The screen is handled as if SCREENMAP is ON even when it is OFF.

Menu items can be selected by entering the item’s first character. The following keys are active in the MENU QUERY:

Key	Action
[PAGE DOWN]	Display next page of fields when in the FIELDS MENU
[PAGE UP]	Display previous page of fields when in the FIELDS MENU
[CURSOR UP]	Skip to previous menu item
[CURSOR DOWN]	Skip to next menu item
[CURSOR LEFT]	If in the FIELDS MENU, then skip to previous workarea
[CURSOR RIGHT]	If in the FIELDS MENU, then skip to next workarea
[RETURN]	Select a menu item
[ABANDON]	Discard query and exit
[EXIT/SAVE]	Exit with constructed query

Any character entered in response to a prompt will normally be converted to upper case and quotation marks will be added automatically at the start and end of the string. To enter a query that is case sensitive and contains lower case characters, it is necessary to enclose the string with quotation marks (“”).

On exit from MENU QUERY, the MENUITEM() function will return the correctly constructed query condition as a character string. If the [ABANDON] key was pressed, or no query was constructed, then MENUITEM() will return a null string and MENU() will return –1.

Example

```

menu query command “REPORT FOR”
store menuitem() to query
if query # “”
    query = “FOR ” + query
endif
report form patrons &query to print

```

Products

Recital Terminal Developer

MENU SCOPE

Class

Menus

Purpose

Display a popup menu for choosing record selection scope

Syntax

MENU SCOPE

[AT<expN1>,<expN2>]

[HELPPFILE <.hlp filename> | (<expC1>)]

[LABEL <expC2>]

[OFF]

[POPUP]

See Also

@...MENU, MENU AT, MENU COMMAND, MENU FIELDS, MENU FILES, MENU FORMAT, MENU QUERY

Description

The MENU SCOPE command displays a menu containing possible record selection scopes.

Scope	Description
	Default scope of the operation, e.g. FOR is all, WHILE is rest.
Next	Process next n records. Enter number of records to process, starting from current record.
All	Process all records.
Record	Process selected record. Enter the record number of the record to process.
Rest	Process all remaining records, starting from current record.
First	Process first n records. Enter number of records to process, starting from top of file.

AT <expN1>,<expN2>

Position top left hand corner of menu at these row, column coordinates.

Keyword	Description
HELPPFILE <.hlp>	A helpful text file can be associated with the menu. The <.hlp filename> will be displayed in a read-only window for viewing when the [HELP] key is pressed. The window will be labeled "Operating instructions." The file name can be substituted with a <expC2>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then ".hlp" is assumed. The command, INSTRUCT, must be set ON when using this option.
LABEL <expC2>	The specified character string <expC2> is displayed at the top of the menu frame.
OFF	The display of messages in the message line is disabled.
POPUP	The screen is automatically saved when the menu is displayed, and restored when the menu is cleared.

The following keys are active in the MENU SCOPE:

Key	Action
[CURSOR UP]	Skip to previous menu item
[CURSOR DOWN]	Skip to next menu item
[RETURN]	Select a menu item
[ABANDON]	Abandon selection
[EXIT/SAVE]	Exit without selecting a scope

You can select a menu item by entering its first character.

On exit from MENU SCOPE the MENUITEM() function will return the record selection scope. If the [ABANDON] key was pressed, or no scope was selected, then MENUITEM() will return an empty string "" and MENU() will return -1.

Example

menu scope at 2,0
store menuitem() to scope
report form patrons &scope to print

Products

Recital Mirage Server, Recital Terminal Developer

MENU TO

Class

Menus

Purpose

Return a number according to the menu item chosen

Syntax

MENU TO <memvar>

See Also

@...MENU, MENU AT, MENU COMMAND, MENU FIELDS, MENU FILES, MENU FORMAT, MENU QUERY, MENU SCOPE

Description

The MENU TO <memvar> command returns a number from 1 to n, according to the menu item selected, and zero if [ABANDON] was pressed. The companion command is @<expN>,<expN> PROMPT "item" MESSAGE "text".

The READVAR() function may be used to return the current <memvar>.

The MENU TO <memvar> command starts at the <memvar> position within the range of 1 and the number of @...MENU items defined in the menu.

The COL() and ROW() function may be used to return the coordinates of the current MENU TO <memvar>.

These commands are supported for compatibility with other products. It is far better to use @...MENU...COMMAND for this purpose.

Example

```
@10,10 prompt "edit";
    message "edit a record."
@11,10 prompt "append";
    message "add a record."
@12,10 prompt "delete";
    message "delete a record."
menu to choice
do case
    case choice=1
do editfunc()
    case choice=2
do appendfunc()
    case choice=3
do deletfunc()
    otherwise dialog box "No choice."
endcase
```

Products

Recital Mirage Server, Recital Terminal Developer

MESSAGE

Class

Screen Forms

Purpose

Display message in the message line, and halt program execution until a key is pressed

Syntax

MESSAGE <expC> [QUERY]

See Also

DIALOG BOX, SET MESSAGE, SET STATUS

Description

The MESSAGE command displays the specified character string on line 24, and waits for any key to be pressed before continuing. If SET STATUS is OFF then the message will not be displayed, but rather the bell will sound on the terminal. Whenever the message is displayed, the current cursor position is saved, and then restored after a key has been pressed. The message is automatically erased from line 24 after a key has been pressed. To change the MESSAGE line from line 24, set MESSAGE ON, STATUS OFF and SCOREBOARD OFF. Then use the SET MESSAGE AT and SET MESSAGE TO commands to reposition the message line.

QUERY

If the QUERY keyword is specified, then the message is displayed in a YES/NO dialog box. The response made at the keyboard can be trapped using the LASTKEY() function.

Example

```
seek "BALLET"
if not found()
    message "Record not found. Continue?" query
endif
if lastkey()=asc("N")
    quit
endif
```

Products

Recital Mirage Server, Recital Terminal Developer

METHOD

Class

Applications

Purpose

External method definition for a class

Syntax

METHOD

<class-name> :: <method-name>

See Also

CLASS, PARAMETER, RETURN

Description

An object encapsulates methods that perform operations on the object. Encapsulation hides methods within an object, and makes the object into a fully self-contained operational unit. The METHOD command allows you to define an external method outside the CLASS...ENDCLASS construct. This allows for a procedural library of methods, or for each method to be stored in a separate physical file. An external method can have a parameter list defined with the PARAMETER command. The RETURN command should be used to define the end of the external method.

Keyword	Description
<class-name>	The <class-name> to which the method belongs.
::	The :: operator is used as a separator between the class name and method name.
<method-name>	A unique <method-name> of up to 32 characters must be specified. This will be the name used in the METHOD clause in the CLASS...ENDCLASS construct.

Example

```
class myclass
// Define properties
property color as character
property size as numeric
// Define external methods
method display external
// Define internal methods
method update
....
return
endclass

// External method for 'myclass' class
method myclass::display
parameter whatever
...
return
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

MODIFY BRIDGE

Class

Terminal Developer Development Tools

Purpose

Full screen modification of a bridge definition file through a form

Syntax

MODIFY BRIDGE <brg filename> | (<expC>)

See Also

CREATE BRIDGE, SET BRIDGE, USE

Description

The MODIFY BRIDGE command is a full screen command used to modify an existing bridge definition <.brg filename>. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then “.brg” will be used.

NOTE: Bridge files can also be created and modified via an ‘ini’ file for use in other Recital products.

See the CREATE BRIDGE command for details.

Example

```
modify bridge rms_patrons  
set bridge to rms_patrons
```

Products

Recital Terminal Developer

MODIFY COMMAND

Class

Terminal Developer Development Tools

Purpose

Execute a text editor to edit program files

Syntax

MODIFY COMMAND <prg filename> | (<expC>)

See Also

ED, TEXTEDIT(), VI

Description

MODIFY COMMAND provides the facility to create or modify program files and other text files. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is present in the file name, then '.prg' is used.

The default editors are: the 'vi' editor under UNIX and Linux; the 'edt' editor under VAX/VMS. You may override these defaults using the SET TEDIT TO command.

MODIFY COMMAND is a synonym of the ED and VI commands.

Example

```
modify command myprogram
ed myprogram
vi myprogram
```

Products

Recital Terminal Developer

MODIFY FILE

Class

Terminal Developer Development Tools

Purpose

Execute a text editor to edit text files

Syntax

MODIFY FILE <txt filename> | (<expC>)

See Also

SET FORMAT TO, CREATE SCREEN, MODIFY COMMAND, VI, ED

Description

MODIFY FILE provides the facility to create or modify text files. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is present in the file name, then '.txt' is used.

The default editors are: the 'vi' editor under UNIX and Linux; the 'edt' editor under VAX/VMS. You may override these defaults using the SET TEDIT TO command.

Example

modify file screen.fmt

Products

Recital Terminal Developer

MODIFY GATEWAY

Class

Data Connectivity

Purpose

Full screen modification of a gateway definition file through a form

Syntax

MODIFY GATEWAY <.gtw filename> | (<expC>)

See Also

CREATE GATEWAY, SET GATEWAY, GATEWAY()

Description

The MODIFY GATEWAY command is a full screen command used to modify an existing gateway definition <.gtw filename>. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then “.gtw” will be used. See the CREATE GATEWAY command for details.

Example

```
modify gateway ora_employees
use ora_employees.gtw
```

Products

Recital Terminal Developer

MODIFY LABEL

Class

Terminal Developer Development Tools

Purpose

Full screen modification of a label definition file through a form

Syntax

MODIFY LABEL <lbl filename> | (<expC>)

See Also

CREATE LABEL, LABEL

Description

The MODIFY LABEL command is a full screen command used to modify existing label format files. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then “.lbl” is used.

See the CREATE LABEL command for full details.

Example

use patrons
modify label operalabel
label form operalabel for event = “OPERA”

Products

Recital Terminal Developer

MODIFY MEMO

Class

Fields and Records

Purpose

Open a memo in the active window

Syntax

```
MODIFY MEMO <memofield>  
[NOWAIT]  
[WINDOW <window name>]
```

See Also

SET WINDOW OF MEMO, MEMOEDIT()

Description

The MODIFY MEMO command opens the memo field of the current table record in the currently active window. A window is an area of the screen designated for output and input. Windows are defined with the DEFINE WINDOW command, and are activated with the ACTIVATE WINDOW command. There is no limit to the number of defined windows. Before issuing the MODIFY MEMO command, you must designate a window in which to open memos with the SET WINDOW OF MEMO to <window-name> command. The <window-name> is the name of a window as specified with the DEFINE WINDOW command. The <memofield> is the name of the field as specified in the structure of the table.

When MODIFY MEMO is issued, the active or designated window displays the window and allows users to make changes to the memo. Program execution pauses until the memo is exited. After a memo is exited, control returns to the currently executing program or to the interactive command prompt.

NOWAIT

When MODIFY MEMO is issued with the NOWAIT keyword, control returns immediately to the executing program or the interactive command prompt without waiting for the user to edit the memo.

WINDOW <name>

You may specify a pre-defined window other than the currently active window by using the WINDOW clause. The <window name> must be the name of a window that has been defined with the DEFINE WINDOW command. Using this clause activates the specified window, and deactivates the currently active window.

Example

```
w = woutput()  
activate window browse  
browse noclear nowait nomenu  
activate window memo  
modify memo notes nowait  
activate window edit  
edit noclear nowait nomenu  
activate window &w
```

Products

Recital Terminal Developer

MODIFY REPORT

Class

Terminal Developer Development Tools

Purpose

Full screen modification of a report format file through a form

Syntax

MODIFY REPORT <frm filename> | (<expC>)

See Also

CREATE REPORT, REPORT, TREPORT

Description

The MODIFY REPORT command is a full screen command used to modify existing report format files. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then “.frm” is used.

See the CREATE REPORT command for more details.

Example

use patrons index names, events
modify report concert
report form concert for event = “CONCERT”

Products

Recital Terminal Developer

MODIFY SCREEN

Class

Terminal Developer Development Tools

Purpose

Full screen modification of screen format file

Syntax

MODIFY SCREEN <scr filename> | (<expC>)

See Also

@...GET, @...MENU, CREATE SCREEN, EDIT, CHANGE, APPEND, QUERY, SET FORMAT, SET QUERYMODE, SET UPDATE

Description

The MODIFY SCREEN command is a full screen command used to modify an existing screen format file. The filename can be substituted with an <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then '.scr' will be used. This command executes the Forms Designer.

See the CREATE SCREEN command for more details.

Example

```
modify screen editscreen  
set form to editscreen  
edit
```

Products

Recital Terminal Developer

MODIFY STRUCTURE

Class

Terminal Developer Development Tools

Purpose

Full screen modification of a table structure through a form

Syntax

MODIFY STRUCTURE [<dbf filename> | (<expC>)]

See Also

CREATE, CREATE FROM, COPY STRUCTURE EXTENDED, SET DESCRIPTIONS, SET MCONFIRM, SET RELATION

Description

The MODIFY STRUCTURE command is a full screen command used to modify the structure of table <dbf filename>. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no <dbf filename> is specified the command will work in the current workarea. If no table is open in the current workarea, you will be prompted for a filename. If no file extension is specified, then “.dbf” will be used.

If the command MCONFIRM is set OFF, the MODIFY STRUCTURE menu bar operates as pulldown menus.

When a table structure is modified, a backup of the old table, with the file name extension ‘.bak’ is created.

When modifying a table structure, the table is automatically reloaded. Any changes to the structure of the table cause records from the old table to be appended into the new table. Since the APPEND command only copies fields which exist in both tables, and are of the same data type, you should use MODIFY STRUCTURE twice if you want to change a field name as well as the field width. APPEND triggers defined in the data dictionary will be called for each record as the table is reloaded after a MODIFY STRUCTURE operation. When the width of a character field is increased, the remainder of the field will be padded with blanks. When the width of a character field is decreased, the field will be truncated to the new width.

Changes of incompatible data types cause the old field contents to be discarded.(e.g. changing from a Date field to a Numeric field.)

See the CREATE command for more details.

Example

use events
modify structure

Products

Recital Terminal Developer

MODIFY VIEW

Class

Terminal Developer Development Tools

Purpose

Full screen modification of a view definition file through a form

Syntax

MODIFY VIEW <vue filename> | (<expC>)

See Also

CREATE VIEW, SET VIEW

Description

The MODIFY VIEW command is a full screen command used to modify an existing view definition file. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then “.vue” is assumed.

See the CREATE VIEW command for full details.

Example

modify view patrons
set view to patrons

Products

Recital Terminal Developer

MOVE WINDOW

Class

Screen Windows

Purpose

Move a pre-defined window to a new position

Syntax

MOVE WINDOW <window-name> TO <row, col> | BY <expN1, expN2> | CENTER | SCREEN

See Also

RESIZE WINDOW, WROWS(), WCOLS()

Description

The MOVE WINDOW command is used to move a window to a new screen position. A window is an area of the screen designated for output and input. Windows are defined with the DEFINE WINDOW command, and are displayed to the screen with the ACTIVATE WINDOW or SHOW WINDOW commands. There is no limit to the number of defined windows. The <window-name> is the name of the window as defined in the DEFINE WINDOW command. A window definition must include the FLOAT keyword for the window to be moveable.

Windows can be moved using the TO <row, col> clause, using the BY <expN1, expN2> clause or centered using the CENTER keyword. The TO clause moves the window so that the upper left corner is positioned at the specified row and column coordinates. The BY clause moves the upper right corner of the window vertically by the number of rows specified in <expN1>, and horizontally by the number of columns specified in <expN2>. The CENTER keyword centers the window in the screen.

The SCREEN keyword is included for language compatibility only.

Example

```
procedure move_up
move window browse by -1,0
return
procedure move_down
move window browse by 1,0
return
procedure move_left
move window browse by 0,-1
return
procedure move_right
move window browse by 0,1
return

set procedure to movewin
set key -1 to move_up
set key -2 to move_down
set key -3 to move_right
set key -4 to move_left
```

Products

Recital Mirage Server, Recital Terminal Developer

NOEXIT

Class

Menus

Purpose

Cause control to remain within a menu after a menu option has been selected

Syntax

NOEXIT

See Also

@...MENU, MENU, MENU AT, MENU COMMAND, MENU FIELDS, MENU FILES, MENU FORMAT, MENU FRAME, MENU QUERY, MENU SCOPE

Description

The NOEXIT command can be used in conjunction with the MENU EXIT command to cause control to remain within the menu after a command is executed. Normally, when the EXIT keyword is used with the MENU command, the menu is exited after a menu item is selected and the associated commands executed. If the NOEXIT command is executed as one of the commands associated with a menu item, then the menu will not be exited.

Example

```
procedure calendar
calendar menu at 2,27
noexit
return
```

Products

Recital Mirage Server, Recital Terminal Developer

NOTE

Class

Applications

Purpose

Comment line

Syntax

NOTE <expC>

See Also

&&, *

Description

The NOTE command allows comment lines to be inserted in programs to enhance their readability and ease of maintenance. The NOTE command can not be placed on the same line as an executable command.

Example

NOTE open the table
use patrons index names

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

ON BAR

Class

Menus

Purpose

Execute a command when a bar is highlighted

Syntax

ON BAR <expN> OF <pop-up> [<command>]

See Also

ACTIVATE POPUP, BAR(), DEACTIVATE POPUP, DEFINE BAR, DEFINE POPUP, MENU FILES LIKE, ON POPUP, ON SELECTION BAR, ON SELECTION POPUP, POPUP(), PROMPT(), SHOW POPUP, SET COMPATIBLE

Description

The ON BAR command is used to specify commands which will execute when the specified bar is highlighted. A bar is a menu option in a Xbase style pop-up menu. These types of menus are created with the DEFINE POPUP and DEFINE BAR commands. The bar is identified by the numeric expression <expN> that was assigned with the DEFINE BAR command.

Used without a specified command, the ON BAR command disables previously specified commands. The command SET COMPATIBLE should be ON when using Xbase style menus. Commands specified by the ON POPUP command execute on highlighted bars that do not have an ON BAR command associated with them.

<command>

The pop-up menu is identified by the name <pop-up> that was assigned with the DEFINE POPUP command. The specified command may be any Recital/4GL command. To specify a command that will execute when a bar is selected, see the ON SELECTION BAR command.

Example

```
define menu main
define pad exit of main;
    prompt "\<Exit" at 0,0
define pad filelist of main;
    prompt "\<Files" at 0,6
on pad filelist of main activate popup files
on selection pad exit of main deactivate menu
```

```
define popup files from 1,06
define bar 1 of files prompt "\<Programs>"
define bar 2 of files prompt "\<Databases>"
on bar 1 of files dialog box "Enter Bar 1"
on bar 2 of files dialog box "Enter Bar 2"
on exit bar 1 of files dialog box "Exit Bar 1"
on exit bar 2 of files dialog box "Exit Bar 2"
activate menu main
```

Products

Recital Mirage Server, Recital Terminal Developer

ON ERROR

Class

Applications

Purpose

Trap program errors

Syntax

ON ERROR [<command>]

See Also

ON ESCAPE, ON KEY, RETRY, ERRNO(), ERROR(), MESSAGE(), SET ONERROR

Description

The ON ERROR command causes the specified <command> to be executed if an error is encountered in a program. If ON ERROR is specified without a <command>, then the default Recital/4GL behavior will be restored. By default, the Recital/4GL will stop execution at the error and an error.mem file will be created

<command>

The <command> can be any Recital/4GL command. After an error is encountered, the ERROR() function will return the error number, and the MESSAGE () function will return the error message. Specifying an '*' as the command causes any errors to be ignored, so should be used with caution. The RETRY command is often used with the ON ERROR trapping facility.

Example

```
procedure badfile
on error
set message to "File does not exist."
return to master
```

```
on error do badfile
use patrons
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

ON ESCAPE

Class

Keyboard Events

Purpose

Trap the interrupt key

Syntax

ON ESCAPE [<command>]

See Also

ON ERROR, ON KEY, SET ESCAPE, SET DOESCAPE

Description

The ON ESCAPE command causes the specified <command> to be executed if the interrupt key is pressed. If ON ESCAPE is specified without a <command>, then the interrupt key will not be trapped.

<command>

The <command> can be any Recital/4GL command.

Example

```
procedure interrupt
on escape
set message to "Processing canceled."
close tables
erase temp.tmp
return to master
```

```
on escape do interrupt
```

Products

Recital Terminal Developer

ON EXIT BAR

Class

Menus

Purpose

Execute a command when the cursor moves off of a pop-up menu bar

Syntax

ON EXIT BAR <expN> OF <pop-up> [<command>]

See Also

ACTIVATE POPUP, BAR(), DEACTIVATE POPUP, DEFINE BAR, DEFINE POPUP, ON BAR, ON POPUP, ON SELECTION BAR, ON SELECTION POPUP, POPUP(), PROMPT(), SHOW POPUP, SET COMPATIBLE

Description

The ON EXIT BAR command is used to specify commands which will execute when the cursor moves off of the specified bar. A bar is a menu option in an Xbase style pop-up menu. These types of menus are created with the DEFINE POPUP and DEFINE BAR commands. The bar is identified by the numeric expression <expN> that was assigned with the DEFINE BAR command. The pop-up menu is identified by the name <pop-up> that was assigned with the DEFINE POPUP command. The command SET COMPATIBLE should be ON when using Xbase style menus.

<command>

The specified command may be any Recital/4GL command. To specify a command that will execute when a bar is selected, see the ON SELECTION BAR command. Used without a specified command, the ON EXIT BAR command disables previously specified commands. Commands specified with the ON EXIT POPUP command execute on highlighted bars that do not have an ON EXIT BAR command associated with them.

Example

```
define menu main
define pad exit of main;
    prompt "\<Exit" at 0,0
define pad filelist of main;
    prompt "\<Files" at 0,6

on pad filelist of main activate popup files
on selection pad exit of main deactivate menu

define popup files from 1,06
define bar 1 of files prompt "\<Programs "
define bar 2 of files prompt "\<Databases"

on bar 1 of files dialog box "Enter Bar 1"
on bar 2 of files dialog box "Enter Bar 2"
on exit bar 1 of files dialog box "Exit Bar 1"
on exit bar 2 of files dialog box "Exit Bar 2"
activate menu main
```

Products

Recital Mirage Server, Recital Terminal Developer

ON EXIT MENU

Class

Menus

Purpose

Execute a command when the cursor exits an Xbase style menu pad

Syntax

ON EXIT MENU <menu name> [<command>]

See Also

ACTIVATE MENU, DEFINE MENU, DEFINE PAD, ON BAR, ON EXIT BAR ON EXIT PAD, ON MENU, ON PAD, ON SELECTION MENU, ON SELECTION PAD, SET COMPATIBLE, SET SCOREBOARD

Description

The ON EXIT MENU command is used to specify commands which will execute when the cursor moves off of menu bars that do not have an ON EXIT PAD command associated with them. A pad is a menu option in a Xbase style menu. These types of menus are created with the DEFINE MENU and DEFINE PAD commands. The menu is identified by the name <menu name> that was assigned with the DEFINE MENU command. The command SET COMPATIBLE should be ON when using Xbase style menus.

<command>

The specified command may be any Recital/4GL command. To specify a command that will execute when an individual pad is selected, see the ON SELECTION PAD command. To specify a command to execute when an individual pad is highlighted, use the ON PAD command. Commands specified by an ON EXIT PAD command override commands specified by the ON EXIT MENU command. Used without a specified command, the ON EXIT MENU command disables previously specified commands.

Example

```
define menu main
define pad exit of main;
    prompt "\<Exit" at 0,0
define pad filelist of main;
    prompt "\<Files" at 0,6
```

```
on menu main dialog box "On Menu"
on exit menu main dialog box "On Exit Menu"
activate menu main
```

Products

Recital Mirage Server, Recital Terminal Developer

ON EXIT PAD

Class

Menus

Purpose

Execute a command when a menu pad is exited

Syntax

ON EXIT PAD <pad name> OF <menu name> [<command>]

See Also

ACTIVATE MENU, DEACTIVATE MENU, DEFINE MENU, DEFINE PAD, ON BAR, ON EXIT BAR, ON EXIT MENU, ON MENU, ON PAD, PAD(), ON SELECTION PAD, SET SCOREBOARD, SET COMPATIBLE

Description

The ON EXIT PAD command is used to specify commands which will execute when the cursor moves off of the specified menu pad. A pad is a menu option in an Xbase style menu. These types of menus are created with the DEFINE MENU and DEFINE PAD commands. The command SET COMPATIBLE should be ON when using Xbase style menus.

<pad-name>

The pad is identified by the name <pad name> that was assigned with the DEFINE PAD command.

OF <menu-name>

The menu is identified by the name <menu name> that was assigned with the DEFINE MENU command.

<command>

The specified command may be any Recital/4GL command. To specify a command that will execute when a pad is highlighted, see the ON SELECTION PAD command. To specify the same exiting command for all pads in a menu, use the ON EXIT MENU command. Commands specified by an ON EXIT PAD command override commands specified by the ON EXIT MENU command. Used without a specified command, the ON EXIT PAD command disables previously specified commands.

Example

```
define menu main
define pad exit of main;
    prompt "\<Exit" at 0,0
define pad filelist of main;
    prompt "\<Files" at 0,6
```

```
on pad filelist of main activate popup files
on selection pad exit of main deactivate menu
on exit pad exit of main dialog box "Exit Pad"
activate menu main
```

Products

Recital Mirage Server, Recital Terminal Developer

ON EXIT POPUP

Class

Menus

Purpose

Execute a command when the cursor moves off of bars in a pop-up Xbase style menu

Syntax

ON EXIT POPUP <pop-up> [<command>]

See Also

ACTIVATE POPUP, BAR(), DEACTIVATE POPUP, DEFINE BAR, DEFINE POPUP, ON BAR, ON POPUP, ON SELECTION BAR, ON EXIT BAR, ON EXIT MENU, ON SELECTION POPUP, POPUP(), PROMPT(), SHOW POPUP, SET COMPATIBLE

Description

The ON EXIT POPUP command is used to specify commands which will execute when the cursor moves off of menu bars. A bar is a menu option in an Xbase style popup menu. These types of menus are created with the DEFINE POPUP and DEFINE BAR commands. The pop-up menu is identified by the name <pop-up> that was assigned with the DEFINE POPUP command. The command SET COMPATIBLE should be ON when using Xbase style menus.

<command>

The specified command may be any Recital/4GL command. To specify a command that will execute when a bar is selected, see the ON SELECTION BAR command. Used without a specified command, the ON EXIT POPUP command disables previously specified commands. Commands specified by an ON EXIT BAR command override commands specified by the ON EXIT POPUP command.

Example

```
define menu main
define pad exit of main;
  prompt "\<Exit" at 0,0
define pad filelist of main;
  prompt "\<Files" at 0,6

on pad filelist of main activate popup files
on selection pad exit of main deactivate menu

define popup files from 1,06
define bar 1 of files prompt "\<Programs>"
define bar 2 of files prompt "\<Databases>"

on popup files dialog box "On Popup"
on exit popup files dialog box "On Exit Popup"
activate menu main
```

Products

Recital Mirage Server, Recital Terminal Developer

ON FINISH

Class

Error Handling and Debugging

Purpose

Execute a command when the Recital process ends

Syntax

ON FINISH <command>

See Also

ERROR(), GETSIG(), ON TERMINATION

Description

The ON FINISH command causes the specified <command> to be executed when the Recital process has finished executing. This can be used in conjunction with the GETSIG() and ERROR() functions to determine if the user exited by themselves or if they received a signal to terminate.

Example

```
procedure on_finish
if getsig() != 0
    dialog box "Signal received. signo = "+alltrim(str(getsig())) label "ON FINISH"
elseif error() > 0
    dialog box "Error received, error = "+alltrim(str(error())) label "ON FINISH"
else
    dialog box "Successful exit" label "ON FINISH"
endif
return

on finish do on_finish
```

Products

Recital Mirage Server, Recital Terminal Developer

ON KEY

Class

Keyboard Events

Purpose

Trap a specified key or any key pressed

Syntax

ON KEY [= <expN> | LABEL <keyname>] [<command>]

See Also

SET PCFKEYS, SET PCKEYS, SET KEY TO, SET KEY...TO

Description

The ON KEY command causes the specified <command> to be executed if a key is pressed. The numeric expression = <expN> uses keycodes to specify which key to trap. The LABEL qualifier uses keynames to specify which key to trap. A specific key that is set to be trapped with the ON KEY command is known as a 'hot key'. When a hot key is pressed from a read or wait state, the specified Recital/4GL <command> is executed. SET PCKEYS must be ON so that the specified keycode is translated to the correct keypad or function key.

<command>

The ON KEY command traps any key, in other words, the next pressed key, when no keycodes or keynames are specified with the command. If ON KEY is specified without a <command>, then no keys will be trapped. The <command> can be any Recital/4GL command.

= <expN>

The optional = <expN> allows a command to be assigned to a particular hot key using keycodes. The following table illustrates the keycode choices and the keys they represent.

Keypad Key	PC Key	Keycode
[1]	F1	315
[2]	F2	316
[3]	F3	317
[4]	F4	318
[5]	F5	319
[6]	F6	320
[7]	F7	321
[8]	F8	322
[9]	F9	323
[0]	F10	324
[CURSOR LEFT]	CURSOR LEFT	331
[CURSOR RIGHT]	CURSOR RIGHT	333
[CURSOR UP]	CURSOR UP	328
[CURSOR DOWN]	CURSOR DOWN	336
[PAGE UP]	PgUp	329
[PAGE DOWN]	PgDn	337

LABEL

The optional LABEL qualifier uses keynames to specify which key to trap. The keynames may be typed in upper, lower, or mixed case. SET PCFKEYS must be ON to enable the use of CTRL-<letter> keynames. The following table illustrates the keynames that may be used with the LABEL qualifier:

Key	Keyname
F1 to F10	F1, F2, F3 ...
CURSOR LEFT	Leftarrow
CURSOR RIGHT	Rightarrow
CURSOR UP	Uparrow
CURSOR DOWN	Downarrow
PAGE UP	PgUp
PAGE DOWN	PgDn
DELETE	Del
INSERT	Ins
TAB	Tab
CTRL A TO CTRL Z	Ctrl-A, Ctrl-B, Ctrl-C

Example

```
// Run mail
procedure mail_procedure
save screen
clear
run mail
clear
restore screen
return
```

```
// Define F10 for mail hot key
set pkeys on
on key = 324 do mail_procedure
```

```
// Define F10 another way
set pkeys on
on key label F10 do mail_procedure
```

```
// Or use Ctrl key
set pkeys on
set pcfkeys on
on key label Ctrl-L do mail_procedure
```

Products

Recital Mirage Server, Recital Terminal Developer

ON MAIL

Class

Input/Output

Purpose

Specify a command to execute when mail messages are trapped.

Syntax

ON MAIL <command>

See Also

!, ALIAS, DO, RUN, SET MAIL, SPAWN, MAIL()

Description

The ON MAIL command causes the specified <command> to execute whenever OpenVMS mail messages are received. The <command> may be any Recital/4GL command. When ON MAIL is specified without a <command>, previously specified commands are cleared. SET MAIL must be ON in order to trap incoming mail messages. When SET MAIL is OFF mail messages are ignored. This command only operates on OpenVMS, UNIX has no way of trapping mail messages.

Example

```
procedure go_mail
save screen
run mail
restore screen
return
```

```
// Go to mail when a message is received
set mail on
on mail do go_mail
use demo
browse
on mail
```

Products

Recital Terminal Developer (OpenVMS only)

ON MENU

Class

Menus

Purpose

Execute a command when a menu pad is highlighted

Syntax

ON MENU <menu name> [<command>]

See Also

ACTIVATE MENU, DEFINE MENU, DEFINE PAD, ON BAR, ON EXIT BAR, ON EXIT MENU, ON EXIT PAD, ON PAD, ON SELECTION MENU, ON SELECTION PAD, SET COMPATIBLE, PAD()

Description

The ON MENU command is used to specify commands which will execute when menu pads that do not have an ON PAD command associated with them are highlighted. A pad is a menu option in an Xbase style menu. These types of menus are created with the DEFINE MENU and DEFINE PAD commands. The menu is identified by the name <menu name> that was assigned with the DEFINE MENU command. The command SET COMPATIBLE should be ON when using Xbase style menus.

<command>

The specified command may be any Recital/4GL command. To specify a command that will execute when an individual pad is selected, see the ON SELECTION PAD command. To specify a command to execute when an individual pad is highlighted, use the ON PAD command. Used without a specified command, the ON MENU command disables previously specified commands. Commands specified by an ON PAD command override commands specified by the ON MENU command.

Example

```
define menu main
define pad exit of main;
  prompt "\<Exit" at 0,0
define pad filelist of main;
  prompt "\<Files" at 0,6
```

```
on menu main dialog box "On Menu"
on exit menu main dialog box "On Exit Menu"
activate menu main
```

Products

Recital Mirage Server, Recital Terminal Developer

ON MOUSE

Class

Mirage Forms

Purpose

Execute a command when a menu operation occurs

Syntax

ON MOUSE [<command>]

See Also

INKEY(), LASTKEY(), MCOL(), MROW()

Description

The ON MOUSE command is used to specify a command which will be executed when a mouse operation occurs. Recital Mirage traps mouse operations using the INKEY(<expN>,"M") function. The INKEY() or LASTKEY() functions can be used to determine the operation which took place and the MCOL() and MROW() functions return the current mouse column and row position on the screen.

<command>

The specified command may be any Recital/4GL command and may be a call to a function or to DO a procedure. Used without a specified command, the ON MOUSE command disables previously specified commands.

In Recital Terminal Developer the ON MOUSE command is ignored.

Example

```
procedure on_mouse
m_colpos = mcol()
m_rowpos = mrow()
m_mouseop = lastkey()
// process based on position and operation
return
```

```
on mouse do on_mouse
inkey(0,"M")
```

Products

Recital Mirage Server, Recital Terminal Developer

ON PAD

Class

Menus

Purpose

Define the pop-up menu selection for dBASE-IV style menus

Syntax

ON PAD <pad name> OF <menu name> [<command>]

See Also

DEFINE MENU, DEFINE PAD, ON BAR, ON EXIT BAR, ON EXIT MENU, ON MENU, ON POPUP, ON SELECTION MENU, ON SELECTION PAD, PAD(), SET SCOREBOARD, SET COMPATIBLE ON

Description

The ON PAD command is used to associate a command with a pad in an Xbase style menu. The pad, which must be defined with the DEFINE PAD command, is specified with <pad name>. The menu which that pad belongs to must be specified with <menu name>.

Typically the ON PAD command is used with the ACTIVATE POPUP command. This activates a pop-up menu when the menu pad is highlighted. Other commands may be used, or a call to a program or procedure may be made.

To specify the same command to execute on all pads, use the ON MENU command. To specify a command to execute when the pad is selected, see the ON SELECTION PAD command.

Example

```
define menu main
define pad exit of main;
    prompt "\<Exit" at 0,0
define pad filelist of main;
    prompt "\<Files" at 0,6
```

```
on pad filelist of main activate popup files
on selection pad exit of main deactivate menu
on exit pad exit of main dialog box "Exit Pad"
activate menu main
```

Products

Recital Mirage Server, Recital Terminal Developer

ON PAGE

Class

Printing

Purpose

Execute a command when the printed output reaches a specified line number.

Syntax

ON PAGE [AT LINE <expN> <command>]

See Also

EJECT PAGE, ON()

Description

The ON PAGE command is used to specify commands which will execute when the printed output reaches a line number specified by <expN>. The ON PAGE command without any <command> clears the previous ON PAGE.

AT LINE <expN> <command>

The specified command may be any Recital/4GL command.

Example

on page at line 50 do footnote

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

ON POPUP

Class

Menus

Purpose

Execute a command when a bar is highlighted

Syntax

ON POPUP <pop-up> [<command>]

See Also

ACTIVATE POPUP, BAR(), DEACTIVATE POPUP, DEFINE BAR, DEFINE POPUP, ON EXIT BAR, ON EXIT POPUP, ON POPUP, ON SELECTION BAR, ON SELECTION POPUP, POPUP(), PROMPT(), SHOW POPUP, SET COMPATIBLE

Description

The ON POPUP command is used to specify commands which will execute when menu bars that do not have an ON BAR command associated with them are highlighted. A bar is a menu option in an Xbase style popup menu. These types of menus are created with the DEFINE POPUP and DEFINE BAR commands. The pop-up menu is identified by the name <pop-up> that was assigned with the DEFINE POPUP command. The command SET COMPATIBLE should be ON when using Xbase style menus.

<command>

The specified command may be any the Recital/4GL command. To specify a command that will execute when a bar is selected, see the ON SELECTION BAR command. Used without a specified command, the ON POPUP command disables previously specified commands. Commands specified by an ON BAR command override commands specified by the ON POPUP command.

Example

```
define menu main
define pad exit of main;
    prompt "\<Exit" at 0,0
define pad filelist of main;
    prompt "\<Files" at 0,6

on pad filelist of main activate popup files
on selection pad exit of main deactivate menu

define popup files from 1,06
define bar 1 of files prompt "\<Programs>"
define bar 2 of files prompt "\<Databases>"

on popup files dialog box "On Popup"
on exit popup files dialog box "On Exit Popup"
activate menu main
```

Products

Recital Mirage Server, Recital Terminal Developer

ON READERROR

Class

Screen Forms

Purpose

Execute specified command when a data entry error occurs

Syntax

ON READERROR [<command>]

See Also

@...GET, APPEND, BROWSE, CHANGE, EDIT, ONERROR, READ

Description

The ON READERROR command executes the specific <command> when a data entry error is made. Data entry errors are: invalid dates, number out of range, failed VALID condition. ON READERROR can be used to replace the default error messages that are given when these errors occur. If no <command> is specified with the ON READERROR command, any previously issued ON READERROR setting is cancelled.

Example

```
procedure readtrap
dialog box "Date is out of range"
return
```

```
dToday = date()
on readerror do readtrap
@10,00 say "Enter Date" get dToday;
    range {01/01/2000},{01/01/2005}
read
on readerror
```

Products

Recital Mirage Server, Recital Terminal Developer

ON SELECTION BAR

Class

Menus

Purpose

Defines commands for pop-up menu selection

Syntax

ON SELECTION BAR <expN> OF POPUP <popup> [<command>]

See Also

ACTIVATE POPUP, DEACTIVATE POPUP, DEFINE BAR, DEFINE POPUP, ON BAR, ON EXIT BAR, ON EXIT POPUP, ON POPUP, ON SELECTION POPUP, SHOW POPUP, SET COMPATIBLE

Description

The ON SELECTION BAR command defines a command that will be executed when the specified menu bar is selected. A menu bar is a selection option in dBASE-IV style pop-up menus. Pop-up menus of this type are created with the DEFINE POPUP and DEFINE BAR commands. The bar is identified by the numeric expression <expN> that was assigned with the DEFINE BAR command. The pop-up is identified by its name which was assigned with the DEFINE POPUP command. The command SET COMPATIBLE should be ON when using dBASE IV style menus.

<command>

To specify commands which execute when the menu bar is highlighted, use the ON BAR command. To specify commands that execute when the menu bar is exited, use the ON EXIT BAR command.

Example

```
define menu main
define pad exit of main;
    prompt "\<Exit" at 0,0
define pad filelist of main;
    prompt "\<Files" at 0,6

on pad filelist of main activate popup files
on selection pad exit of main deactivate menu

define popup files from 1,06
define bar 1 of files prompt "\<Programs>"
define bar 2 of files prompt "\<Databases>"

on selection bar 1 of popup files;
    dialog box "Bar 1"
on selection bar 2 of popup files;
    dialog box "Bar 2"
activate menu main
```

Products

Recital Mirage Server, Recital Terminal Developer

ON SELECTION MENU

Class

Menus

Purpose

Define actions performed on every pad in a dBASE-IV style menu

Syntax

ON SELECTION MENU <menu> [<command>]

See Also

ACTIVATE POPUP, DEACTIVATE POPUP, DEFINE BAR, DEFINE POPUP, ON BAR, ON EXIT BAR, ON EXIT POPUP, ON POPUP, ON SELECTION POPUP, ON SELECTION PAD, SET COMPATIBLE

Description

The ON SELECTION MENU command defines actions performed on pads that do not have an ON SELECTION PAD command associated with them. Pads are options in Xbase style menus. These types of menus are created with the DEFINE MENU and DEFINE PAD commands. The menu is identified with the name that was assigned with the DEFINE MENU command. Pad selection occurs when the user moves the highlight onto the menu prompt and presses the [RETURN] key. The command SET COMPATIBLE should be set ON when using Xbase style menus.

<command>

If no command is given, ON SELECTION MENU disables previously specified commands. To specify the same command for individual pads in a menu, use the ON SELECTION PAD command. To specify commands which will execute when pads are highlighted, use the ON PAD command.

Example

```
define menu main
define pad exit of main;
  prompt "\<Exit" at 0,0
define pad filelist of main;
  prompt "\<Files" at 0,6
```

```
on selection menu main dialog box prompt()
```

```
activate menu main
```

Products

Recital Mirage Server, Recital Terminal Developer

ON SELECTION PAD

Class

Menus

Purpose

Define actions performed on pad selection in a dBASE-IV style menu

Syntax

ON SELECTION PAD <pad> OF <menu> [<command>]

See Also

ACTIVATE MENU, DEACTIVATE MENU, DEFINE MENU, DEFINE PAD, MENU(), ON EXIT PAD, ON PAD, PAD(), PROMPT(), ON SELECTION MENU, SET COMPATIBLE

Description

The ON SELECTION PAD command defines actions performed on a pad selection within a dBASE-IV style menu. The pad and menu are identified with the names that were assigned with the DEFINE MENU and DEFINE PAD commands. Pad selection occurs when the user moves the highlight onto the menu prompt and presses the [RETURN] key. The command SET COMPATIBLE should be set ON when using Xbase style menus.

<command>

If no command is given, ON SELECTION PAD disables previously specified commands. To specify the same selection command for all pads in a menu, use the ON SELECTION MENU command.

Example

```
define menu main
define pad exit of main;
    prompt "\<Exit" at 0,0
define pad filelist of main;
    prompt "\<Files" at 0,06
```

```
on pad filelist of main activate popup files
on selection pad exit of main deactivate menu
```

```
activate menu main
```

Products

Recital Mirage Server, Recital Terminal Developer

ON SELECTION POPUP

Class

Menus

Purpose

Defines actions for Xbase style pop-up menu selection

Syntax

ON SELECTION POPUP <popup> | ALL [BLANK] [<command>]

See Also

DEFINE MENU, DEFINE PAD, ON PAD, ON SELECTION PAD, ACTIVATE MENU, DEACTIVATE MENU, RELEASE MENUS, SHOW MENU, DEFINE POPUP, DEFINE BAR, ACTIVATE POPUP, DEACTIVATE POPUP, RELEASE POPUPS, CLEAR POPUPS, SHOW POPUP

Description

The ON SELECTION POPUP command defines actions for dBASE IV style pop-up menu selections. These types of pop-up menus are created with the DEFINE POPUP and DEFINE BAR commands. The pop-up is identified using the name which was assigned with the DEFINE POPUP command. The command SET COMPATIBLE should be ON when using dBASE-IV style menus.

ALL

The ALL clause will apply the specified <command> to all pop-up menus.

BLANK

The BLANK keyword clears the active pop-up menu from the screen before executing the specified commands.

<command>

The command to be run when the specified popup is selected. If no command is specified, the popup is reset.

Example

```
define menu main
define pad exit of main;
    prompt "\<Exit" at 0,0
define pad filelist of main;
    prompt "\<Files" at 0.06
on pad filelist of main activate popup files
on selection pad exit of main deactivate menu
```

```
define popup files from 1,06
define bar 1 of files prompt "\<Programs"
define bar 2 of files prompt "\<Databases"
define bar 3 of files prompt "\<Reports"
define bar 4 of files prompt "\<Screens"
on selection popup files dialog box prompt()
activate menu main
```

Products

Recital Mirage Server, Recital Terminal Developer

ON TERMINATION

Class

Environment

Purpose

Execute a command when Recital receives a hang-up or kill signal

Syntax

ON TERMINATION [<command>]

See Also

ON TIMEOUT, ON()

Description

When the Recital/4GL receives a hang-up or kill signal, the program cleans up its environment and terminates. The ON TERMINATION command is used to specify commands which will execute when Recital receives the signal. The ON TERMINATION command without any <command> clears the previous ON TERMINATION.

<command>

The specified command may be any Recital/4GL command.

Example

on termination do exitfile

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

ON TIMEOUT

Class

Keyboard Events

Purpose

Execute a command when the timeout event flag is triggered

Syntax

ON TIMEOUT [<command>]

See Also

SET TIMEOUT, SET CLOCK, SET CLOCKDISPLAY

Description

The ON TIMEOUT command is used to specify a single command that will execute when the timeout event flag has been triggered. The timeout period is defined by the SET TIMEOUT command. Issuing an ON TIMEOUT statement without a <command> clears the previous ON TIMEOUT. SET CLOCK ON must be active for the timeout command to function, but the SET CLOCKDISPLAY command can be set to OFF if no visible clock is required.

<command>

The specified command may be any the Recital/4GL command.

Example

```
procedure p_timeout
    dialog box "Timeout Occurred"
    set timeout off
    quit
return

set clock on
on timeout do p_timeout
set timeout to 20
set timeout on
m_var = space(10)
@ 00,00 say "Enter Value:" get m_var
read
return
```

Products

Recital Mirage Server, Recital Terminal Developer

PACK

Class

Fields and Records

Purpose

Remove records from the active table that are marked for deletion

Syntax

PACK [ALL]

See Also

DELETE, RECALL, REINDEX, ZAP, SET DELETED, DELETED()

Description

The PACK command removes all the records from the active table that are marked for deletion, and frees the disk space that they occupied. If the active table is indexed, the index is updated as the records are removed from the table. Exclusive use of the table is required for the PACK operation.

ALL

If the ALL option is specified, the PACK command will start at workarea 1 and PACK every open table in all workareas.

Example

```
use diary
? reccount()
  800
set talk on
delete all for date < date()
100 record(s) deleted.
pack
Pack complete, 700 records copied.
? reccount()
  700
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

PACK DATABASE

Class

Databases

Purpose

Packs each table in the active database or packs the catalog and rebuilds the catalog index tags for a specified database

Syntax

PACK DATABASE [<database name> | ?]

See Also

ALTER INDEX, ALTER TABLE, BACKUP DATABASE, CLOSE DATABASES, CLOSE TABLES, COMPILE DATABASE, CREATE DATABASE, CREATE TABLE, CREATE INDEX, CREATE VIEW, DISPLAY DATABASE, DISPLAY INDEXES, DISPLAY TABLES, DROP DATABASE, DROP INDEX, DROP TABLE, INDEX, LIST DATABASE, LIST INDEXES, LIST TABLES, OPEN DATABASE, PACK, REBUILD DATABASE, RESTORE DATABASE, USE, SET AUTOCATALOG, SET EXCLUSIVE, ADATABASES(), DBUSED(), GETENV(), DB_MAXWKA

Description

The PACK DATABASE command packs all the tables in the active database. Packing a table removes all records previously marked for deletion from the table.

Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directories are sub-directories of the Recital data directory. The environment variable / symbol DB_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality. Databases are opened using the OPEN DATABASE command.

If the <database name> is specified, the PACK DATABASE command will operate on the specified database's catalog file: the catalog file will be packed and its index tags rebuilt using INDEX ON. If the question mark, '?', is included instead of the <database name>, the 'SELECT A FILE' dialog will be displayed, allowing the user to select the database. The dialog defaults to the DB_DATADIR directory.

Example

```
VFP/SQL>open database southwind
VFP/SQL>pack database
VFP/SQL>close databases
VFP/SQL>pack database southwind
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

PARAMETERS

Class

Applications

Purpose

Declare formal parameters to a procedure or program

Syntax

PARAMETERS <parameter list>

See Also

DO, LOCAL, LPARAMETERS, PUBLIC, PRIVATE, PROCEDURE, SET CLIPPER, SET PROCEDURE, DECLARE, &, SET PROCEDURE ADDITIVE

Description

The PARAMETERS command declares a list of private memory variables or arrays, and assigns them the values of the actual parameters specified on a DO <program | procedure> WITH command. The parameters are initially declared as logicals with the value .F.. The PARAMETERS command must be the first executable command in a procedure or program. The PCOUNT() function is used to determine how many parameters were passed.

Parameters may be passed which are memory variables (i.e. they are not part of an expression). The contents of these memory variables will be updated when the procedure or program returns. This type of parameter passing is known as call by reference. This is the default for Recital/4GL with PROCEDURES and PROGRAMS. The '@' character may be placed in front of the memory variable name in User Defined Functions (UDF), so that they are called by reference.

If you do not wish the parameters to be modified by the called PROCEDURE or PROGRAM, you should enclose the memory variable in round brackets. This type of parameter passing is known as call by value. Any expressions that you specify as parameters are always call by value parameters. The default passing of parameters with User Defined Functions (UDF) is call by value. If COMPATIBLE is set ON then the parameters will be passed by reference. The limit to the number of parameters that you can pass is 40.

The private memory variables created by the PARAMETERS command are always released when the procedure or program returns. If CLIPPER is set ON and not all parameters are passed, the variables in the PARAMETERS command not passed will be defined as type 'U' instead of .F..

Example

```
procedure add
parameters para1, para2
result = para1+ para2
return
```

```
private result
do add with 10, 40
? result
50
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

POP KEY

Class

Keyboard Events

Purpose

Restores ON KEY LABELS that were placed on the stack with PUSH KEY

Syntax

POP KEY [ALL]

See Also

PUSH KEY, ON KEY LABEL, ON()

Description

The POP KEY command is used to recover ON KEY LABEL command settings from the stack following a corresponding PUSH KEY command. SET COMPATIBLE should be set to ON when using this command.

ALL

Using the ALL clause will clear all currently active key assignments defined with ON KEY LABEL as well as all key assignments from the stack that were defined with ON KEY LABEL.

Example

```
clear all
on key label f10 do proc1
push key clear
on key label f10 do proc2
read
return

procedure proc1
@ 5,5 say "PROC 1 -- Popped The On Key Label"
return

procedure proc2
@ 6,6 say "PROC 2 -- Resetting The On Key Label"
pop key
return
```

Products

Recital Mirage Server, Recital Terminal Developer

POP MENU

Class

Menus

Purpose

Retrieve a menu bar definition from the stack

Syntax

POP MENU <MENU> [TO MASTER]

See Also

ACTIVATE MENU, DEFINE MENU, POP POPUP, PUSH MENU, PUSH POPUP

Description

POP MENU recovers the specified <menu> from the stack following a corresponding PUSH MENU command. Menus are always placed on and removed from the stack in a last in, first out order. SET COMPATIBLE should be set to ON when using this command.

TO MASTER

Using the TO MASTER clause will restore the first menu pushed onto the stack and then clear all others from the stack.

Example

```
push menu _msysmenu
set sysmenu to _mfile, _medit
pop menu _msysmenu
```

Products

Recital Mirage Server, Recital Terminal Developer

POP POPUP

Class

Menus

Purpose

Restore specified popup from the stack

Syntax

POP POPUP <popup>

See Also

ACTIVATE POPUP, DEFINE POPUP, PUSH POPUP

Description

POP POPUP recovers the specified <popup> from the stack following a corresponding PUSH POPUP command. Popup definitions are always placed on and removed from the stack in a last in, first out order. SET COMPATIBLE should be set to ON when using this command.

Example

```
define popup poptest from 5,5
define bar 1 of poptest prompt "one"
define bar 2 of poptest prompt "two"
define bar 3 of poptest prompt "three"
define bar 4 of poptest prompt "four"
activate popup poptest nowait
push popup poptest
wait "Popup Pushed" window
release bar 2 of poptest
wait "This Is The Modified Popup" window
pop popup poptest
wait "Popup Popped, Original Popup Restored" window
deactivate popup poptest
release popup poptest
```

Products

Recital Mirage Server, Recital Terminal Developer

PRINT

Class

Printing

Purpose

Print a text file on the printer

Syntax

PRINT <.txt filename> | (<expC>)

See Also

TYPE, RUN, SPAWN, PUTENV()

Description

The PRINT command prints the specified <.txt filename> on the printer. The filename can be substituted with an <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then '.txt' is assumed. If the file extension is '.tmp' the PRINT command will delete the file after printing it.

The Recital/4GL login file contains an environment variable called DB_PRINT which points to a file in the root directory called print.<operating system name>. It contains the commands used by the host operating system to print files

The print.<operating system name> file can be customized for your own environment. Multiple print files can be set up and assigned by changing the DB_PRINT environment variable with the PUTENV() function. This allows particular printer files to be associated with a specified group of users, or particular printers.

Example

```
// Print 4 copies of a report
m_filenam = sys(3) = ".txt"
report form analysis to file &m_filenam
for i=1 to 4
    print &m_filenam
next
erase &m_filenam
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

PRIVATE

Class

Memory Variables

Purpose

Declare memory variables private to a procedure or program

Syntax

PRIVATE <memvar> | <memvar-list> | <array> | ALL [EXCEPT | LIKE <skeleton>]

See Also

PARAMETERS, DO, PUBLIC, DISPLAY MEMORY, SET CLIPPER, LOCAL

Description

The PRIVATE command provides a facility for declaring memory variables or arrays which are local to a procedure or program. When the procedure or program returns, all of the memory variables or arrays that were declared by PRIVATE, are released. The memory variables are initially declared as logicals with the value .F., unless CLIPPER is set ON in which case they are defined as 'U'.

You can declare PRIVATE memory variables with the same name as other memory variables, which were declared at lower levels. Any procedures or programs that are called can access these private memory variables. Any memory variables or arrays that need to be accessed globally should be declared using the PUBLIC command. By default any memory variables declared at the Recital Terminal Developer development prompt, are declared as PUBLIC memory variables, and any others are declared as PRIVATE memory variables.

The ALL option allows you to define all current memory variables as private. The ALL EXCEPT <skeleton> allows the user to define all the current memory variables that do not match the wildcard <skeleton> as private. The ALL LIKE <skeleton> option allows you to define all the current memory variables that match the wildcard <skeleton> specification as private. PRIVATE ALL LIKE <skeleton> and ALL EXCEPT <skeleton> will privatize the memory variables that exist as public when used in a lower level procedure.

See DECLARE or DIMENSION for more details on array declaration.

Example

```
private i,j,k
? j
.F.
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

PROCEDURE

Class

Applications

Purpose

Declare a procedure

Syntax

PROCEDURE <procedure name>

See Also

COMPILE, DEBUG, DO, ENDPROC, FUNCTION, LINK, PARAMETERS, PRIVATE, PUBLIC, RETURN, ACC(), CALC(), SET PROCEDURE

Description

The PROCEDURE statement marks the beginning of a procedure. A procedure declaration is terminated with a RETURN statement. You may have other RETURN statements within the procedure body, provided that they are properly nested between IF...ENDIF, DO WHILE ...ENDDO, or DO CASE ...ENDCASE. Procedure names must be unique within the first 32 characters. They must begin with a letter or underscore, followed by any combination of letters (A-Z), digits (0-9), and underscores (_), and may not contain any blanks or spaces.

You can include procedures in your program files, as well as in procedure library files. If a procedure is included in a program file, then it must be defined before it is used. For example, if you issue a command DO MAIN, then the command PROCEDURE MAIN must be on a line before the line containing the DO MAIN statement. You make all the procedures in a procedure library file known to the Recital/4GL by using the SET PROCEDURE TO command.

To execute a procedure, you just issue a DO <procedure name> command, as if you were calling a program file, or you can use the procedure as a user defined function (UDF) and pass parameters to it as with any standard Recital function. The limit to the number of parameters that you can pass is 40.

Note that procedures can be called using a DO statement such as:

do procname with para1, para2, para3

or they can be called as functions without assigning a return value, as in:

procname(para1,para2,para3)

There is no limit to the number of procedures that can be declared in the Recital/4GL. The commands LIST | DISPLAY PROCEDURE will list all currently active functions and procedures.

If SET COMPATIBLE is set to VFP, the PROCEDURE declaration can be terminated with an ENDPROC command rather than a RETURN. It will also be terminated when another PROCEDURE or FUNCTION command is reached.

If a procedure library contains a procedure with the same name as the containing library, a call to that name will run the procedure if SET COMPATIBLE is set to VFP or FOXPRO.

Example

```
// File: finance.prg  
procedure payment2  
parameters pmt  
return pmt
```

```
procedure future  
...  
return
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

PUBLIC

Class

Memory Variables

Purpose

Declare public memory variable

Syntax

PUBLIC [<memvar-list>] | [<array>]

See Also

PRIVATE, PARAMETERS, DECLARE, SET COMPATIBLE

Description

The PUBLIC statement declares global memory variables or arrays. Memory variables or arrays that have been declared PUBLIC can be shared and modified by all procedures and programs. By default, any variables created at the Recital Terminal Developer development prompt are declared PUBLIC. A PUBLIC memory variable is initially declared as a logical, with a value of .F.. See DECLARE or DIMENSION for more details on array declaration.

Example

```
public i, j, k
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

PUSH KEY

Class

Keyboard Events

Purpose

Puts ON KEY LABEL settings on the stack

Syntax

PUSH KEY [CLEAR]

See Also

POP KEY, ON KEY LABEL, ON()

Description

The PUSH KEY command is used to place all ON KEY LABEL command settings on a stack in memory. These key assignments are then effectively saved and can then be changed while still allowing the original assignments to be restored with the POP KEY command. SET COMPATIBLE should be set to ON when using this command.

CLEAR

Including the CLEAR clause saves all current ON KEY LABEL assignments to the stack and then clears the current assignments.

Example

```
clear all
on key label f10 do proc1
push key clear
on key label f10 do proc2
read
return

proc proc1
@ 5,5 say "PROC 1 -- Pop'd The On Key Label"
return

proc proc2
@ 6,6 say "PROC 2 -- Resetting The On Key Label"
pop key
return
```

Products

Recital Mirage Server, Recital Terminal Developer

PUSH MENU

Class

Menus

Purpose

Place a menu bar definition on the stack

Syntax

PUSH MENU <MENU>

See Also

ACTIVATE MENU, DEFINE MENU, POP POPUP, POP MENU, PUSH POPUP

Description

PUSH MENU is used to place the specified <menu> on the stack. The menu definition can then be changed while still allowing the original definition to be restored with POP MENU. Menus are always placed on and removed from the stack in a last in, first out order. SET COMPATIBLE should be set to ON when using this command.

Example

```
push menu _msysmenu
set sysmenu to _mfile, _medit
pop menu _msysmenu
```

Products

Recital Mirage Server, Recital Terminal Developer

PUSH POPUP

Class

Menus

Purpose

Puts a popup definition on the stack

Syntax

PUSH POPUP <popup>

See Also

ACTIVATE POPUP, DEFINE POPUP, POP POPUP

Description

PUSH POPUP places the specified <popup> on the stack. This allows popup definitions to be changed while still allowing the original definition to be recovered via the POP POPUP command. Popup definitions are always placed on and removed from the stack in a last in, first out order. SET COMPATIBLE should be set to ON when using this command.

Example

```
define popup poptest from 5,5
define bar 1 of poptest prompt "one"
define bar 2 of poptest prompt "two"
define bar 3 of poptest prompt "three"
define bar 4 of poptest prompt "four"
```

```
activate popup poptest nowait
push popup poptest
wait "Popup Pushed" window
```

```
release bar 2 of poptest
wait "This Is The Modified Popup" window
```

```
pop popup poptest
wait "Popup Popped, Original Popup Restored" window
```

```
deactivate popup poptest
release popup poptest
```

Products

Recital Mirage Server, Recital Terminal Developer

QUERY

Class

Screen Forms

Purpose

Full screen viewing of records in a table through a form

Syntax

QUERY [<scope>]
[FIELDS <field list>]
[FOR <condition>]
[KEY <exp>]
[NOCLEAR]
[WHILE <condition>]

See Also

APPEND, EDIT, BROWSE, CHANGE, INSERT, READ, SET FORMAT, SET KEY, SET UPDATE, SET QUERYMODE, @...GET, @...MENU, CREATE SCREEN, SET MOUSE, SET NAVIGATE, FMT(), RESTORE SCREEN, SAVE SCREEN, LABEL

Description

The QUERY command provides a read only version of the EDIT and CHANGE commands. Unless the SET FORMAT TO <.fmt filename> command has been issued, QUERY will use the default form. You can design your own forms in the Forms Designer using the command CREATE SCREEN. The Forms Designer will automatically generate a format file that contains @...SAY...GET commands. This form can be used at any time with the database by issuing the SET FORMAT TO command.

Keyword	Description
<scope>	If no <scope> is specified, QUERY is activated on the current record and all records are accessible using the [NEXT RECORD] and [PREVIOUS RECORD] keys.
FIELDS <field list>	The active fields can be restricted to those specified in the comma separated <field list>.
FOR <condition>	Record navigation is restricted to those records that match the <condition>.
KEY <exp>	The active records can be restricted to those that match the specified <exp>. The <exp> must be based on the index key of the current master index.
NOCLEAR	Erasing of the screen on entry and exit from QUERY is disabled.
WHILE <condition>	Record navigation is restricted to those records that match the specified <condition>. Navigation cannot continue beyond a record that does not match the <condition>.

If SET MOUSE is ON, cursor keys will move the cursor anywhere on the screen rather than just from field to field. If SET NAVIGATE is ON, the cursor moves to fields following the direction specified by the key being pressed, rather than following the order of the GETS on the form. When the RETURN key is pressed, the cursor moves to the nearest field when SET NAVIGATE is ON.

The following keys are active in QUERY:

Key	Action
ABANDON	Exit from the form
CURSOR DOWN	Skip to next field
CURSOR LEFT	Skip to previous field

CURSOR RIGHT	Skip to next field
CURSOR UP	Skip to previous field
EXIT/SAVE	Exit from the form
FIND	Specify search key/condition
FIND NEXT	Search for next key/condition as specified by FIND
HELP	Activate pop-up help menu (field list)
MENUBAR	Activate the QUERY menu bar
NEXT RECORD	Skip to next record
PREV RECORD	Skip to previous record
REFRESH	Redraw the form
TAB	Toggle key identification menu on and off

The following menu options are available from the QUERY menu bar in the default form:

Menu Item	Action
Descriptions	Toggle the field descriptions on and off
Top	Position to the top of the database
Bottom	Position to the bottom of the database
Order	Select index file order
Help	Activate on-line help system

Example

set format to myscreen
query all for event = "BALLET"

Products

Recital Mirage Server, Recital Terminal Developer

QUIT

Class

Applications

Purpose

Terminate session

Syntax

QUIT [TO <command>]
[WITH <expN>]

See Also

CLOSE, RUN

Description

The QUIT command closes all open tables, associated index and format files, open program files, the procedure library, and returns control to the operating system.

TO <command>

The optional TO <command> allows an operating system command to be executed after exiting from the Recital session.

WITH <expN>

The WITH clause is used to specify an integer from -1 to 254 to be returned upon exiting. This is useful as an exit code that can be checked by calling programs, command files, or shell scripts.

Example

```
// Quit if [ABANDON] was pressed
if lastkey()=ctrl('g')
    quit
endif
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

READ

Class

Screen Forms

Purpose

Activate objects created with @...GET commands

Syntax

READ
[COLOR <standard/enhanced>]
[COLOR SCHEME <color scheme> | COLOR <color code>]
[CYCLE]
[LOCK | NOLOCK]
[NOCLEAR]
[NOMOUSE]
[NOREFRESH]
[NOUPDATE]
[SAVE]
[TIMEOUT <expN1>]
[VALID <expL1>]
[WHEN <expL2>]

See Also

@, SET FORMAT, EDIT, CHANGE, APPEND, QUERY, INSERT, CREATE SCREEN, SET MOUSE, SET NAVIGATE

Description

The READ command activates all objects created with the @...GET command. If no @...GETS are pending, the READ command waits for any key to be pressed. Normally, the READ command will clear the GETS after screen input has completed. A READ can be terminated in several ways. Moving forward past the last object or backward past the first object, if the CYCLE clause is not specified, pressing the [EXIT/SAVE] key, or selecting an object that was designed to terminate the READ. Calling an event procedure from an active object and issuing another READ allows for the creation of nested READs. The READ commands can be nested to a depth of five levels.

If the GET fields that you have specified are fields from a table as opposed to memory variables, then the Recital/4GL will automatically lock all records in the designated tables, if the active table is opened for shared use.

If SET MOUSE is ON, cursor keys will move the cursor anywhere on the screen rather than just from field to field. If SET NAVIGATE is ON, the cursor moves to fields following the direction specified by the key being pressed, rather than following the order of the GETS on the form. When the RETURN key is pressed, the cursor moves to the nearest field when SET NAVIGATE is ON.

Note: If an attempt is made to exit from a form by pressing the [EXIT/SAVE] key, and some of the fields below the current one have the MUST_ENTER attribute, then Recital will beep, indicating an error and position the cursor on the MUST_ENTER field.

COLOR <standard/enhanced>

COLOR SCHEME <color scheme> | COLOR <color code>

The color of the current @...GET can be set by including the number of an existing color scheme using the COLOR SCHEME syntax or by including a color pair using the COLOR <standard/enhanced> syntax.

CYCLE

The CYCLE keyword prevents the READ from being terminated when you move forward past the last object, or backwards past the first object. Instead, if you are on the last object and you move forward, the first object gets focus and if you are on the first object and move backwards past the first object then the last object gets focus.

LOCK | UNLOCK

The LOCK keyword locks all records accessed from the READ. The UNLOCK keyword makes all fields accessed by the READ read-only.

NOCLEAR

The NOCLEAR keyword disables the erasing of the screen when a format file is processed.

NOMOUSE

The NOMOUSE keyword only applies to Recital Mirage applications. It forces data entry to occur in the order of the @...GETs on the screen as it would in a Recital Terminal Developer environment.

NOREFRESH

The NOREFRESH keyword prevents the initial contents of the GET fields from being displayed. It is most useful when you have a lot of CALCULATED BY fields.

NOUPDATE

The NOUPDATE option prevents updating of the index files. This should only be used if no fields are updated which affect the index key.

SAVE

The SAVE keyword causes the GETS to remain active, so that subsequent reads can be executed in a loop, without having to reissue the @...GETs.

TIMEOUT <expN1>

The TIMEOUT clause determines how long in seconds, <expN1>, the READ will wait for user input before exiting automatically.

VALID <expL1>

The VALID clause causes <expL1> to be evaluated before the READ is exited. If <expL1> evaluates to true (.T.), the read is exited, if <expL1> evaluates to false (.F.), the read remains active.

WHEN <expL2>

The WHEN clause determines if the READ is activated. The expression <expL2> must evaluate to true for the READ to be activated, otherwise the READ command is ignored.

Example

```
mvar = "xxx"  
@10,10 GET mvar  
read
```

Products

Recital Mirage Server, Recital Terminal Developer

READ MENU TO

Class

Menus

Purpose

Activate a FoxBASE+ style pop-up menu

Syntax

READ MENU TO <memvar>

[SAVE]

[TIMEOUT <expN>]

See Also

@...MENU, READ MENU BAR TO, MENU BAR, MENU

Description

The READ MENU TO activates a pop-up menu defined by the @...MENU command. The <memvar> is a variable with the initial starting position of the highlighting bar. This <memvar> returns the selected menu item number.

SAVE

The menu options are not released and can be activated by another READ MENU TO command.

TIMEOUT <expN>

The TIMEOUT clause determines how long in seconds, <expN>, the READ MENU TO will wait for user input before exiting automatically.

Example

```
// Define menu
dimension choice(3,1)
choice(1) = "Edit"
choice(2) = "Delete"
choice(3) = "Add"
m_choice = 0
@ 5,0 menu choice,3 title "Choices"
// Activate menu
read menu to mchoice
```

Products

Recital Mirage Server, Recital Terminal Developer

READ MENU BAR TO

Class

Menus

Purpose

Activate a FoxBASE+ style pull-down menu system

Syntax

READ MENU BAR TO <memvar1>,<memvar2>

[SAVE]

[TIMEOUT <expN>]

See Also

READ MENU TO, MENU BAR, MENU

Description

The READ MENU BAR TO command activates a pulldown menu system. The <memvar1> and <memvar2> variables are used to position the light bar menu on the required pulldown. On selection of a pulldown item, the variables return the item selected. The menu clears from the screen after a menu selection is made, unless the SAVE option is specified. The SAVE option causes the menu to remain on the screen after a menu selection is made.

As this is the last step in creating a pulldown menu system, menu arrays and menu bars must be initialized and installed prior to using the READ MENU BAR TO command.

SAVE

The menu options are not released and can be activated by another READ MENU BAR TO command.

TIMEOUT <expN>

The TIMEOUT clause determines how long in seconds, <expN>, the READ MENU BAR TO will wait for user input before exiting automatically.

Example

```
// Install the pull-down menu system
menu bar top,3
menu 1,files,7,7
menu 2, modify 5,5
menu 3,data,9,9
// Activate the pull-down menu system
row = 1
col = 1
read menu bar to row, col save
```

Products

Recital Mirage Server, Recital Terminal Developer

RECALL

Class

Fields and Records

Purpose

Reinstate records that are marked for deletion

Syntax

```
RECALL [<scope>]  
[FOR <condition>]  
[WHILE <condition>]
```

See Also

DELETE, PACK, SET FILTER, SET DELETED, ZAP

Description

The RECALL command is used to reinstate those records that are marked for deletion in the active table. The default scope for the RECALL command is only to recall the current record. Records marked for deletion cannot be recalled once a table has been packed. If SET FILTER TO is in effect, then any records that do not satisfy the filter condition are not recalled. While SET DELETED ON is in effect, all deleted records are automatically filtered and cannot be recalled.

FOR <condition>

If the FOR clause is specified, then only those records which satisfy the <condition> are considered for recalling. If no <scope> was specified and the FOR clause was specified, then a scope of ALL is used.

WHILE <condition>

If the WHILE clause is specified, then the RECALL command terminates as soon as the specified <condition> is .F.. The WHILE clause is often used in conjunction with the FIND or SEEK commands to RECALL records which have a common key.

If the table is indexed, then the records are read in index order unless you specify RECORD <expN>, as the scope. If the currently selected table is not opened exclusively, the Recital/4GL will automatically lock each record in turn, recall it if required, then unlock the record.

Recital positions to EOF when RECALL specifies FOR or WHILE conditions and SET COMPATIBLE TO <XBASE> is in effect.

Example

```
use patrons index events  
delete for event = "OPERA" and eventdate < date()  
seek "OPERA"  
recall rest;  
    while event = "OPERA" and eventdate < date()
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

RECOVER

Class

Table Basics

Purpose

Recover table and index files from a previous backup and a journal file

Syntax

RECOVER FROM <.dbj filename> | (<expC>)

See Also

SET JOURNAL, COPY FILE

Description

The RECOVER command is used to recover a table and index files after a fatal error such as a disk head crash. The file name can be substituted with an <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then '.dbj' is assumed. A journal of all transactions applied to a table can be kept by issuing the SET JOURNAL command. Whenever changes are made to the table associated with the journal file, then the Recital/4GL will write an 'after image' of the modified table record and other associated information into the journal file. See the SET JOURNAL command for further details.

The journal file should be kept on a separate device to that on which the table and index files reside. At the end of each day, or if the table is constantly changing as in an application such as an order processing system, the table and index files should be 'checkpointed' to suitable backup media. If a fatal error occurs, such as a disk head crash, the following procedure should be carried out. Firstly, restore the table and index files from the last backup, then execute the RECOVER command.

The Recital/4GL will 'apply' all of the transactions in the journal file to the table, and update the indexes. After the RECOVER command has completed, you can continue with normal processing. Note that occasionally a few transactions may have been 'lost' as they may not have been written into the journal file when there was a fatal error. The journal file is a normal Recital table, so you can USE it and see the last transaction to be written, and if need be, apply the missing transactions manually. It is important to reinitialize the journal file after each backup of the table and index files. Use the ZAP command, or delete the file.

Example

```
use patrons index events, names
recover from patrons
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

REINDEX

Class

Indexing

Purpose

Rebuild all of the active index files

Syntax

REINDEX [ALL] [UNIQUE]

See Also

INDEX, SET INDEX, SET UNIQUE, USE, SET ICACHE

Description

The REINDEX command rebuilds all of the index files associated with the currently active table. The disk space previously occupied by the index files is not released, but merely reinitialized. Any FILTER is ignored in the reindexing process.

REINDEX cannot be executed if the table or tables in questions are not opened exclusively. Increasing the size of the index key 'cache' can optimize the performance of the REINDEX operation. The SET ICACHE command can be used to accomplish this.

ALL

If the ALL option is specified, all indexes associated with each open table will be reindexed.

UNIQUE

If the UNIQUE option is specified, or SET UNIQUE ON is in effect, then duplicate keys are discarded from the index files.

Example

```
set exclusive on
use patrons index events, names, dates
reindex
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

REINDEX DATABASE

Class

Databases

Purpose

Rebuilds the indexes for each table in the active database or rebuilds the catalog index tags for a specified database

Syntax

REINDEX DATABASE [<database name> | ?]

See Also

ALTER INDEX, ALTER TABLE, BACKUP DATABASE, CLOSE DATABASES, CLOSE TABLES, COMPILE DATABASE, CREATE DATABASE, CREATE TABLE, CREATE INDEX, CREATE VIEW, DISPLAY DATABASE, DISPLAY INDEXES, DISPLAY TABLES, DROP DATABASE, DROP INDEX, DROP TABLE, INDEX, LIST DATABASE, LIST INDEXES, LIST TABLES, OPEN DATABASE, PACK DATABASE, REBUILD DATABASE, RESTORE DATABASE, USE, SET AUTOCATALOG, SET EXCLUSIVE, ADATABASES(), DBUSED(), GETENV(), DB_MAXWKA

Description

The REINDEX DATABASE command rebuilds the indexes for each table in the active database.

Databases in Recital are implemented as directories containing files that correspond to the tables and associated files in the database. Operating System file protection can be applied individually to the files for added security. The directories are sub-directories of the Recital data directory. The environment variable / symbol DB_DATADIR points to the current Recital data directory and can be queried using the GETENV() function. Files from other directories can be added to the database using the ADD TABLE command or via the database catalog and SET AUTOCATALOG functionality. Databases are opened using the OPEN DATABASE command.

If the <database name> is specified, the REINDEX DATABASE command will operate on the specified database's catalog file: the catalog's index tags will be rebuilt. If the question mark, '?', is included instead of the <database name>, the 'SELECT A FILE' dialog will be displayed, allowing the user to select the database. The dialog defaults to the DB_DATADIR directory.

Example

```
VFP/SQL>open database southwind
VFP/SQL>reindex database
VFP/SQL>close databases
VFP/SQL>reindex database southwind
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

RELEASE

Class

Memory Variables

Purpose

Delete memory variables and free the storage that they were occupying

Syntax

RELEASE <memvar> | <memvar list> | ALL [LIKE <skeleton>][EXCEPT <skeleton>]

See Also

PRIVATE, PUBLIC, RETURN, CLEAR MEMORY, STORE, SAVE, RESTORE

Description

The RELEASE command deletes memory variables, and releases the storage that they were occupying. Recital will automatically release PRIVATE memory variables when a PROCEDURE or PROGRAM returns. You cannot release memory variables belonging to other procedures. If you RELEASE ALL from the '>' prompt, then all memory variables will be deleted. If you RELEASE ALL from a PROCEDURE, then only those memory variables that are PRIVATE to the PROCEDURE will be deleted. The <skeleton> takes the usual conventions of '?' matching any single character, and '*' matching zero or more characters.

Example

```
release i,j,k
release all like code_*
release all except c?de_*
release all
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

RELEASE LIBRARY

Class

Applications

Purpose

Close an active API library file built with the Recital SDK

Syntax

RELEASE LIBRARY <library filename>

See Also

CLOSE PROCEDURE, DO, FUNCTION, LINK, LIST PROCEDURE, PARAMETERS, PROCEDURE, SET LIBRARY

Description

The RELEASE LIBRARY <library filename> command closes the specified API procedure library file. The SET LIBRARY command is used to open API procedure library files and can also be used to close all active API procedure library files.

The SET LIBRARY and RELEASE LIBRARY commands only affect API procedure library files, not Recital/4GL procedure library files: these are handled by the SET PROCEDURE and CLOSE PROCEDURE commands.

The active API procedures and functions can be listed with the LIST or DISPLAY PROCEDURE commands.

For full details on using the Recital SDK, please see the SDK documentation.

Example

```
// Open Samples.so API procedure library
set library to /usr/recital/UnixDeveloper/sdk/lib/Samples.so
// Open pdf.so procedure library without closing active libraries
set library to /usr/recital/UnixDeveloper/sdk/lib/pdf.so additive
// Close pdf.so API procedure library
release library /usr/recital/UnixDeveloper/sdk/lib/pdf.so
// Close all active API procedure library files
set library to
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

RELEASE MENUS

Class

Menus

Purpose

Releases defined Xbase style menus

Syntax

RELEASE MENUS [<expC>]

See Also

CLEAR MENUS, DEACTIVATE MENU, SET COMPATIBLE

Description

The RELEASE MENUS command deletes menu definitions and frees the storage space that Xbase style menus were occupying. A menu currently in use cannot be released, and must be deactivated first. Using the RELEASE MENUS command without a menu name list <expC> will release all menus.

Example

release menus sort_men

Products

Recital Mirage Server, Recital Terminal Developer

RELEASE POPUPS

Class

Menus

Purpose

Releases Xbase style pop-up menus

Syntax

RELEASE POPUPS [<expC>]

See Also

DEACTIVATE POPUP, CLEAR POPUPS, SET COMPATIBLE

Description

The RELEASE POPUPS command deletes popup definitions and frees the storage space that Xbase style pop-up menus were occupying. Specified popups in the name list <expC> will be deactivated and all related ON SELECTION POPUP commands will be cleared. If you do not specify a pop-up name list <expC>, the RELEASE POPUP command erases all pop-up menus from the screen and from memory.

Example

release popups popup_4

Products

Recital Mirage Server, Recital Terminal Developer

RELEASE WINDOWS

Class

Screen Windows

Purpose

Delete window definitions and free the storage space that they were occupying

Syntax

RELEASE WINDOWS <window-name> | <window-name list> | ALL

See Also

ACTIVATE WINDOW, CLEAR WINDOWS, DEFINE WINDOW, DEACTIVATE WINDOW

Description

The RELEASE WINDOWS command removes windows from memory and from the screen. A window is an area of the screen designated for output and input. There is no limit to the number of defined windows. Windows are defined with the DEFINE WINDOW command, and activated with the ACTIVATE WINDOW command. You may release a single window, a group of windows, or all currently defined windows. The <window-name> is the name of the window as specified in the DEFINE WINDOW command. To release a group of windows, use a <window-name list>, which is a list of window names, each separated by a comma. To release all currently defined windows, use the ALL keyword.

The RELEASE WINDOWS command is a quick way to clear the screen and reclaim memory space for more windows. Once the RELEASE WINDOWS command is issued, the DEFINE WINDOW command must be used to establish further window definitions, and the ACTIVATE WINDOW or SHOW WINDOW commands must be used to display them. The RELEASE WINDOWS command is synonymous with the CLEAR WINDOWS command.

If you wish to clear a window from the screen, but retain its definition in memory, use the DEACTIVATE WINDOW command. If you wish to clear a window from the screen, but keep it active, use the HIDE WINDOW command. If you wish to clear windows from the screen, and save the window definition and the current window contents to a file, use the SAVE WINDOW and RESTORE WINDOW commands.

Example

release windows

Products

Recital Mirage Server, Recital Terminal Developer

RENAME

Class

Disk and File Utilities

Purpose

Change the name of a file

Syntax

RENAME <.dbf filename1> | (<expC>)[TO] <.dbf filename2> | (<expC>)

See Also

COPY FILE, ALIAS

Description

The RENAME command renames <filename1> to <filename2>. Either filename can be substituted with an <expC>, enclosed in round brackets, which returns a valid filename. If <filename2> already exists, the original file is overwritten. If no file extension is present in the file name, then the Recital/4GL uses '.dbf'. You cannot rename a file that is open. You cannot rename a file onto another disk, the COPY FILE command can be used for this purpose.

Example

```
rename patrons.dbf to patron.dbf  
rename events.ndx event.ndx
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

REPLACE

Class

Fields and Records

Purpose

Modify fields in a table file

Syntax

```
REPLACE [<scope>]
<field> WITH <exp> [,<field> WITH <exp>] | <memo-field> with <exp> [ADDITIVE]
[,<field> WITH <exp>...] | [,<memo-field> with <exp> [ADDITIVE]...]
[BLANK]
[FROM ARRAY <array-name>]
[FOR<condition>]
[WHILE <condition>]
[REINDEX]
```

See Also

:=, CHANGE, EDIT, READ, BROWSE, SET FILTER, SET DELETED, SET RELATION, SET DICTIONARY TO, DELETE, RECALL, PACK, USE, ZAP

Description

The REPLACE command updates fields in the active table. The default <scope> is the current record. The REPLACE command may also update fields in open tables other than the active table. Fields in other tables must be prefixed with an alias pointer, which is an alias name followed by the symbol ‘->’ or ‘.’. The alias name is the name optionally assigned in the USE command (if no alias is specified, the table basename is used) or the workarea letter.

If SET FILTER TO is in effect, then only those records that satisfy the filter are processed. If SET DELETED ON is in effect, then only those records that are not marked for deletion are processed.

BLANK

If the optional keyword BLANK is used the current record is cleared and replaced into the table. Default settings defined in the Data Dictionary apply to the replaced record.

<memo-field> WITH <expC> [ADDITIVE]

The <memo-field> with <exp> command is used to replace the specified memo field with an expression. If the ADDITIVE keyword is not used, existing memo field contents are overwritten by <exp>. The ADDITIVE keyword is used to append new text to the end of memo contents instead of overwriting the contents of the memo.

FROM ARRAY <array>

The REPLACE FROM ARRAY command allows you to replace fields in the current table with the contents of a previously declared two-dimensional <array> of the specified name. The data types and sizes of elements in the rows of the arrays must correspond to the fields in the table.

FOR <condition>

If the FOR <condition> clause is specified, only those fields in the rows which satisfy the specified <condition> will be replaced. When the FOR clause is used, the <scope> defaults to ALL.

WHILE <condition>

The WHILE <condition> clause can be used to restrict the number of records replaced. When the condition becomes false, the REPLACE operation will stop. If the WHILE condition is used in conjunction with the

FIND or SEEK commands on index files, the range of records being replaced can be restricted and performance can be optimized. When the WHILE clause is used, the <scope> defaults to REST.

REINDEX

The REINDEX keyword allows the indexes to be rebuilt upon completion of the command.

The field and the expression data types must be compatible. If the table is indexed, then the Recital/4GL processes the records in the table in the order as specified in the master index file. When you update fields in indexed files, and if the field being updated is part of the key, then the index file will also be updated automatically. Block replacements on indexed files have undefined results, as the next record keeps moving as the indexes are repositioned. You can overcome this if you issue the command SET ORDER TO 0 before executing REPLACE.

The REPLACE command can be used to update memos from long strings. If the currently selected table is shareable, then the Recital/4GL will automatically lock and then unlock each record in turn as it performs the REPLACE operation. If a replacement requires an index file to be updated, then the Recital/4GL will automatically lock the index file, update it and unlock it. If the record cannot be locked, an error message will be returned. The ON ERROR command can be used to trap this message.

The REPLACE command will evaluate any validation defined in the Applications Data Dictionary and return an error if the validation fails. The error can be trapped with the ON ERROR command. To bypass the Dictionary temporarily, use the SET DICTIONARY TO command.

If SET LOCKTYPE TO OPTIMISTIC is active, an attempt to use the REPLACE command on a record that has been modified since it was last read will generate an error.

Example

```
use patrons index events, names
replace all date with ctod("26/04/2000");
  for event = "PHANTOM"
```

```
// Multiple Replace Statements
```

```
replace firstname with m_firstname,;
      surname with m_surname,;
      state with m_state
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

REPLACE AUTOMEM

Class

Memory Variables

Purpose

Update fields with values stored in correspondingly named memory variables

Syntax

REPLACE AUTOMEM

See Also

GATHER, READ, SCATTER, STORE AUTOMEM, USE...AUTOMEM

Description

The REPLACE AUTOMEM command updates fields in the current record of the active table. It assumes the existence of memory variables that match the fields in terms of name, data type and size. Such memory variables can be generated automatically using the STORE AUTOMEM or USE...AUTOMEM commands.

Example

```
set locktype to optimistic
use customer
store automem
@1,1 get m.name
@2,1 get m.address
@3,1 get m.state
read
if not change()
    replace automem
endif
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

REPLAY

Class

Keyboard Events

Purpose

Replay a keyboard macro

Syntax

REPLAY FROM <.kbm filename> | (<expC>)

See Also

SET CAPTURE, &, MENU QUERY

Description

The REPLAY command is used to replay a keyboard macro <.kbm filename>. The file name can be substituted with an <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, '.kbm' is assumed. REPLAY works in conjunction with the SET CAPTURE TO command. SET CAPTURE TO initiates the capturing of all keys pressed at the keyboard to be stored in a keyboard macro file.

Everything typed, whether at the '>' prompt, or in a form or menu, is stored in the keyboard macro file. When keyboard capture is complete, the capture file can be closed by issuing the SET CAPTURE TO command without a <.kbm filename> specified. The REPLAY command then allows the keys to be played back. This is particularly useful for saving a session where you set up a report format, a table or similar file, or build a query using the MENU QUERY command.

As an alternative, keyboard macros can be initiated from within forms and menus by pressing the key sequence [^OC], then replayed by pressing the key sequence [^OR]. This allows you to store standard 'form filling' sequences and replay them when needed.

Example

```
set capture to mydbfkb
create mydbf
use
replay from mydbfkb
```

Products

Recital Terminal Developer

REPORT

Class

Reports

Purpose

Generate a report as defined in a report format file

Syntax

```
REPORT FORM <.frm filename> | (<expC>)  
[<scope>]  
[FOR <condition>]  
[WHILE <condition>]  
[PLAIN]  
[HEADING <expC>]  
[NOCONSOLE]  
[NOEJECT]  
[PREVIEW]  
[UNDERLINE]  
[SEPARATE]  
[SEPCHAR <expC>]  
[SUMMARY]  
[LPP <expN>]  
[TO TERMINAL]  
[TO PRINT]  
[TO FILE <.txt filename> | (<expC>)]
```

See Also

@, SET PRINT, SET ALTERNATE, CREATE REPORT, TREPORT, PRINT, SET FILTER, SET DELETED, ACC(), CALC()

Description

The REPORT command formats information from the active table, and any related tables, using the report format <.frm filename> set up with the CREATE REPORT Recital Terminal Developer development tool. The filename can be substituted with an <expC>, enclosed in round brackets, which returns a valid filename. The default <scope> for the report is to process ALL records, unless the WHILE condition is used, in which case the default is REST. You can report from more than one table by issuing the SET RELATION TO command, and using alias pointers in the expressions that you have specified. If SET FILTER TO is in effect, then only those records that satisfy the FILTER condition will be processed. If SET DELETED ON is in effect, then only those records that are not marked for deletion will be processed. If no file extension is specified in the FORM name, then '.frm' is assumed. If the report format contains group/subtotals, then the table must be either sorted, or indexed in group/subtotal order.

FOR <condition>

If the FOR clause is specified, then only those records which satisfy the specified <condition> will be processed.

WHILE <condition>

If the WHILE clause is specified, then processing will terminate when the specified <condition> is .F.. The WHILE clause can be used in conjunction with the FIND or SEEK commands, and the REST <scope> to restrict the records that are processed, and therefore optimize the performance of the command.

PLAIN

If the PLAIN option is specified, then the report is printed with the heading defined in the report format file, on the first page only.

HEADING

If the HEADING option is specified, then it will be printed on the top line of each page, in the center. HEADING accepts multiple lines of text that are separated by the ‘;’ character.

NOCONSOLE

If the NOCONSOLE option is specified, the report will not be displayed on the screen when being sent to a printer or file.

NOEJECT

If the NOEJECT option is specified, then there will be no initial form feed output.

LPP

The LPP clause is used to specify the number of lines per report page. The LPP clause will override the page length specification in the report format file (.frm).

PREVIEW

If the PREVIEW option is specified, the report will be previewed on the screen.

TO PRINT

If the TO PRINT option is specified, the report will be output to the printer.

TO FILE <file>

If the TO FILE option is specified, then the report will be output to an ASCII text <.txt filename>. If no file extension is specified in the file name, then ‘.txt’ is assumed. The file name can be substituted with an <expC>, enclosed in round brackets, which returns a valid filename.

TO TERMINAL

If the TO TERMINAL option is specified, then the report will be output in pages to the issuing terminal. The user will be prompted to press a key between pages.

UNDERLINE

If the UNDERLINE option is specified, then subtotal and total lines will be underlined.

SEPARATE

If SEPARATE is specified, then each report row will be separated from the next with ‘-’ characters.

SEPCHAR <expC>

The SEPCHAR option allows you to specify a character <expC> to separate each column. If the “|” character is used for this purpose, together with the SEPARATE option to separate rows, the report will be generated with the appearance of a table with horizontal and vertical rules.

SUMMARY

The SUMMARY option can be specified to overwrite the summary flag as specified with CREATE REPORT. If this option is used, only headings, sub-headings, sub-totals and totals will be produced from the report.

Word wrapping of columns is implemented if the column expression is longer than the column width. MEMO fields are automatically word wrapped in the column. User Defined Functions (UDF) can be used as part of the column expression to create more complex reports. These must be active when the report is created as the Recital/4GL evaluates the column expression for syntax errors. If you need to calculate horizontal totals across each line of the report, the ACC() and CALC() functions can be used to provide this functionality (see the CREATE REPORT command for details). If the active table is indexed, then the Recital/4GL processes the records in the order of the master index file.

Example

```
use patrons index events
report form patrons;
  for event = "BALLET" and date>date();
  heading "BALLET PATRONS";
  noject;
  separate
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

RESET IN

Class

Transaction Processing

Purpose

Disable journaling in a specified workarea

Syntax

RESET IN <workarea | alias>

See Also

ISMARKED(), ROLLBACK, ROLLBACK(), BEGIN...END TRANSACTION, COMPLETED(), SET ROLLBACK

Description

The RESET IN command is used to disable Before Image Journaling (BIJ) in the specified workarea. When BEGIN TRANSACTION is issued, all currently open tables and all tables opened between BEGIN and END TRANSACTION will have BIJ invoked automatically. If BIJ is not required on a particular table, then the RESET IN command should be issued for the relevant workarea.

Example

```
// Clear BIJ in workarea suppliers  
reset in suppliers
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

RESIZE WINDOW

Class

Screen Windows

Purpose

Change the size of a pre-defined window

Syntax

RESIZE WINDOW <window-name> BY <expN1>,<expN2>,<expN3>,<expN4>

See Also

ACTIVATE WINDOW, DEFINE WINDOW, MOVE WINDOW, WROWS(), WCOLS()

Description

The RESIZE WINDOW command changes the size of a window that has been previously defined with the DEFINE WINDOW command, and activated with the ACTIVATE WINDOW command. Resizable windows are specified by including the GROW keyword in the DEFINE WINDOW command. The <window-name> is the name of the window as specified in the DEFINE WINDOW command.

The RESIZE WINDOW command changes the size of a window according to the ordinates specified by numeric expressions <expN1> - <expN4>. The numeric expressions refer to the top and bottom rows, and the left and right columns, respectively. The ordinates are relative to the window's current position, not to the entire screen. Depending on the numeric expression a negative ordinate will resize the window to the left or up, and a positive ordinate will resize the window to the right or down.

Example

resize window browse by -1,0,0,0

Products

Recital Mirage Server, Recital Terminal Developer

RESTORE

Class

Memory Variables

Purpose

Restore memory variables and arrays previously saved with the SAVE command

Syntax

RESTORE FROM <.mem filename> | (<expC>)
[ADDITIVE]
[FOXPRO]

See Also

DISPLAY STATUS, MODIFY, PRIVATE, PUBLIC, RELEASE, SAVE, STORE, DB_FOXMEM

Description

The RESTORE command restores memory variables and arrays previously saved with the SAVE TO <.mem filename> command. The file name can be substituted with an <expC>, enclosed in round brackets, which returns a valid filename. All variables that exist prior to the RESTORE command being issued are released unless the ADDITIVE option is specified. The memory variables that are restored are all declared as PRIVATE at the current level, regardless of whether they were PRIVATE or PUBLIC when they were saved. If no file extension is specified on the FROM file, then '.mem' is used.

The Recital/4GL memory files are normal ASCII text files which can be modified with MODIFY COMMAND. They contain a series of STORE...TO commands, produced by the SAVE command.

If the FOXPRO keyword is specified, the memory files are treated as FoxPro style binary files. The FOXPRO keyword is not required: the RESTORE command will recognize the file format automatically.

Example

restore from monday additive

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

RESTORE COLOR

Class

Screen Forms

Purpose

Restore saved color schemes

Syntax

RESTORE COLOR FROM <memvar> | FILE <.col filename> | (<expC>)

See Also

@...GET COLOR, @...FILL, @...TO, SAVE COLOR, SET COLOR SCHEME TO, ISCOLOR(), SETCOLOR()

Description

The RESTORE COLOR FROM command allows the current color scheme to be changed to a previously saved color scheme.

FROM <memvar>

The FROM <memvar> clause can be used to restore the color scheme from a memory variable that was saved with the SAVE COLOR TO <memvar> command.

FROM FILE <.col filename>

The FROM FILE <.col filename> clause can be used to restore the color scheme from a color file. This file must have been created with the SAVE COLOR TO FILE <.col filename> command. The file name can be substituted with an <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, '.col' is assumed.

Example

```
save color to m_old_color
set color
// Restore color scheme from a memory variable
restore color from m.m_old_color
// Restore color scheme from a color file
restore color from file payroll.col
```

Products

Recital Mirage Server, Recital Terminal Developer

RESTORE GETS

Class

Screen Forms

Purpose

Restore saved @...GETS

Syntax

RESTORE GETS [FROM <memvar>]

See Also

@...GET, SAVE GETS, READ, READVAR()

Description

The RESTORE GETS command allows reactivation of @...GETS previously saved with the SAVE GETS command. The optional FROM <memvar> clause can be used to restore GETS from a memory variable. This allows for multiple SAVE and RESTORE GETS. Since the Recital/4GL allows @...GET commands to be specified within a validation routine, you can execute a screen form in a procedure invoked by the @...GET...VALIDATE command. When the validation procedure is terminated, the READ in the calling screen form is reactivated. This process is called nesting reads. The RESTORE GETS command also restores the value returned by the READVAR() function.

Example

```
// Set up validation procedure
procedure checkit
parameters check
save screen
// Specify SAVE GETS before @...gets
save gets
@1,1 get t1
@3,1 get t3
read
// Reactivate reads in main screen form
restore gets
restore screen
return

@1,1 get field1
@2,1 get field2;
    when field1="g"
@3,1 get field3 validate with checkit
read
```

Products

Recital Mirage Server, Recital Terminal Developer

RESTORE KEYS

Class

Keyboard Events

Purpose

Restore saved hot key assignments

Syntax

RESTORE KEYS FROM <memvar>

See Also

SET KEY TO, CLEAR KEYS, SAVE KEYS, ON KEY

Description

The RESTORE KEYS command reinstates hot key assignments that were previously saved with the SAVE KEYS command. The SAVE KEYS command allows you to save hot key assignments to a memory variable. Any keys except a-z or 0-9 may be assigned to procedures using the SET KEY or ON KEY commands. When a user presses this key, the assigned procedure executes. Keys used in this way are known as 'hot keys'. The <memvar> is the name of the memory variable to which the hot key assignments were saved, and from which they will be restored. To reset the hot key assignments back to default, use the CLEAR KEYS command.

Example

```
set key -1 to helpfunc
save keys to m_hotkeys
clear keys
....
restore keys from m.m_hotkeys
```

Products

Recital Mirage Server, Recital Terminal Developer

RESTORE MENU

Class

Menus

Purpose

Restore a previously saved menu

Syntax

RESTORE MENU [FROM <memvar>]

See Also

SAVE MENU TO, SET KEY TO, SET KEY...TO, SET PCKEYS, SET PREMENUM, SET POSTMENU, ON KEY, MENU(), MENUITEM()

Description

The RESTORE MENU command restores a menu that was previously saved with the SAVE MENU command.

FROM <memvar>

The optional FROM <memvar> clause is used when you have saved more than one menu. Without the FROM <memvar> clause the RESTORE MENU command restores the last menu saved. The <memvar> is a memory variable to which @...MENU context has been stored with the SAVE MENU TO command.

Example

```
procedure check_value
save menu to m_rcv
restore menu from m_rcv
return
```

Products

Recital Mirage Server, Recital Terminal Developer

RESTORE RECORDVIEW

Class

Fields and Records

Purpose

Restore a previously saved workarea status

Syntax

RESTORE RECORDVIEW FROM <memvar>

See Also

DO, SAVE RECORDVIEW TO, SET KEY TO, SET KEY...TO, SET PCKEYS, ON KEY, SKIP, REPLACE()

Description

The RESTORE RECORDVIEW command restores a workarea status that was previously saved to the specified <memvar> with the SAVE RECORDVIEW command. RESTORE RECORDVIEW restores the following information:

Workarea number
Current record number
Current index order
Lock status

The SAVE and RESTORE RECORDVIEW commands are useful in validation and hot key procedures when you want to move off the current record, execute a validation procedure, and then return to the same record. The SKIP 0 command must be used prior to a GOTO command in order to flush locked records to the disk if they have been modified. SET CLIPPER must be ON for SKIP 0 to work correctly.

Example

```
procedure check_value
save recordview to m_recv
// Validate data
restore recordview from m_recv
return
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

RESTORE SCREEN

Class

Screen Forms

Purpose

Restore a previously saved screen to the terminal display

Syntax

RESTORE SCREEN

[AT <expN1>,<expN2> TO <expN3>,<expN4>]

[FROM <memvar>] | [FROM FILE <.img filename> | (<expC>)]

See Also

SET SCREENMAP, SAVE SCREEN, SAVE SCREEN TO, @...SAY, @...MENU, MENU, ACHOICE()

Description

The RESTORE SCREEN command restores a screen that has been saved using the SAVE SCREEN command. Screen imaging must be enabled with SET SCREENMAP ON when using RESTORE SCREEN. All output to the screen except that from the RUN command is stored as part of the screen image. Up to 20 screen images can be saved in succession. Each time RESTORE SCREEN is executed, it restores from the last screen saved. When RESTORE SCREEN is executed, only those parts of the screen that have changed since the SAVE SCREEN was issued are refreshed.

AT <expN1>,<expN2> TO <expN3>,<expN4>]

The optional AT can be used to restrict the area of the screen to be restored by specifying the top row <expN>1 and column <expN2> and bottom row <expN3> and column <expN4> coordinates.

FROM <memvar> | FROM FILE <.img filename> | (<expC>)

The FROM option restores a screen which has been stored to a memory variable or file using the TO option. The filename can be substituted with an <expC>, enclosed in round brackets, which returns a valid filename.

Note: No SCREEN memory variables are saved to the <.mem> file if the command SAVE TO is used. If you require the SCREEN IMAGE to be saved for subsequent recall, use the SAVE SCREEN TO FILE command.

Example

```
save screen
dialog query
restore screen
```

Products

Recital Mirage Server, Recital Terminal Developer

RESTORE WINDOW

Class

Screen Windows

Purpose

Restore windows from a file

Syntax

RESTORE WINDOW <window-name> | <window-name list> | ALL
FROM <.win filename>

See Also

ACTIVATE SCREEN, ACTIVATE WINDOW, CLEAR WINDOWS, DEACTIVATE WINDOW, DEFINE WINDOW, HIDE WINDOW, MOVE WINDOW, MODIFY MEMO, RELEASE WINDOWS, RESIZE WINDOW, SAVE WINDOW, SHOW WINDOW, SET COMMANDWINDOW, SET ERRORWINDOW, SET STATUS, SET TRACEWINDOW, SET WINDOW OF EDIT, SET WINDOW OF MEMO, WROWS(), WCOLS(), WEXIST(), WVISIBLE(), WONTOP(), WOUTPUT()

Description

The RESTORE WINDOW command restores windows to memory from a file that was created with the SAVE WINDOW command. A window is an area of the screen designated for output and input. There is no limit to the number of defined windows. Windows are created with the DEFINE WINDOW command and activated with the ACTIVATE WINDOW command. The SAVE WINDOW command saves windows to a file with a '.win' extension.

You may restore a single window, a group of windows, or all the windows in the window file. The <window-name> is the name of the window as specified in the DEFINE WINDOW command. To restore a group of windows, use the <window-name list>, which is a list of window names each separated by a comma. To restore all currently defined windows, use the ALL keyword. Windows are restored with the same status they had when last stored with the SAVE WINDOW command. Windows being restored will overwrite any current window with the same name.

Example

```
clear windows
restore window sales from stats
activate window sales
```

Products

Recital Mirage Server, Recital Terminal Developer

RESUME

Class

Error Handling and Debugging

Purpose

Continue processing of a suspended program

Syntax

RESUME

See Also

SUSPEND, SET STEP, SET ECHO

Description

The RESUME command is used in conjunction with the SUSPEND command. RESUME continues processing of a program which has been suspended with the SUSPEND command. This command is primarily used when debugging programs. It can be used in conjunction with the SET STEP and SET ECHO commands.

Example

suspend

....

resume

Products

Recital Terminal Developer

RETRY

Class

Error Handling and Debugging

Purpose

Retry a command after an error was encountered

Syntax

RETRY

See Also

RETURN, ON ERROR, ERROR(), ERRNO(), MESSAGE ()

Description

The RETRY command re-executes a command that was reported as being in error. It is used in conjunction with the ON ERROR command.

Example

procedure file_open

```
if error() = 15 .and. errno() = 11
    dialog message message() + “. Retry?”
    if lastkey() = asc(“Y”)
        retry
    endif
endif
return
```

```
on error do file_open
use accounts exclusive
on error
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

RETURN

Class

Applications

Purpose

Return from a procedure, function, or program

Syntax

RETURN [TO MASTER] [<exp>]

See Also

PROCEDURE, SET PROCEDURE, RETRY, ON ESCAPE, ON KEY, ON ERROR, FUNCTION

Description

The RETURN statement closes the active program file, releases memory variables and arrays defined as private, and passes control back to the calling program. The RETURN statement is also used to denote the end of a procedure definition.

TO MASTER

If the TO MASTER is specified, then control is passed back to the highest level calling procedure.

<exp>

If the optional <exp> is specified, it will be returned to the calling program if the procedure or function was called as a User Defined Function.

Example

```
procedure example_1
do example_2
// Returns here<-----
return

procedure example_2
do example_3
return

procedure example_3
if .T.
    return to master -----
endif
return

do example_1
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

ROLLBACK

Class

Transaction Processing

Purpose

Restore tables to their condition at the beginning of a transaction

Syntax

ROLLBACK [<workarea | alias> | (<expC>)]

See Also

SET ROLLBACK, BEGIN...END TRANSACTION, ISMARKED(), RESET IN, COMPLETED()

Description

The ROLLBACK command is used in a BEGIN TRANSACTION ... END TRANSACTION block to roll back changes made to tables by any transaction performed between the two statements. ROLLBACK used by itself affects all open tables, but can be directed to a single table by specifying the <.dbf filename>. The filename can be substituted with an <expC>, enclosed in round brackets, which returns a valid filename. A 'transaction' is considered to be all the file modifications that occur within the commands BEGIN TRANSACTION and END TRANSACTION.

This command is particularly useful if an error occurs during a program modifying files. When BEGIN TRANSACTION is issued, all currently open files and all files opened between BEGIN and END TRANSACTION will have Before Image Journaling (BIJ) invoked automatically. The journals are stored in a log file <.log> that the Recital/4GL generates automatically. You can optionally specify the disk and directory path if you include them after the BEGIN TRANSACTION statement. If BIJ is not required on a particular file, then the command RESET IN <workarea> should be issued and journaling will no longer occur in that workarea.

The ROLLBACK() function returns .T. if a rollback succeeds. The COMPLETED() function can be used after the END TRANSACTION command to determine if any errors occurred during processing of the commands between BEGIN and END TRANSACTION. It returns .F. if errors occurred and .T. otherwise. If the command SET ROLLBACK is OFF, the COMPLETED() function can be used in conjunction with the ROLLBACK command to force a manual rollback. If SET ROLLBACK is ON and an error is encountered, any modifications to files within the transaction will be rolled back automatically to their state before the BEGIN TRANSACTION was executed.

Please note the following commands are not allowed during a transaction:

CLEAR ALL
CLOSE ALL
CLOSE DATABASE
CLOSE INDEX
MODIFY STRUCTURE
PACK
ZAP

If users do not have a specific reason for setting up manual error trapping and rollback routines using the ROLLBACK command, Recital Corporation recommends the use of the SET ROLLBACK ON command to perform automatic Before Image Journaling and rollback.

Example

procedure recovery

```

rollback
if rollback()
    dialog box "Rollback was ok."
else
    dialog box "Rollback not completed."
endif
return

use setcomm
on error do recovery
    begin transaction
        delete first 15
        replace all t1 with (t2*t3)/100
        list
    end transaction
if completed()
    dialog box "Transaction completed"
else
    dialog box "Errors occurred during transaction"
endif

```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

RUN

Class

Disk and File Utilities

Purpose

Execute an operating system command or external program.

Syntax

RUN | ! <os-command>

See Also

ALIAS, QUIT, KEYWORD, FUNCTION, SPAWN, LIST HISTORY, SET HISTORY

Description

The RUN command and the ! command are synonymous. They provide the facility for running operating system commands or external programs from within the system.

Example

run dir

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

SAVE COLOR

Class

Screen Forms

Purpose

Save current color scheme

Syntax

SAVE COLOR TO <memvar> | FILE <.col filename>

See Also

@...GET COLOR, @...FILL, @...TO, RESTORE COLOR, SET COLOR SCHEME TO, ISCOLOR(), SETCOLOR()

Description

The SAVE COLOR TO command saves the current color scheme to a memory variable or a file. When Recital Terminal Developer is activated from the operating system prompt, the colors are set from a system wide color file, called default.col, stored in the software root directory. Also, a local default.col file can be created in your current directory to be called after the system file. These files can be changed with the SAVE COLOR TO FILE and SET COLOR SCHEME TO commands.

TO <memvar>

The TO <memvar> clause can be used to save the color scheme to a memory variable. This variable can be restored with the RESTORE COLOR FROM <memvar> command.

TO FILE <.col filename>

The TO FILE <.col filename> clause can be used to save the color scheme to a color file. This file can be restored with the RESTORE COLOR TO FILE <.col filename> command. The filename can be substituted with an <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, '.col' is used.

Example

```
save color to m_old_color
set color
restore color from m->m_old_color
save color to file payroll.col
```

Products

Recital Mirage Server, Recital Terminal Developer

SAVE ERROR

Class

Error Handling and Debugging

Purpose

Save error.mem information to a specified file when an error occurs

Syntax

SAVE ERROR TO <mem filename>

See Also

ON ERROR, SET ERRORVERSION, ERRNO(), ERROR(), MESSAGE(), PROCLINE(), PROCNAME()

Description

The SAVE ERROR command saves the error.mem information to a specified file <mem filename> when an error occurs. This includes the following information:

- Recital Software version, patch release and compilation date
- Date and time file created
- Machine and user names
- Stack trace
- Active public and private procedures and functions
- Public and private memory variables
- Active status of workareas
- Settings as per DISPLAY STATUS

The SAVE ERROR command should be used in conjunction with the ON ERROR command. NOTE: multiple numbered error.mem files can be created automatically if SET ERRORVERSION is ON.

Example

```
procedure errproc
on error
lerrflag = .T.
save error to errlog
return
```

```
// Attempt to open non-existent table
lerrflag = .F.
on error do errproc
use nontable
if not lerrflag
    // Continue processing
else
    dialog box "Error has occurred"
endif
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

SAVE GETS

Class

Screen Forms

Purpose

Save status of current GETS

Syntax

SAVE GETS [TO <memvar>]

See Also

@...GET, @ GET VALIDATE, RESTORE GETS, SAVE SCREEN, RESTORE SCREEN, FMT()

Description

The SAVE GETS command lets you save a description of the GETS in the current screen form for subsequent retrieval by the RESTORE GETS command. Since the Recital/4GL allows @...GET commands to be specified within a validation routine, the SAVE GETS command is particularly useful in letting you execute a screen form in a procedure invoked by the @...GET...VALIDATE command. When the validation procedure is terminated, the READ in the calling screen form is reactivated. This process is called nesting reads.

TO <memvar>

The optional TO <memvar> clause can be used to save the active GETS to a memory variable. This allows for multiple SAVE and RESTORE GETS.

Example

```
// Set up validation procedure
procedure checkit
parameters check
save screen
// Specify SAVE GETS before @...gets
save gets
@1,1 get t1
// Activate gets in validation procedure
read
// Reactivate reads in main screen form
restore gets
restore screen
return

@1,1 get field1
@2,1 get field2;
    when field1="g"
@3,1 get field3 validate with checkit
read
```

Products

Recital Mirage Server, Recital Terminal Developer

SAVE KEYS

Class

Keyboard Events

Purpose

Save hot key assignments to a memory variable

Syntax

SAVE KEYS TO <memvar>

See Also

SET KEY TO, CLEAR KEYS, RESTORE KEYS, ON KEY

Description

The SAVE KEYS command allows you to save hot key assignments to a memory variable. Any keys, except a-z and 0-9 may be assigned to procedures using the SET KEY or ON KEY commands. When a user presses this key, the assigned procedure executes. Keys used in this way are known as 'hot keys'. The <memvar> is the name of the memory variable to which currently active hot key assignments will be saved. The CLEAR KEYS command resets hot key assignments back to their default. The RESTORE KEYS command may be used to recall stored hot key assignments from the specified memory variable.

Example

```
set key -1 to helpfunc
save keys to m_hotkeys
clear keys
....
restore keys from m.m_hotkeys
```

Products

Recital Mirage Server, Recital Terminal Developer

SAVE MENU

Class

Menus

Purpose

Save current @...menu context

Syntax

SAVE MENU [TO <memvar>]

See Also

RESTORE MENU TO, SET KEY TO, SET KEY...TO, SET PCKEYS, SET PREMENU, SET POSTMENU, ON KEY, MENU(), MENUITEM()

Description

The SAVE MENU command saves a menu that can be restored again with the RESTORE MENU command. The SAVE MENU command saves the current @...MENU context, which is useful in validation and hot key procedures when you want to move off and then back on the current menu.

TO <memvar>

The optional TO <memvar> clause is used to store the current @...MENU context to a memory variable. Menus saved to a memory variable can be restored with the RESTORE FROM <memvar> command. Without the FROM <memvar> clause the RESTORE MENU command restores the last menu saved.

Example

```
procedure check_value
save menu to m_recv
restore menu from m_recv
return
```

Products

Recital Mirage Server, Recital Terminal Developer

SAVE RECORDVIEW

Class

Fields and Records

Purpose

Save the status of the currently active workarea to a memory variable

Syntax

SAVE RECORDVIEW TO <memvar>

See Also

RESTORE RECORDVIEW FROM, SET KEY...TO, SET PCKEYS, ON KEY, SKIP, REPLACE()

Description

The SAVE RECORDVIEW TO command is used to save the status of the active workarea to the specified memory variable <memvar>. The SAVE RECORDVIEW command saves the following information pertaining to the active workarea:

Workarea number
Current record number
Current index order
Lock status

The RESTORE RECORDVIEW command may be used to restore the above information from the specified memory variable. The SAVE and RESTORE RECORDVIEW commands are particularly useful in validation and hot key procedures when you want to move off and then back on the current record. The SKIP 0 command must be used prior to a GOTO command in order to flush locked records to disk if they have been modified. SET CLIPPER must be ON for SKIP 0 to work correctly.

Example

```
procedure check_value
save record view to m_recv
// Validate data
restore recordview from m_recv
return
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

SAVE SCREEN

Class

Screen Forms

Purpose

Save the currently displayed screen in a memory variable or file

Syntax

SAVE SCREEN

[AT <expN1>,<expN2> TO <expN3>,<expN4>]

[TO <memvar>] | [TO FILE <.img filename> | (<expC>)]

See Also

SET SCREENMAP, RESTORE SCREEN, CHANGE, EDIT, QUERY, APPEND, BROWSE, @...SAY, @...MENU, MENU

Description

When SCREENMAP is ON, the SAVE SCREEN command allows an image of the current terminal display to be saved. All screen output except that from the RUN command is stored as part of the screen image. The screen image can be restored with the RESTORE SCREEN command. Up to 20 screen images can be saved at any one time. Each time RESTORE SCREEN is executed, it restores from the last screen saved. When RESTORE SCREEN is executed, only those parts of the screen that have changed since the SAVE SCREEN was issued are refreshed. These commands are particularly useful when used in conjunction with pop-up menus from within forms.

AT <expN1>,<expN2> TO <expN3>,<expN4>

The optional AT clause will save a screen image of the screen from top row <expN1> and column <expN2> to bottom row <expN3> and column <expN4>.

TO <memvar>

The optional TO clause can be used to save the screen image to a <memvar>.

TO FILE <.img filename> | (<expC>)

The optional TO FILE clause can be used to save the screen image to an <.img filename>. The file name can be substituted with an <expC>, enclosed in round brackets, which returns a valid filename.

Example

save screen

...

restore screen

Products

Recital Mirage Server, Recital Terminal Developer

SAVE TO

Class

Memory Variables

Purpose

Save the current memory variables to a file

Syntax

SAVE TO <.mem filename> | (<exp>)

[ALL LIKE <skeleton>]

[ALL EXCEPT <skeleton>]

[FOXPRO]

See Also

PRIVATE, PUBLIC, RESTORE, STORE, DB_FOXMEM

Description

The SAVE TO command saves all of the memory variables and arrays to an ASCII text file <.mem filename>. The filename can be substituted with an <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, then the Recital/4GL uses '.mem'. By default, all memory variables are saved unless the ALL LIKE or ALL EXCEPT clauses are specified. The <skeleton> takes the normal rules of '?' matching any single character and '*' matching zero or more characters.

The saved file is a normal ASCII text file, which can be edited using MODIFY COMMAND. The contents of this file consist of a series of STORE commands that give details of the value of the memory variable when it was saved. The memory variables can be restored from the file using the RESTORE FROM command. Whenever an error is detected in a program file, the Recital/4GL automatically saves the state of all the current memory variables in the file 'error.mem'. This file also contains the same information that is displayed with the DISPLAY STATUS command.

If the FOXPRO keyword is specified, the memory files are created as FoxPro style binary files. This is also the case if the DB_FOXMEM environment variable / symbol is set (on, true, yes).

Example

```
save to monday all like mon_*  
save to others all except mon_*
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

SAVE WINDOW

Class

Screen Windows

Purpose

Save window definition to a file

Syntax

SAVE WINDOW <window-name> | <window-name list> | ALL
TO <.win filename>

See Also

ACTIVATE SCREEN, ACTIVATE WINDOW, CLEAR WINDOWS, DEACTIVATE WINDOW, DEFINE WINDOW, HIDE WINDOW, MOVE WINDOW, MODIFY MEMO, RELEASE WINDOWS, RESIZE WINDOW, RESTORE WINDOW, SHOW WINDOW, SET COMMANDWINDOW, SET ERRORWINDOW, SET STATUS, SET TRACEWINDOW, SET WINDOW OF EDIT, SET WINDOW OF MEMO, WROWS(), WCOLS(), WEXIST(), WVISIBLE(), WONTOP(), WOUTPUT()

Description

The SAVE WINDOW command saves window definitions to a file that can be restored later with the RESTORE WINDOW command. A window is an area of the screen designated for output and input. There is no limit to the number of defined windows. Windows are created with the DEFINE WINDOW command and activated with the ACTIVATE WINDOW command. You may save a single window, a group of windows, or all the currently defined windows in the window file. The <window-name> is the name of the window as specified in the DEFINE WINDOW command. To save a group of windows, use the <window-name list>, which is a list of window names, each separated by a comma. To save all currently defined windows, use the ALL keyword.

By default, the SAVE WINDOW command saves windows to a file with a “.win” extension, however you may specify any extension desired. When windows are restored, they have the same status they had when last stored with the SAVE WINDOW command. Windows being restored will overwrite any current window with the same name.

Example

```
clear
define window sales from 2,1 to 13,75;
    title "Today's Sales"
use sales
total on amt_rcvc to temp;
    for date = date()
use temp
list fields item, amt_recvd;
    save window sales to stats
```

Products

Recital Mirage Server, Recital Terminal Developer

SCAN

Class

Applications

Purpose

Perform list of commands interactively

Syntax

```
SCAN [<scope>] [FOR <condition>] [WHILE <condition>]  
[EXIT]  
[LOOP]  
ENDSCAN
```

See Also

LOCATE, CONTINUE, DO WHILE, IF

Description

The SCAN ... ENDSCAN command executes a list of commands repeatedly for a specified selection of records while an optionally specified condition is true, or until an EXIT is encountered. The optional LOOP keyword forces control to the beginning of the SCAN.

The SCAN...ENDSCAN command can be used to reduce the programming involved in a LOCATE ... DO WHILE ... CONTINUE construct.

Example

```
scan all for code = "HMT"  
    display name, address  
endscan
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

SCATTER

Class

Array Processing

Purpose

Copy the contents of fields to an array or to a series of memory variables

Syntax

SCATTER [FIELDS <field list>] [MEMO] TO <array> [BLANK] | MEMVAR [BLANK]

See Also

COPY FROM ARRAY, COPY TO ARRAY, GATHER, PRIVATE, PUBLIC, AFILL(), ADIR(), ASORT(), ALLEN(), AFIELDS(), AINS(), ADEL(), ACHOICE(), ASCAN()

Description

The SCATTER command copies the contents of fields from the current table record into an array or series of memory variables.

FIELDS <fields>

The optional FIELDS clause is used to copy only the contents of fields specified in the <field list>. If the FIELDS clause is not specified, the SCATTER command copies the contents of all fields.

MEMO

By default, memo fields are ignored by the SCATTER command. If the MEMO keyword is specified, memo fields will be included.

To <array> [BLANK]

The fields are copied into consecutive elements of the specified array. If the array does not exist, then it is created. If the BLANK keyword is specified, the elements are created but are empty and are the same size and data types as the specified fields.

MEMVAR [BLANK]

The fields are copied into a series of memory variables with the same name as the field names. If the memory variables do not exist, then they are created. If the BLANK keyword is specified, the memory variables are created but are empty and are the same size and data types as the specified fields.

NOTE: 'TO' should not be included in the MEMVAR clause.

Example

```
use addresses index add_1
seek "Seymour House"
if found()
    scatter to aTemp
endif
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

SCROLL

Class

Screen Forms

Purpose

Designate a scrollable screen area

Syntax

SCROLL <expN1>,<expN2>,<expN3>,<expN4>,<expN5>

See Also

SCROLL (), HSCROLL, MAXCOL(), MAXROW(), MENU BROWSE

Description

The SCROLL command designates a rectangular portion of the screen as scrollable. <expN1> and <expN2> represent the upper left row and column coordinates of the rectangle, respectively. <expN3> and <expN4> represent the lower right row and column coordinates of the rectangle, respectively. The <expN5> specifies the number of rows to scroll inside the defined rectangular area. A negative value will scroll down, and a positive value will scroll up.

Example

```
for i = 1 to 20
  @ i,10 say replicate(str(i,2),30)
next
for i = 3 to 18
  scroll 3,19,18,59,1
  sleep 1
next
```

Products

Recital Terminal Developer

SEEK

Class

Indexing

Purpose

Search for specified key in the master index and if found, position the record pointer in the table

Syntax

SEEK <key expression>

See Also

FIND, INDEX ON, SET EXACT, SET TALK, DBXDESCEND(), DESCEND(), DESCENDING(), DTOS(), EOF(), FOUND(), LTOS(), STR()

Description

The SEEK command is identical to the FIND command, except that you can specify any valid expression as the key without the need to use macros. The SEEK command looks up the specified key in the master index file. If the key is found, then the FOUND() function will return .T., and the EOF() function will return .F.. If the key is not found, then the FOUND() function will return .F., and the EOF() function will return .T..

In the Recital/4GL, you can build indexes on any data type or any combination of data types. The conversion functions STR(), DTOS() and LTOS() are used to build indexes on mixed data types. The DESCEND() function can also be used to build indexes in descending key order. If the DESCEND() function is used to create the index key, it must also be used in the search <key expression>. Tag indexes built with the DESCENDING keyword require the use of the DBXDESCEND() function in the <key expression>. The DESCENDING() function can be used to determine whether a particular tag was built with the DESCENDING keyword.

If SET TALK ON is in effect and the specified key is not found, then the system displays a message on the screen. If SET EXACT is OFF, then the Recital/4GL will match partial keys. If SET EXACT is ON, then the Recital/4GL will match only complete keys.

Example

```
use patrons
index on event to events
seek "BALLET"
```

```
index on date to dates
seek ctod("01/01/2000")
```

```
index on dtos(eventdate) + "/" + event to dates
seek "20010101/BALLET"
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

SELECT

Class

Table Basics

Purpose

Select a workarea

Syntax

SELECT <workarea | alias>

See Also

USE, ALIAS(), SELECT (), WORKAREA(), SET FILTER, SET RELATION, SET VIEW

Description

The SELECT command is used to select a workarea. By default Recital environments have 20 workareas. At any given time, a particular workarea is active and selected. The number of workareas may be configured up to the maximum supported by setting the environment symbol DB_MAXWKA.

The workareas are numbered 1 to 20 (or DB_MAXWKA) and can be selected by number. When you USE a table in a selected workarea, an ALIAS name may be optionally specified. If none is specified, the table basename can also be used as the alias as can the workarea letter A to T (up to Z if DB_MAXWKA is set higher). The alias 'm' is reserved for memory variables and cannot be used to reference a workarea. This ALIAS name gives the workarea an identification, so that you can reference fields in workareas other than the currently selected one, by preceding the field name with the ALIAS name followed by '->' or '.' followed by the field name. This construction is known as an alias pointer.

Each workarea contains the context for the table that has been opened in that workarea. The current record pointer, the current record, the format file, the index files, the filter condition, and the relationships to other workareas. The SELECT 0 command selects the next available workarea and provides an alternative to using the WORKAREA() function. The LIST STATUS command provides full details of the current status of each workarea.

Example

```
select a
use patrons index events, dates alias pat
select b
use addresses;
    index addr_names alias add
select pat
? add.state
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

SHOW GET

Class

Screen Forms

Purpose

To redisplay the specified 'get' object or call a 'get' method

Syntax

```
SHOW GET <expN1> | <expC1> [, <expN2> [PROMPT <expC2>]]  
[COLOR SCHEME <expN3>] | [COLOR <expC3>]  
[ENABLE | DISABLE]  
[LEVEL <expN4>]  
[PROPERTIES <expC4>]
```

See Also

@...GET, READ, SHOW GETS, SHOW OBJECT

Description

The SHOW GET command is used to update the screen display for a specific get or to call a get method. If the value of a get is changed without user intervention, for example via a validation function, the SHOW GET command can be used to redisplay the altered value on the screen. The <expN1> is the number of the get based on its screen position. The numbers start at 1 for the top left-hand get. The get can also be specified using a character expression, <expC1>, that returns the variable or field name of the get.

PROMPT <expC2>

Where the specified get is a pushbutton, radio button or check box, the PROMPT clause can be used to change the current prompt of the get. The new prompt is specified in <expC2>.

COLOR SCHEME <expN3> | COLOR <expC3>

The COLOR SCHEME | COLOR clause is used to update the foreground and background colors of the specified get. The <expN3> is the number of a color scheme. The <expC3> is a color pair in the format foreground / background.

ENABLE | DISABLE

The ENABLE keyword enables the get, allowing it to be modified or selected, whereas the DISABLE keyword disables the get and it cannot be modified or selected.

LEVEL <expN4>

The LEVEL clause can be used to specify the level number of the READ, <expN4>. SHOW GET defaults to the current READ level.

PROPERTIES <expC4> (Recital Mirage only)

The PROPERTIES clause can be used to either specify properties for the specified get or to call one of its methods. For information on the properties and methods appropriate to each object, please see The Mirage Object Model in the Recital Mirage documentation.

Example

```
function v_button  
  do case  
    case mbutton = 1  
      show get 1 properties "method=moveNextPage"  
    case mbutton = 2  
      show get 1 properties "method=movePreviousPage"
```

```

case mbutton = 3
    dialog message "Do you want to quit the Mirage demo?"
    if lastkey() == 89
        show get 2 properties "method=quit"
        quit
    endif
case mbutton = 4
    printFile("c:\Program files\Recital\UAS\Mirage\Mirage_demo\mirage_demo.prg" + ;
        "?tabs=4&topMargin=50&autoNumber=true", ;
        "{FILE}},{DATE}","-- Page {PAGE} --,")
endcase
return .T.

```

Products

Recital Mirage Server, Recital Terminal Developer

SHOW GETS

Class

Screen Forms

Purpose

Refreshes current @...GETs on the screen.

Syntax

SHOW GETS

See Also

@...GET, SET POSTRECORD, SET PRERECORD, SET PREMENU, SET POSTMENU

Description

The SHOW GETS command refreshes current @...GET values on the screen. This command is particularly useful when called from a VALID function or from trigger procedures which change the values of currently displayed @...GETs. NOTE: This command is not designed for use in table fields, created with the DEFINE TABLE command.

Example

```
function get_state
do case
  case substr(m_zip,1,2) = "01"
    m_state = "MA"
  case substr(m_zip,1,2) = "91"
    m_state = "CA"
  otherwise
    m_state = "XX"
endcase
show gets
return .T.

m_zip = space(5)
m_state = space(2)
@10,05 say "Enter Zip" get m_zip;
  valid get_state()
@12,05 say "Enter State" get m_state
read
```

Products

Recital Mirage Server, Recital Terminal Developer

SHOW MENU

Class
Menus

Purpose
Displays an Xbase style menu on the screen without activating it

Syntax
SHOW MENU <expC1> [PAD <expC2>]

See Also
ACTIVATE MENU, CLEAR, CLEAR MENU, DEFINE MENU, DEFINE PAD, SET COMPATIBLE

Description
The SHOW MENU command displays the specified Xbase style menu named <expC1> over any existing display. The SHOW MENU command does not activate the menu. The command SET COMPATIBLE should be set ON when using Xbase style menus.

PAD <expC>
The PAD option highlights the specified pad name <expC2> when the menu displays.

Example
show menu sort_men

Products
Recital Mirage Server, Recital Terminal Developer

SHOW OBJECT

Class

Screen Forms

Purpose

To redisplay a Recital Mirage object or call a Recital Mirage object method

Syntax

SHOW OBJECT <expC1> [PROPERTIES <expC2>]

See Also

@...GET, READ, SHOW GET, SHOW GETS

Description

The SHOW OBJECT command is only available in Recital Mirage: it is ignored in Recital Terminal Developer. It allows Recital Mirage objects to be redisplayed on the screen or their methods to be called. Objects are assigned in Recital Mirage using the id = <object_name> syntax in the PROPERTIES clause of @...SAY and @...GET commands. For more information on the PROPERTIES clause and the methods appropriate to each object, please see The Mirage Object Model in the Recital Mirage documentation.

Example

```
function ButtonEventHandler
    show object "customer_name" properties "method=hide"
return .t.

@2,0 say "Customer Name:" properties "id=customer_name"
store 1 to mchoice
@3,3 get mchoice picture "@*H Update" valid ButtonEventHandler()
read
```

Products

Recital Mirage Server

SHOW POPUP

Class

Menus

Purpose

Displays specified Xbase style pop-up menu to screen

Syntax

SHOW POPUP <expC> [SAVE]

See Also

ACTIVATE POPUP, CLEAR POPUPS, DEFINE BAR, DEFINE POPUP, POPUP(), SET COMPATIBLE

Description

The SHOW POPUP command displays an Xbase style pop-up menu named <expC> to the screen. This command does not activate the specified pop-up menu. The command SET COMPATIBLE should be set ON when using Xbase style menus.

SAVE

The optional SAVE keyword can be used to automatically save the screen prior to the display of the popup and restore the screen when the popup is no longer hidden or exited.

Example

```
show popup popup_4
```

Products

Recital Mirage Server, Recital Terminal Developer

SHOW WINDOW

Class

Screen Windows

Purpose

Display a pre-defined window

Syntax

SHOW WINDOW <window-name> | <window-name list> | <ALL>

See Also

ACTIVATE WINDOW, DEFINE WINDOW, RESTORE WINDOW, SAVE WINDOW

Description

The SHOW WINDOW command displays a pre-defined window, but does not activate it. A window is an area of the screen designated for output and input. There is no limit to the number of defined windows. Windows are defined with the DEFINE WINDOW command, and are activated with the ACTIVATE WINDOW command. If the window was previously activated with the ACTIVATE WINDOW command, the SHOW WINDOW command displays the window in an active state.

The SHOW WINDOW command can display a single window, a group of windows, or all previously defined windows. The <window-name> is the name of the window as specified with the DEFINE WINDOW command. A <window-name list> is a list of window names, each separated by a comma. To display all currently defined windows, use the ALL keyword.

Used in conjunction with the HIDE WINDOW command, the SHOW WINDOW command can be used to display previously hidden windows. The HIDE WINDOW command removes a window or group of windows from the screen. Hidden windows remain active in memory, and output may be directed to hidden windows. Hidden windows may be revealed with either the SHOW WINDOW or ACTIVATE WINDOW commands.

The HIDE WINDOW and SHOW WINDOW commands may be used in a hot key procedure to switch the screen display from windows to full screen. Full screen display is enabled with the ACTIVATE SCREEN command. Hot keys enable users to press a key that causes execution of a specified procedure while running an application that is waiting for keyboard input.

Example

show window all

Products

Recital Mirage Server, Recital Terminal Developer

SKIP

Class

Fields and Records

Purpose

Move the record pointer forwards and backwards in the active table

Syntax

SKIP [<expN>] [IN | ALIAS <workarea | alias>]

See Also

LOCATE, CONTINUE, FIND, REPLACE, SEEK, SET CLIPPER, GOTO, BOF(), EOF(), RECNO(), FOUND()

Description

The SKIP command moves the record pointer forwards or backwards in the currently selected table. The numeric expression <expN> can be positive or negative. If the SKIP command is issued with no numeric expression <expN> specified then the record pointer advances on to the next record. If the currently selected table is indexed, then the SKIP command follows the order of the master index. The SET ORDER TO command can be used to select which of the open index files should be the master or controlling index.

If the record pointer is currently positioned on the last record of the table and the SKIP command is issued, the EOF() function will return .T.. If the record pointer is currently positioned on the first record of the table, and the SKIP -1 command is issued, the BOF() function will return .T..

If SET CLIPPER is ON, the SKIP 0 command will flush locked records to disk if they have been modified. If no records are locked, SKIP 0 is ignored. SKIP 0 will not work this way in interactive mode. SKIP will ignore empty tables, and no error will occur. SKIP 0 is most effective from within forms using REPLACE commands in a validation procedure. The SKIP command is primarily used in conjunction with the DO...WHILE command.

IN | ALIAS <workarea | alias>

The optional IN <workarea | alias> or ALIAS <workarea | alias> clause allows you to move the record pointer in a workarea other than the current workarea. The record pointer in the current workarea is not moved as a result of this option.

Example

use patrons index events, dates, names

m_event = event

do while .not. eof()

 display off date, event, name

 if m_event # event

 ?

 m_event = event

 endif

 skip

enddo

// Another example

skip 9 in orders

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

SLEEP

Class

Applications

Purpose

Pause program execution for a specified number of seconds

Syntax

SLEEP <expN>

See Also

SET MESSAGE, DIALOG BOX, INKEY(), ON KEY

Description

The SLEEP command pauses execution of a program for the specified number of seconds.

Example

```
use patrons index events, dates, names
seek "CONCERTO"
if .not. found()
  set message to "Event not found."
  sleep 2
  set message to
endif
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

SORT

Class

Fields and Records

Purpose

Sort a table to another in a specific order

Syntax

```
SORT [<scope>] TO <.dbf filename> | (<expC>)  
ON <field> [/A] [/D] [/C]  
[FOR <condition>]  
[WHILE <condition>]
```

See Also

INDEX, DESCEND(), SET INDEX

Description

The SORT command copies records from the currently selected table to another table in the specified sorted order. The file name can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, '.dbf' is used. If no <scope> is specified, a scope of ALL is used, unless the WHILE clause is specified, in which case the <scope> will default to REST. When you want to sort a table on multiple fields, then you should specify the most important field first.

/A /D /C

The /A, /D, and /C options can be specified for each sort field. The /A option sorts on the specified field in ascending order. The /D option sorts in descending order. The /C option can be used in conjunction with /A or /D to provide case insensitive sorting for character fields. For example, /AC causes the Recital/4GL to treat all characters in the sort field as upper case. If no sort order is specified on a field, then /A is assumed.

FOR <condition>

If the FOR <condition> is specified, then only those records which satisfy the specified <condition> are processed.

WHILE <condition>

The WHILE <condition> is often used in conjunction with the SEEK command, and the REST <scope>, to restrict the range of records processed, and therefore reduce the time needed to sort the table.

If the currently selected table is indexed, then SORT processes the records in order of the master index. You cannot sort on logical fields. You cannot sort a table on expressions using the SORT command, this can be achieved using the INDEX command. The INDEX command is a much faster way of organizing a table. The DESCEND() function can be used with the INDEX command to create indexes in descending order.

Example

```
use patrons  
sort to temp on date/a, event/ac, name/ac;  
for year(date)=1999
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

SPAWN

Class

Disk and File Utilities

Purpose

Execute an external program and optionally wait for its completion

Syntax

SPAWN | !! <os-command>

See Also

SPAWNPID(), ACTIVEPID(), CANCELPID()

Description

The SPAWN command is synonymous with the !! command. These commands provide the facility for executing external programs, in the 'background', from within the Recital/4GL. The SPAWN command differs from the RUN command in that the specified <os command> executes in parallel without waiting until command execution is complete. It is particularly useful for generating reports from shared tables. When running a command in the background, terminal output is disabled for the background command if it is running the Recital/4GL.

The SPAWNPID() function returns the identity of the spawned process. The ACTIVEPID(<pid>) function returns .T. if the specified process is still active and .F. otherwise. The CANCELPID(<pid>) function returns .T. if the specified process could be 'killed' and .F. otherwise. A spawned process will only remain active while the user remains logged in. All spawned processes will be terminated when the user logs out.

Example

```
spawn db printrep
pid = spawnpid()
on escape killed = cancelpid(pid)
if .not. activepid(pid)
    set message to "Printing completed."
endif
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

STORE

Class

Memory Variables

Purpose

Save the result of an expression in a memory variable

Syntax

STORE <exp> TO <memvar> [,<memvar>,...]

See Also

:=, AVERAGE, COUNT, PARAMETERS, PRIVATE, PUBLIC, SUM

Description

The STORE command saves the result of the expression <exp> into a memory variable. This is equivalent to the assignment statement:

<memvar>=<exp>

If the memory variable does not exist, then it is created. If the memory variable already exists, its contents are updated.

Recital automatically performs type conversions for memory variables. If, for example, an existing memory variable called NAME contains a character string, and the command STORE 10 TO NAME is issued, the memory variable will automatically be converted to a numeric memory variable.

Memory variables are normally created with the STORE command, the assignment command '=', the PUBLIC command, or the PRIVATE command. When a program is being executed, Recital gives field variables the 'highest precedence' in expressions. In other words, if a field name in a table is the same as the name of a memory variable, then Recital will take the value of the field variable. To overcome this you can use the special alias name 'm->' or 'm.' to reference the memory variable. You only need to use this notation where an expression can be specified.

Memory variables created with the STORE command are declared PUBLIC if they are created at the '>' prompt, and PRIVATE otherwise. When assigning expressions to memory variables, Recital allows character variables to be added together, full date arithmetic is also supported.

Numeric memory variables may be incremented and decremented by 1 by placing '++' or '--' at the beginning of a command line. The SET DECIMALS and SET FIXED commands can be used to specify numeric accuracy for calculations involving decimal places. The STORE command and the assignment command '=' cannot be used to modify the contents of field variables - the REPLACE command should be used for this purpose.

Recital allows any word to be used as a memory variable, but it is strongly recommended that Recital keywords are not used, as any program written in this way is more difficult to read and maintain. Memory variables can be saved in a file by issuing the SAVE TO command and restored with RESTORE FROM. Memory variable files are normal text files containing a series of STORE commands. They can be viewed and modified with MODIFY COMMAND. When Recital detects an error in a program, it will SAVE the active memory variables in a file in the current directory called "error.mem". This file also contains the information that is displayed with the DISPLAY STATUS command, and can be viewed with MODIFY COMMAND to inspect the values of the memory variables at the time the error was detected.

The DISPLAY MEMORY command can be used to inspect the current status of memory variables. There is no fixed limit to the number of memory variables that can be declared in Recital. Memory variables that

are PRIVATE to a procedure or program are automatically released when the RETURN statement is encountered. The RELEASE command can be used to release memory variables if they are no longer needed. The CLEAR MEMORY command releases all current memory variables.

Example

```
store "hello " to string1  
store string1 + "world" to string2  
? string2  
hello world
```

```
area = length * width  
area = "change to a string"
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

STORE AUTOMEM

Class

Memory Variables

Purpose

To generate memory variables corresponding to the current record

Syntax

STORE AUTOMEM

See Also

GATHER, REPLACE, SCATTER, USE

Description

The STORE AUTOMEM command is used to generate memory variables corresponding to the current record. A memory variable with a matching name, data type and length is created for each field in the current record. The memory variables are initialized with the field values.

Example

```
set locktype to optimistic
use customer
store automem
@1,1 get m.name
@2,1 get m.address
@3,1 get m.state
read
if not change()
    replace customer.name with m.name,;
    customer.address with m.address,;
    customer.state with m.state
endif
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

SUM

Class

Fields and Records

Purpose

Sum numeric expressions against records in the currently selected table

Syntax

```
SUM [<scope>] [<expN1> [,<expN2>....]]  
[TO <memvar list>]  
[FOR <condition>]  
[WHILE <condition>]
```

See Also

COUNT, TOTAL, AVERAGE, REPORT

Description

The SUM command totals the specified numeric expressions with respect to the records in the currently selected table. If <scope> is not specified, then ALL is used, unless the WHILE clause is specified, in which case the <scope> will default to REST. Up to fifteen numeric expressions can be summed. You can specify different target memory variables, with comma separators, for storing the different expressions. If SET DELETED ON is in effect, then records that are marked for deletion will not be included in the calculations. If a FILTER <condition> is active, then only those records that satisfy the <condition> will be included in the calculations. If the target memory variables do not exist, then they will be automatically created.

TO <memvar list>

If the TO <memvar list> clause is not specified, then the results are displayed on the screen only if TALK is ON. If TALK is OFF, then a <memvar list> must be specified so that the results are accessible.

FOR <condition>

If a FOR <condition> is specified, then only those records which satisfy the <condition> are included in the calculations.

WHILE <condition>

The WHILE <condition> can be used in conjunction with the FIND or SEEK commands, and the REST <scope>, to restrict the number of records which are processed and therefore optimize the performance of the SUM command.

TO ARRAY <array>

The TO ARRAY clause is used to store the SUM results in a one-dimensional array. The result of the first numeric expression is placed in the first array element; the second result is placed in the second element, and so on. If there are fewer elements than expressions, the SUM command will only store results for which there are elements. If there are more elements than expressions, the remaining elements are left empty.

Example

```
use patrons  
sum seats to m_seats;  
    for event = "BALLET" and date = date()  
? m_seats  
1680
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

SUSPEND

Class

Error Handling and Debugging

Purpose

Suspend program execution

Syntax

SUSPEND

See Also

RESUME, CANCEL, LIST HISTORY, SET HISTORY, SET ECHO, SET STEP, SET HISTORY, SET DOHISTORY, SET STEP, SET DEBUG, SET ESCAPE

Description

The SUSPEND command suspends the execution of a program, and returns control to the ">" prompt. Current memory variables are not released, and all files currently open are not closed. When a program has been suspended, you can execute Recital/4GL commands at the ">" prompt to help you in program debugging. Memory variables can be inspected and updated if required.

Program execution continues when the RESUME command is issued. If SET ESCAPE is ON, and the interrupt key is pressed, the Recital/4GL allows you to suspend program execution at that point. When debugging a program in single step mode, with the SET STEP ON command, you can also choose when you want to suspend program execution. The SET HISTORY command and the SET DOHISTORY command can be used in conjunction with SUSPEND to provide an execution 'trace' of the program. When a program is suspended, you can cancel it altogether by issuing the CANCEL command.

You cannot SUSPEND in a runtime environment. If a SUSPEND command is encountered, program execution terminates and the user is returned to the operating system.

Example

```
set history to 500
set history on
set dohistory on
on error suspend
do testprg
```

Products

Recital Terminal Developer

TEXT

Class

Input/Output

Purpose

Display a block of text on the screen or printer

Syntax

TEXT [TO <memvar> [ADDITIVE] [TEXTMERGE] [NOSHOW] [PRETEXT <expC> | <expN>]]
<text to be displayed>
ENDTEXT

See Also

TREPORT, @...SAY, SET DEVICE

Description

The TEXT and ENDTEXT commands can be used to delimit a block of text that will be output to the screen or printer. The Recital/4GL performs & macro substitution on the text unless SET MACROS is OFF. All of the text is displayed exactly as it appears within the TEXT and ENDTEXT commands.

If SET PRINT is ON, then the text will be output to the printer. The printer can be assigned to be the system printer, a location specific printer, or a printer attached to the printer port of the issuing terminal. See the SET PRINTER TO command for full details. If SET ALTERNATE is ON, then the text will be output to the alternate file.

TO <memvar>

If the optional TO <memvar> is included, the text will be sent to the memory variable specified in <memvar>. The variable will be created if it does not already exist. The following keywords can also be used in conjunction with TO <memvar>:

Keyword	Description
ADDITIVE	If ADDITIVE is specified, the text will be added to the contents of <memvar>, otherwise the contents will be overwritten.
TEXTMERGE	Operates as if SET TEXTMERGE is ON.
NOSHOW	Disables the display of the text on the screen.
PRETEXT <expC>	Inserts the specified character expression, <expC> at the start of each delimited line of text.
PRETEXT <expN>	Specifies a numeric flag (1-7) with the following values: 1 – Eliminates spaces before each delimited line. 2 – Eliminates tabs before each delimited line. 4 – Eliminates carriage returns before each delimited line. Multiple options can be specified by adding the values.

Example

```
set print on
TEXT
```

```
The following patrons attended the
BALLET event on 01/10/1999
```

```
-----
```

```
ENDTEXT
```

```
use patrons index events, dates, names
seek "BALLET"
list off name, seats, amount;
  while event = "BALLET"
set print off
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

TOTAL

Class

Fields and Records

Purpose

Total the numeric fields in the currently selected table creating a new table to contain the results

Syntax

TOTAL ON <key expression> TO <.dbf filename> | (<expC>) [<scope>]
[FIELDS <field list>]
[FOR<condition>]
[SUMMARY]
[WHILE <condition>]

See Also

SUM, AVERAGE, COUNT, CREATE REPORT, UPDATE, SET ICACHE

Description

The TOTAL command creates a new table <.dbf filename> containing a record for each unique <key expression> in the active table. The file name can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, '.dbf' is used. If no <scope> is specified, then ALL is used, unless the WHILE clause is specified, in which case the <scope> will default to REST.

FIELDS <field list>

If the FIELDS clause is specified, then only those fields specified in the <field list> will be totaled in the new table. All numeric fields copied to the new table will be totaled.

FOR <condition>

If the FOR <condition> is specified, then only those records which satisfy the <condition> will be processed.

SUMMARY

The optional SUMMARY clause will total and only copy to the new table, the fields specified in the FIELD <field list> option.

WHILE <condition>

The WHILE <condition> can be used in conjunction with the FIND, SEEK or LOCATE commands, and the REST <scope> to restrict the range of records which are processed. This can be used to optimize the performance of the TOTAL command.

If a FILTER <condition> is active, then only those records that satisfy the <condition> are processed. If SET DELETED is ON, then records that are marked for deletion are not processed.

When a table is being totaled on a particular expression, all non-numeric fields will contain the values from the first occurrence of the expression in the active table. For TOTAL to work correctly, a numeric field being totaled must be large enough to hold the totaled value.

The currently selected table does not have to be indexed or sorted on the <key expression> for TOTAL to operate. During the operation of TOTAL, the Recital/4GL builds a temporary index file for the <key expression>. If there are a large number of unique key expressions, then adjusting ICACHE will accelerate the TOTAL command. See SET ICACHE for full details.

Example

```
// Select wages
seek dtos(date())
total on dtos(pay_date) + emp_no to;
  (cmonth(date()) + strzero(day(date()),2) );
  while pay_date = date()
return
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

TREPORT

Class

Reports

Purpose

Free format text based report writer

Syntax

```
TREPORT FROM <.trf filename> | (<expC>)  
[FOOTER <expC1>, <expC2>, <expC3>]  
[HEADING <expC4>, <expC5>, <expC6>]  
[LPP <expN>]  
[TO FILE <.txt filename> | (<expC>)]  
[TO PRINT]  
[TO TERMINAL]
```

See Also

CREATE REPORT, REPORT, MODIFY COMMAND, SET DEVICE, SET PRINT, SET PRINTER, PRINT

Description

TREPORT is a free format text based report writer, which formats a file of text and commands written in the TREPORT report definition language (RDL). Reports defined in RDL can extract data from one or more tables.

FOOTER <expC1>,<expC2>,<expC3>

The FOOTER clause specifies a footer line to appear at the bottom of each report page. The character expression <expC1> represents text to appear on the left side of the footer line. Expression <expC2> represents text to appear in the center of the footer line, and <expC3> will appear on the right side of the footer line. The FOOTER clause is synonymous with the #FOOTER report definition language directive.

HEADING <expC4>,<expC5>,<expC6>

The HEADING clause specifies a heading line to appear at the top of each report page. The character expression <expC4> represents text to appear on the left side of the heading line. Expression <expC5> represents text to appear in the center of the heading line, and <expC6> will appear on the right side of the heading line. The HEADING clause is synonymous with the #HEADING report definition language directive.

LPP

The LPP clause is used to specify the number of lines per page. The numeric expression <expN> represents the number of lines. By default the number of lines is 60.

TO FILE <file>

If the TO FILE <.txt filename> clause is specified, then the report is output to a file, which can be printed later with the PRINT command. The file name can be substituted with an <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is specified, '.txt' is used.

TO PRINT

If the TO PRINT clause is specified, the report is output to the printer. The printer can be assigned to be the system printer, a location specific printer, or a printer attached to the printer port of the issuing terminal. See the SET PRINTER TO command for full details.

TREPORT operates by reading the specified <.trf filename> and formatting the free format text contained within it, as specified by the RDL directives which are intermingled within the text. The text contained in the <.trf filename> can be printed in bold or underlined. To define text to be printed in bold, surround the text with ^B and ^ (e.g. ^BHAMLET^ would be printed in bold). To define text to be printed underlined, surround the text with ^S and ^ (e.g. ^SHAMLET^) would be printed underlined. The RDL contains a wide range of directives. These directives are defined by preceding them with a '#' character. There must be no space between the '#' character and the RDL directive. RDL directives can be entered in upper or lower case.

Lines of text which do not have '#' as the first printable character, are output just as they appear in the <.trf filename>. & macro substitution is performed on each line as it is read from the <.trf filename>. You can indent directives to assist in the readability of the RDL program. You can also use Recital/4GL expressions as the arguments to RDL directives.

The following RDL layout directives are available:

#HEADING “<left>'<middle>'<right>”

The HEADING, or HD directive defines a one line heading to appear at the top of each report page. The heading is specified in three parts. The <left> part is displayed at the leftmost side of the page. The <right> part is displayed at the rightmost side of the page, and the <middle> part is displayed in the center of the page. If '#' appears in any of the heading text, it is replaced with the current page number.

#HEADING2 “<left>'<middle>'<right>”

THE HEADING2, or HD2 directive defines a one line heading to appear below the HEADING line. The heading is defined in the same way as HEADING. If the '#' character appears in any of the HEADING2 text, it is replaced with the current page number.

#HEADER_START

<lines of text>

#HEADER_END

The HEADER_START...HEADER_END, or HS...HE directives define a block of text to be displayed as a header below the HEADING line at the top of each page. & macro substitution is not performed until the header block is processed. This allows & macros to be included in the header block.

#FOOTER “<left>'<middle>'<right>”

The FOOTER, or FO directive defines a one line footer to appear at the bottom of each report page. The FOOTER text is defined in the same way as HEADING. The <left> part is displayed at the leftmost side of the page. The <right> part is displayed at the rightmost side of the page, and the <middle> part is displayed in the center of the page. If '#' appears in any of the footer text, it is replaced with the current page number.

#FOOTER_START

<lines of text>

#FOOTER_END

The FOOTER_START...FOOTER_END, or FS...FE directives define a block of text to be displayed as a footer above the FOOTER line at the bottom of each page. & macro substitution is not performed until the footer block is processed. This allows & macros to be included in the footer block.

#NEEDLINES <expN>

The NEEDLINES, or NE directive informs TREPORT that if less than <expN> lines are available at the bottom of the current page, then it should eject to the next page. This directive is used when you want to keep a block of text together on the same page, rather than split it across a page boundary.

#BLANKLINES <expN>

The BLANKLINES, or SP directive outputs <expN> blank lines in the report.

#CENTRE_ON [(<expN>)]

The CENTRE_ON, or CE directive instructs TREPORT to center all text lines from the next line onwards. The optional numeric expression <expN> may be used to specify a line length in which to center the text. The default line length for centering is 80 characters.

#CENTRE_OFF

The CENTRE_OFF, or CF directive instructs TREPORT to turn off centering of text lines.

#MACROS_ON

The MACROS_ON, or MO directive instructs TREPORT to macro substitute field names that are prefixed with the '?' character.

#MACROS_OFF

The MACROS_OFF, or MF directive instructs TREPORT to turn off macro substitution of field names prefixed with the '?' character.

#NUMBER_ON

The NUMBER_ON, or NO directive instructs TREPORT to number each remaining text line from the <.trf filename>. This directive can be used to produce program listings to include in system documentation. The line numbers restart at 1 each time NUMBER_ON is encountered.

#NUMBER_OFF

The NUMBER_OFF, or NF directive turns off line numbering which has been turned on with NUMBER_ON.

#LINESPACING <expN>

The LINESPACING or LS directive defines the spacing between lines in the report. A LINESPACING of 2 will produce a double spaced report.

#INDENT <expN>

The INDENT, or IN directive defines an indentation of <expN> spaces from the left margin. To stop the indentation, specify a second INDENT directive.

#EJECT

The EJECT, or BP directive ejects to the head of the next page. The footer is output at the bottom of the current page, and the header is displayed at the head of the next page. This directive should only be used if you specifically want to “force” a page eject, as TREPORT handles pagination automatically.

#LEFTMARGIN <expN>

The LEFTMARGIN, or PO directive defines the left margin offset where the printer report lines should start. TREPORT pads lines out to the required column before outputting them.

#LINELENGTH <expN>

The LINELENGTH, or LL directive defines the maximum width of a line of text. Lines that are wider than LINELENGTH are truncated.

#TEMPINDENT <expN>

The TEMPINDENT, or TI directive defines an indentation of <expN> for the next line only. Following processing of the next line, indentation will return to that defined with the INDENT directive.

#SUBTITLE <expC>

The SUBTITLE, or SH directive is used to output a subtitle as specified by <expC>. The SUBTITLE directive is processed as follows: if there are less than 3 lines left on the current page then eject to the next page, output one blank line, output the specified text <expC>, then output one more blank line.

#PAGELENGTH <expN>

The PAGELength, or PL directive defines the number of lines to be placed on each output page of the report. The number specified in <expN>, is reduced by 3, to leave space for HEADING, HEADING2, and FOOTER. By default, PAgELength is 60.

#PARAGRAPH

The PARAGRAPH, or PP directive is processed in the following way: if there are less than 2 lines left on the current page then eject to a new page, output one blank line, then set a temporary indentation of 4 spaces.

The following RDL processing directives are available:

#SINGLE_SHEET

The SINGLE_SHEET, or SS directive informs TREPORT that the report is to be printed on a printer which does not have an automatic sheet feeder. Whenever a new page is ejected, TREPORT asks you to place the next sheet in the printer, then press the RETURN key.

#TRANSLATE "<expC1>,<expC2>"

The TRANSLATE directive is used to translate the meaning of a significant character to a different character. The <expC1> is the character whose meaning you wish to translate. The <expC2> is the character you wish to translate to.

#INCLUDE "<.trf filename>"

The INCLUDE, or SO directive causes TREPORT to read the specified <.trf filename> and process the text or RDL directives contained within it.

#EXECUTE <command>

The EXECUTE, or EX directive is used to execute a Recital/4GL command. EXECUTE is normally used to perform report calculations. Typical commands used with EXECUTE are SKIP, SEEK, and STORE. You should not execute TREPORT from an EXECUTE directive. Any output from the Recital/4GL command executed with EXECUTE is not included in TREPORT formatting.

#IF <condition>

<text or RDL directives>

#ELSEIF

<text or RDL directives>

#ELSE

<text or RDL directives>

#ENDIF

The IF...ELSEIF...ELSE...ENDIF directives conditionally process the text and RDL directives of a <.trf filename>, depending upon the specified <condition>. The ELSE and ELSEIF blocks are optional.

IF...ELSEIF...ELSE...ENDIF may contain other IF...ELSE...ENDIF directives as well as DO WHILE...ENDDO repetition directives.

#DO WHILE <condition>

<text or RDL directives>

#ENDDO

The DO WHILE...ENDDO directives repeatedly process the <text or RDL directives> until the specified <condition> is .F. These directives are typically used in conjunction with the EXECUTE directive.

#SCAN FOR <condition>

#ENDSCAN

The SCAN FOR...ENDSCAN directives locate all records that match the <condition> and repeatedly process the <text or RDL directives> until the specified <condition> is .F.. These directives are typically used in conjunction with the EXECUTE directive.

#! <comment line>

The ! directive allows comments to be included in the <.trf filename>. Text contained on a ! line is not formatted, but can be used to improve the readability and maintainability of the RDL program.

Example

```
#!*****
#!* REPORT IDENTIFICATION SECTION*
#!*****
#! FILE : patrons.trf
#! PURPOSE: report for event HAMLET
#!
#!*****
#!* REPORT LAYOUT SECTION *
#!*****
#macros on
#execute store date() to today
#heading "HAMLET'Patron List'&today"
#heading2 "-----'"
#header_start
NAME OF                NUMBER OF                ADDRESS OF
PATRON                SEATS                PATRON
=====
=====

#header_end
#!
#footer_start

-----
Total patrons this page: &m_totpatrons
Total seats sold this page: &m_totseats
#footer_end
#footer "'- # -'"
#!
#!*****
#!* REPORT PROCESSING SECTION *
#!*****
#execute use patrons index events,dates,names
```

The following report provides details of the patrons who attended the showing of ^BHAMLET^. This is a summary report only; a full report is also available if you need it.

```
#execute store 0 to m_totpatrons
#execute store 0 to m_totseats
#ex seek "HAMLET"
#do while event="HAMLET"
#needlines 4
?name?seats ?street
?city
?postcode
#execute store m_totpatrons+1 to m_totpatrons
#execute store m_totseats+m_seats to m_totseats
#ex skip
#enddo
#execute use
#! ** End of TREPORT definition **
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

TRY...ENDTRY

Class

Error Handling and Debugging

Purpose

Structure to handle errors and exceptions within a block of code

Syntax

```
TRY  
[ <tryCommands>]  
[CATCH [ TO <memvar> ] [ WHEN <expL> ]  
    [<catchCommands>]]  
[THROW [<exp>]]  
[EXIT]  
[FINALLY  
    [<finallyCommands>]]  
ENDTRY
```

See Also

ON ERROR, ERROR(), MESSAGE()

Description

TRY...CATCH...FINALLY is a command structure to handle errors and exceptions within a block of code. The <tryCommands> which follow the TRY statement are executed. If no error occurs in the <tryCommands> program execution continues from the FINALLY statement. If an error occurs, program execution jumps immediately to the CATCH statement.

CATCH

The CATCH block with its <catchCommands> handles the error. If the optional TO <memvar> clause is specified, a reference to an Exception object is stored to <memvar>. Please see the table below for Exception Class properties. If the optional WHEN <expL> clause is specified, the <expL> is evaluated and must be equal to True (.T.) for the <catchCommands> to be processed.

THROW

The THROW command causes an error to be thrown. This is primarily used to escalate an error to an outer TRY...CATCH...FINALLY structure. If the THROW is not in a nested TRY...CATCH...FINALLY structure, the error is handled by the active ON ERROR error handler or if none is active, by the default Recital error handler, which will generate an error.mem error log. The optional <exp> is used to specify the UserValue property of a new Exception object.

EXIT

The EXIT command is used to break out of the structure and execution continues from the FINALLY block or after the ENDTRY if no FINALLY block exists or the EXIT is called from the FINALLY block itself.

FINALLY

The FINALLY block <finallyCommands> are run unless a CANCEL or QUIT has been used to exit the structure. It can be used to clean up or release any resources used in the TRY or CATCH blocks.

ENDTRY

The ENDTRY statement completes the structure.

Exception Class

PROPERTY	DESCRIPTION
BaseClass	The System base class: 'Exception'.
Class	The class: 'Exception'.
ClassLibrary	The class library: '' for System classes.
Comment	Descriptive text string.
Details	Additional details relating to the Message property value.
ErrorNo	The error number for the last error that occurred (same as ERROR()). If the Exception object is created by a THROW <exp>, ErrorNo is 2071.
LineContents	The line that caused the error (same as MESSAGE(1)).
LineNo	The number of the line that caused the error (same as LINENO()).
Message	The error message (same as MESSAGE()).
Name	The name used to reference the object: 'EXCEPTION'.
Parent	The parent object.
ParentClass	The parent object class.
Procedure	The name of the procedure in which the error occurred.
StackLevel	The stack level at which the error occurred.
Tag	Used to store any additional string data required.
UserValue	The value passed by the THROW statement, which can be used as an object reference.

Example

```
try
  use example exclusive
catch
  dialog box [Unable to open example table]
endtry
```

//Another example

```
try
  use example exclusive
catch to oExc
  if oExc.message = "ALIAS name already in use"
    select example
    exit
  else
    dialog box [Unable to open example table]
  endif
endtry
```

//Another example

```
try
  ? [Outer Try]
  try
    use example exclusive
  catch to oExc
    oExc.UserValue = "Nested CATCH message: Unable to handle"
    ?[ Nested Catch]
    ?[ Inner Exception Object: ]
    ?[ Error: ] + str(oExc.ErrorNo)
    ?[ LineNo: ] + str(oExc.LineNo)
    ?[ Message: ] + oExc.Message
    ?[ Procedure: ] + oExc.Procedure
    ?[ StackLevel: ] + str(oExc.StackLevel)
    ?[ LineContents: ] + oExc.LineContents
```

```

    ?[ UserValue: ] + oExc.UserValue
    throw oExc
finally
    ?[: Nested FINALLY executed ]
endtry
catch to oExc1
    ?[: Outer CATCH ]
    ?[ Outer Exception Object: ]
    ?[ Error: ] + str(oExc1.ErrorNo)
    ?[ LineNo: ] + str(oExc1.LineNo)
    ?[ Message: ] + oExc1.Message
    ?[ Procedure: ] + oExc1.Procedure
    ?[ StackLevel: ] + str(oExc1.StackLevel)
    ?[ LineContents: ] + oExc1.LineContents
    ?[ ->UserValue becomes inner exception THROWN from nested TRY/CATCH ]
    if oExc1.UserValue.Message = "ALIAS name already in use"
        select example
    endif
finally
    ?[: FINALLY executed ]
endtry

```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

TYPE

Class

Input/Output

Purpose

Display a text file on the screen or printer

Syntax

TYPE [FILE] <.txt filename> | (<expC>) [TO PRINT]

See Also

PRINT, ALIAS, SET PAGELength TO

Description

The TYPE command displays the specified text file on the screen. If no filename is specified, '.txt' is used. The file name can be substituted with an <expC>, enclosed in round brackets, which returns a valid filename.

FILE

If the optional FILE keyword is specified, then the text file will be displayed in pages, with line numbers, and the name of the file will be displayed as a heading above each page. The length of each page may be specified with the SET PAGELength TO command. This feature is particularly useful for producing program listings.

TO PRINT

If the optional TO PRINT clause is specified, then the file will be output to the printer. The TO PRINT option will default to a local printer unless the command SET PRINTER TO \\SPOOLER is issued. SET PRINTER TO \\SPOOLER causes the file to be spooled to the system printer defined by the environment variable DB_PRINT.

Example

```
set printer to \\spooler
file = dir("*.prg", .T.)
do while .not. empty(file)
    type file &file to print
    file = dir("*.prg", .F.)
enddo
set printer to
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

#UNDEF

Class

Memory Variables

Purpose

Stops text substitution for a constant created with #DEFINE

Syntax

#UNDEF <constant>

See Also

#DEFINE, LOCAL, PRIVATE, PUBLIC

Description

The #DEFINE command is used to define FoxPro compatible constants. Constants declared using #DEFINE can be overridden by a memory variable of the same name, but cannot be modified or manually released after their initial declaration. Constants are automatically updated if the value of <exp> changes and are released on exit from the session. The #UNDEF command stops the text substitution for a constant created with #DEFINE

Example

```
#DEFINE NEXT_LOOP
for I = 1 to NEXT_LOOP
    ? i
next
#UNDEF NEXT_LOOP
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

UNLOCK

Class

Manual Locking

Purpose

Release active table and record locks

Syntax

UNLOCK [ALL] [IN <alias>]

See Also

LOCKR, LOCKF, RLOCK(), FLOCK()

Description

The UNLOCK command releases any active file or record locks for the currently selected workarea. This command is only effective with shared tables.

ALL

If the optional ALL keyword is specified, then the active file and record locks are released for all workareas.

IN <alias>

The IN <alias> clause is used to release file or record locks in another workarea. Alias names may be assigned to tables with the USE command.

The UNLOCK command works in conjunction with the RLOCK() and FLOCK() functions, and the LOCKR and LOCKF commands. This command is supported to provide compatibility with Xbase, but is normally not needed as the Recital/4GL performs automatic file and record locking.

Example

```
set exclusive off
use patrons index events, dates, names
seek "PHANTOM"
// Lock record with dBASE III compatibility
do while not rlock()
    sleep 2
enddo
// NOTE Recital automatically locks the
// record in the EDIT command. The RLOCK()
// and UNLOCK are not really necessary.
edit
unlock
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

UPDATE

Class

Fields and Records

Purpose

Update the contents of the active table with data from another table

Syntax

```
UPDATE ON <key expression> FROM <workarea | alias>  
REPLACE <field> WITH <exp> [,<field> WITH <exp>...]  
[RANDOM]
```

See Also

TOTAL, SUM, COUNT, AVERAGE, SET RELATION, SET FILTER

Description

The UPDATE command updates fields in the active table based upon data from another table. The <key expression> must exist in both tables. The active table must be indexed on the specified <key expression> unless the RANDOM keyword is specified. The FROM table can be in any order. If a FILTER <condition> is active in the FROM workarea, then only those records which satisfy the <condition> are processed. If SET DELETED is ON, then records which are marked for deletion in the FROM workarea are not processed.

It is strongly recommended that the active table be indexed on the <key expression>, otherwise ALL records satisfying the <key expression> values in the FROM file are updated in the active table, which can be a lengthy process. The UPDATE command is primarily used to update a 'master' file from records held in a 'transaction' file.

Example

```
select b  
use newpatrons alias new  
select a  
use patrons index names, events, dates  
update on name from new;  
    replace balance with balance + new->balance
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

USE

Class

Table basics

Purpose

Open a table

Syntax

```
USE [<filename> | (<expC1>)  
[AGAIN]  
[ALIAS <name>]  
[AUTOMEM]  
[ENCRYPTION <expC2>]  
[EXCLUSIVE]  
[IN <workarea>]  
[INDEX < index filelist> | (<expC3>)]  
[NODBX]  
[NOUPDATE]  
[ORDER <.ndx file>|[TAG] <.dbx tag> [OF <.dbx filename>]]  
[ORDER BY <exp1> WHERE <exp2>]]
```

See Also

CREATE BRIDGE, CREATE VIEW, DECRYPT, ENCRYPT, SET AUTOFORMAT, SET COMPATIBLE, SET ENCRYPTION, SET FORMAT, SET INDEX, SET ORDER TO, SET VIEW

Description

The USE command opens an existing table in the currently selected workarea or in the workarea specified in the IN <workarea> option. The <filename> can be substituted with a <expC1> enclosed in round brackets that returns a valid file name. The <filename> can include the encryption key for encrypted database tables. The three part comma-separated key should be enclosed in angled brackets and appended to the filename, e.g. mytable<key_1,key_2,key_3>.

AGAIN

The AGAIN keyword is used to open an already open table in another workarea. The compatibility mode must be set to an alternative Xbase mode. Please see the SET COMPATIBLE command for more information.

ALIAS <name>

An optional alias name can be specified for the workarea with the ALIAS keyword. Once a table is opened in a workarea, the workarea can be identified using any of the following 'aliases': the workarea letter (A-Z) excluding M; the specified ALIAS name, or if none is specified, the first 32 characters of the file name

AUTOMEM

An empty memory variable of corresponding name, data type and size will be created for each field from the table.

ENCRYPTION <expC2>

The ENCRYPTION <expC2> clause is used to specify the encryption key for encrypted tables. The <expC2> is a string containing a three part comma-separated key, e.g. "key_1,key_2,key_3". The key may optionally be enclosed in angled brackets, e.g. "<key_1,key_2,key_3>". ". The SET ENCRYPTION command allows a default encryption key to be defined. If the ENCRYPTION <expC2> clause is not specified and the key is not included in the <filename>, this default key will be used. If the default key is not the correct key for the table, an error will be given. If no default key is active, a dialog box will be displayed in Recital Terminal Developer to allow the user to enter the key.

EXCLUSIVE

If the EXCLUSIVE keyword is specified, then the table is opened for private use, disallowing access by other users. To open a table for shared use, issue the SET EXCLUSIVE OFF command before the USE command.

IN <alias>

If the IN keyword is specified then the table is opened in the workarea specified by alias. Alias can be either: the workarea letter (A-Z) excluding M or the workarea number. Specifying 0 as the workarea number causes the table to be opened in the lowest free workarea.

INDEX <index filelist>

The INDEX < index filelist> clause specifies a list of index files which should be opened and associated with the table. The <index filelist> may contain both single (.ndx), and multiple (.dbx) files. The first index in the list is known as the master index file, and is used to search for key values with the SEEK and FIND commands. The SET ORDER TO command can be used to reselect a master index from the list of open index files. Index files are created within the CREATE and MODIFY STRUCTURE Terminal Developer Development Tools, or using the INDEX ON command.

NODBX

If the NODBX keyword is specified then the table is opened without its associated production index file. All index definitions in the table header are released.

NOUPDATE

When NOUPDATE is specified, the table and related index file are opened with read only access. When used with an RMS bridge on OpenVMS, the RMS file is opened with 'shared read' access instead of 'shared read-write' access.

ORDER <.ndx file>|[TAG] <.dbx tag> [OF <.dbx filename>]

The ORDER qualifier can be used to specify the master index. The order can be specified by using the name of an .ndx file or by specifying the name of a tag. The OF <.dbx filename> may be used to explicitly specify the .dbx file to use.

ORDER BY <exp1> where <exp2>

This format of the USE command is only available for database gateways. When the ORDER BY <exp1> clause is used, the data file on the information server is initially opened and ordered by the expression specified. When the WHERE <exp2> clause is used, the data file on the server will extract only records that meet the criteria defined by <exp2>.

All open files associated with the currently selected workarea are closed before any new files are opened. The USE command on its own closes all open files associated with the currently selected workarea. After a table has been opened, the record pointer is positioned to record number 1 if no index files have been specified, or to the first record in the master index if the table has been opened with associated index files.

Please see the SQL USE command for information on the MySQL compatible USE <database> usage.

Example

```
use accounts index address.dbx;  
  order tag last of address;  
  in 3
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

VI

Class

Terminal Developer Development Tools

Purpose

Execute a text editor to edit program files

Syntax

VI <prg filename> | (<expC>)

See Also

MODIFY COMMAND, TEXTEDIT(), ED

Description

VI provides the facility to create or modify program files and other text files. The filename can be substituted with a <expC>, enclosed in round brackets, which returns a valid filename. If no file extension is present in the file name, then '.prg' is used.

The default editors are: the 'vi' editor under UNIX and Linux; the 'edt' editor under VAX/VMS. You may override these defaults using the SET TEDIT TO command.

VI is a synonym of the MODIFY COMMAND and ED commands.

Example

```
modify command myprogram  
ed myprogram  
vi myprogram
```

Products

Recital Terminal Developer

WAIT

Class

Keyboard Events

Purpose

Suspend program execution until a key is pressed at the keyboard

Syntax

WAIT [<expC>]
[CLEAR]
[NOWAIT]
[TO <memvar>]
[TIMEOUT <expN>]
[WINDOW]

See Also

DIALOG BOX, MESSAGE, ACCEPT, @...GET, READ, INPUT

Description

The WAIT command displays the specified prompt <expC> on the screen and suspends program execution until a key is pressed. If no <expC> is specified, then “Press any key to continue...” is displayed. The key that is read from the keyboard is not echoed.

CLEAR

The CLEAR keyword will remove a system window or window message when the WAIT command is called from a program.

NOWAIT

The NOWAIT keyword will create a message like the Recital/4GL system message, which is displayed in the upper right hand corner. A message created with the NOWAIT keyword does not discard the keystroke you use to remove the message.

TIMEOUT <expN>

The TIMEOUT clause lets you specify the number of seconds, <expN>, the message will remain on the screen before the program execution continues. This is provided for Xbase language compatibility only.

TO <memvar>

If the optional TO <memvar> clause is specified, then the key that is pressed is stored as a character string in the designated memory variable. If the [RETURN] key is pressed, or the key which is pressed does not represent a printable character, then a null string, “”, is stored in the memory variable. If the memory variable does not exist, it is created.

WINDOW

The WINDOW keyword will cause the message to be displayed in the system message window.

Example

```
@23,0  
wait “Enter your selection...” to option
```

Products

Recital Mirage Server, Recital Terminal Developer

WITH...ENDWITH

Class

Objects

Purpose

Specify multiple properties for an object

Syntax

```
WITH <object>  
    <property definitions>  
ENDWITH
```

See Also

DEFINE CLASS...ENDDEFINE, CLASS

Description

The WITH...ENDWITH block is used to specify multiple properties for an object. The object name is specified in <object>. The <property definitions> consists of property assignments. The properties are specified using preceding dot notation.

Example

```
with oProduct  
    .productname = "Recital Mirage"  
    .productversion = "2.0"  
    .productyear = "20001"  
endwith
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer

ZAP

Class

Fields and Records

Purpose

Permanently remove all records from the active table

Syntax

ZAP

See Also

DELETE, PACK, SEQNO(), SET SAFETY, SET SEQNO

Description

The ZAP command permanently removes all of the records from the active table, without affecting the table structure. ZAP is equivalent to DELETE ALL, followed by PACK. The ZAP command can only be performed on a table that is open for exclusive use. The ZAP command is typically used to reinitialize a 'transaction' table, or a journal file. If SET SAFETY is ON, a message prompting for confirmation of the operation will be displayed before the ZAP can take place. ZAP also resets the table SEQNO to zero.

Any index files that have been associated with the active table are also reinitialized. Once a table has been 'zapped', there is no way of recovering the records that were previously contained in it. You should always use this command with caution. Permission to use the ZAP command on a table can be disabled. The ADMIN field in the <SECURITY> option from the menu bar of the CREATE, MODIFY STRUCTURE table work surface is used for this purpose.

Example

```
set exclusive on
use newpatrons
zap
```

Products

Recital Database Server, Recital Mirage Server, Recital Terminal Developer